Integration of Network Security Mechanisms in Softwarized Networks with Data Plane Programming and Software-Defined Networking

Dissertation
der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Frederik Hauser
aus Nagold

Tübingen
2021

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	28.04.2022
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr. Michael Menth
2. Berichterstatter:	Prof. Dr. Wolfgang Küchlin
3. Berichterstatter:	Prof. Dr. Kurt Rothermel

Contents

List of Abbreviations

AA	authentication and authorization
AAM	authentication and authorization module
ACL	access-control list
AH	Authentication Header
API	application programming interface
ASIC	application-specific integrated circuit
BIER	Bit Index Explicit Replication
bmv2	Behavioral Model version 2
CLI	command line interface
DTMC	discrete-time Markov chain
EAP	Extensible Authentication Protocol
ESP	Encrypted Security Payload
FPGA	field programmable gate array
GUI	graphical user interface
IDS	intrusion detection system
IKE	Internet Key Exchange
INT	in-band network telemetry
IP	Internet protocol
IPsec	Internet Protocol Security
LLDP	Link Layer Discovery Protocol
LPM	longest prefix match
MA	moving average
MACsec	Media Access Control Security
MAT	match-action-table
MH	moving histogram
NAC	network access control
NFV	network function virtualization
NIC	network interface card
NPU	network processing unit
OF	OpenFlow
ONF	Open Networking Foundation
OS	operating system
PISA	protocol-independent switching architecture
PPF	packet processing function
RAC	restricted application container
RADIUS	Remote Authentication Dial-In User Service
SA	security association

SAD	security association database
SC	secure channel
SDN	software-defined networking
SP	security policy
SPD	security policy database
TDRM	time-dependent rate measurement
TLS	Transport Layer Security
UEMA	Unbiased Exponential Moving Average
UI	user interface
UTEMA	Unbiased Time-Exponential Moving Average
VM	virtual machine
VPN	virtual private network
VSA	vendor-specific attribute

Danksagung

Mein mit Abstand größter Dank gilt meinem Doktorvater Prof. Dr. Michael Menth, der mich in allen Phasen der vergangenen Jahre eng begleitet hat. Als Förderer aber auch als ”Antreiber” mit großem Vertrauen machte er viele Erfahrungen und Erfolge möglich. Wenn es Schwierigkeiten gab, konnte ich mich immer auf seine Unterstützung verlassen. Neben der Promotion ermöglichte er Mark Schmidt und mir den Aufbau einer gemeinsamen Lehrveranstaltung und unterstützte mich bei der Absolvierung einer Hochschuldidaktikausbildung. Auch nach meiner Zeit am Lehrstuhl zeigte er mir viele Wege auf, um weiter aktiv in Forschung und Lehre bleiben zu können.

Ein besonderer Dank gilt auch Prof. Dr. Wolfgang Küchlin und Prof. Dr. Kurt Rothermel, welche die Begutachtung meiner Dissertation übernahmen. Prof. Dr. Thomas Walter und Jun. Prof. Dr. Setareh Maghsudi möchte ich für ihren Einsatz als Prüfer bei meiner Disputation danken.

Meine Dissertation ist das Ergebnis einer intensiven Zusammenarbeit mit Kolleg:innen und Student:innen. Namentlich möchte ich mich besonders bei meinen ehemaligen Kolleg:innen Wolfgang Braun, Gülşen Ergün-Karagkiozidou, Marco Häberle, Florian Heimgärtner, Michael Höfling, Steffen Lindner, Daniel Merling, Mark Schmidt, Andreas Stockmayer und Thomas Stüber bedanken. Ohne die gute Zusammenarbeit, gegenseitige Hilfe und freundschaftliche Atmosphäre würde es diese Dissertation nicht geben.

Bedanken möchte ich mich auch bei meiner Familie und meinen Freunden, die mir immer Unterstützung und Rückhalt gegeben sowie Verständnis entgegengebracht haben. Mein abschließender Dank gilt meiner Partnerin Anna. Ohne Dein großes Verständnis in den besonders arbeitsintensiven Phasen und Deine unermüdlichen Motivationsanstöße wäre es nicht möglich gewesen.

Summary

Abstract

In network softwarization, traditional network appliances with fixed features and limited configurability are replaced by programmable software- or hardware-based platforms. Two popular concepts of this new trend are software-defined networking (SDN) and data plane programming. SDN allows programmers to bypass the control plane of networking devices and introduce own software-based control plane algorithms. Data plane programming extends this programmability to the data plane. These new networking concepts are the basis for next-generation networks as utilized in cloud computing or 5G. OpenFlow (OF) and P4 are the most widespread standards for SDN and data plane programming, respectively.

However, introducing SDN and data plane programming in existing networks requires transition strategies for the integration of existing network functions, protocols, and applications. The research of this thesis focuses on the integration of network security functions. It investigates whether existing and widespread network security mechanisms are implementable, how potential concepts and architectures of integrations may be engineered, and if mechanisms can benefit from SDN and data plane programming in terms of more efficient operation with automation, increased security, or new features. Subsequently, this research is complemented with a literature study analyzing how data plane programming with P4 is applied in fields other than network security.

The results of my research are covered in five accepted and peer-reviewed papers and two papers that are currently in peer-review. Research results on OF-based SDN include an integration of 802.1X and a novel mechanism for network access and execution control for containerized applications. Research results on P4 data planes with SDN control include integrations of MACsec, IPsec, and 802.1X. The results of the literature study are covered in an extensive survey paper. Five more accepted and peer-reviewed papers are additional content of this thesis. These publications include research results on SDN transition strategies not related to network security and research results in the field of modelling and simulation.

The majority of my research work was part of the bwNET100G+ research project [bwnb]. Additional research work was funded by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/1-2 and by Siemens AG.

Kurzfassung

Unter Network Softwarization versteht man den Trend von traditionellen Netzwerkgeräten mit fest definiertem Funktionsumfang und beschränkter Konfigurierbarkeit hin zu programmierbaren Software- oder Hardware-Plattformen. Zwei populäre Ausprägungen sind Software-Defined Networking (SDN) und programmierbare Data Planes. SDN erlaubt es, die Control Plane von Netzwerkgeräten zu umgehen und eigene softwarebasierte Control-Plane-Algorithmen einzuführen. Programmierbare Data Planes erweitern diese Flexibilität auf die Data Plane. Diese neuen Konzepte sind die Basis für Netzwerke der nächsten Generation, wie sie beispielsweise in Cloud Computing oder 5G benötigt werden. OpenFlow (OF) und P4 sind die am weitesten verbreiteten Standards für SDN und programmierbare Data Planes.

Für die Einführung von SDN und programmierbaren Data Planes in existierenden Netzwerken werden Übergangsstrategien für die Integration von existierenden Netzwerkfunktionen, -protokollen und -programmen benötigt. Die Forschungsarbeiten dieser Dissertation widmen sich der Integration von Netzwerksicherheitsfunktionen in neuen auf SDN und programmierbaren Data Planes basierenden Netzen. Es wird analysiert, inwiefern existierende und weitverbreitete Netzwerksicherheitsmechanismen implementiert werden können. Ferner werden potenzielle Konzepte und Architekturen für die Einbindung dieser Netzwerksicherheitsmechanismen entwickelt. Es wird untersucht, ob SDN und programmierbare Data Planes einen effizienteren Betrieb durch Automation, erhöhte Sicherheit, oder neue Funktionalitäten schaffen können. Die Forschungsarbeit wird durch eine Literaturstudie zu programmierbaren Data Planes mit P4 komplettiert. In dieser Studie wird die Anwendung von P4 in anderen Bereichen außerhalb der Netzwerksicherheit untersucht.

Kernergebnisse meiner Forschungsarbeit sind in fünf Peer-Reviews-akzeptierten Veröffentlichungen zusammengefasst. Zwei weitere Veröffentlichungen befinden sich derzeit im

Peer-Review. Ergebnisse meiner Forschung zu OF-basiertem SDN umfassen eine Integration von 802.1X sowie einen neuartigen Mechanismus für eine Netzwerkzugriffs- und Ausführungskontrolle für containerbasierte Anwendungen. Forschungsergebnisse zu programmierbaren Data Planes mit P4 und SDN-basierter Steuerung umfassen Integrationen von MACsec, IPsec und 802.1X. Die Resultate der Literaturstudie sind Teil einer umfangreichen Übersichtsveröffentlichung. Fünf weitere nach Peer-Review akzeptierte Veröffentlichungen sind ergänzender Inhalt dieser Dissertation. Diese Arbeiten beschreiben Forschungsergebnisse zu Übergangsstrategien für SDN, die nicht in den Bereich der Netzwerksicherheit fallen. Zudem werden Forschungsergebnisse aus dem Bereich der Modellierung und Simulation von Kommunikationsnetzen vorgestellt.

Die Mehrheit meiner Forschungsarbeit entstand im bwNET100G+ Forschungsprojekt [bwnb]. Weitere Forschungsarbeiten wurden von der Deutschen Forschungsgemeinschaft (DFG) im Rahmen der Forschungsförderung ME2727/1-2 und von der Siemens AG gefördert.

List of Publications

The personal contributions of all publications (§ 6 Abs. 2 Satz 3 der Promotionsordnung) can be found in the appendix.

Accepted Manuscripts (Core Content)

1. Frederik Hauser, Marco Häberle, Mark Schmidt, and Michael Menth. **P4-IPsec: Site-to-Site and Host-to-Site VPN With IPsec in P4-Based SDN** [HHSM20]. *IEEE Access* journal, vol. 8, pp. 139567-139586, 2020. The published version of this publication can be found in the Appendix 1.1. The paper is also available online at the following URL: `https://doi.org/10.1109/ACCESS.2020.3012738`

2. Frederik Hauser, Mark Schmidt, Marco Häberle, and Michael Menth. **P4-MACsec: Dynamic Topology Monitoring and Data Layer Protection With MACsec in P4-Based SDN** [HSHM20]. *IEEE Access* journal, vol. 8, pp. 58845-58858, 2020. The published version of this publication can be found in the Appendix 1.2. The paper is also available online at the following URL: `https://doi.org/10.1109/ACCESS.2020.2982859`

3. Frederik Hauser, Mark Schmidt, and Michael Menth. **xRAC: Execution and Access Control for Restricted Application Containers on Managed Hosts** [HSM20]. Proceedings of the 32nd *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Budapest, Hungary, 2020, pp. 1-9. The published version of this publication can be found in the Appendix 1.3. The paper is also available online at the following URL: `https://doi.org/10.1109/NOMS47738.2020.9110380`

4. Frederik Hauser and Michael Menth. **Demo: Execution and Access Control for Restricted Application Containers on Managed Hosts (xRAC)** [HM20].

Proceedings of the 32nd *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Budapest, Hungary, 2020, pp. 1-2. The published version of this publication can be found in the Appendix 1.4. The paper is also available online at the following URL: `https://doi.org/10.1109/NOMS47738.2020.9110350`

5. Frederik Hauser, Mark Schmidt, and Michael Menth. **Establishing a session database for SDN using 802.1X and multiple authentication resources** [HSM17b]. Proceedings of the 53rd *IEEE International Conference on Communications (ICC)*, Paris, France, 2017, pp. 1-7. The published version of this publication can be found in the Appendix 1.5. The paper is also available online at the following URL: `https://doi.org/10.1109/ICC.2017.7997200`.

Submitted Manuscripts (Core Content)

6. Frederik Hauser, Marco Häberle, and Michael Menth. **P4sec: Automated Deployment of 802.1X, IPsec, and MACsec Network Protection in P4-Based SDN** [HHM21a]. Initial submission to the *IEEE Network* magazine on 2021-03-31. Revised versions of the paper were submitted on 2021-11-26 and 2022-04-13 and are still under review. The most recent version of this publication can be found in the Appendix 2.1.

7. Frederik Hauser, Marco Häberle, Daniel Merling, Steffen Lindner, Vladimir Gurevich, Florian Zeiger, Reinhard Frank, and Michael Menth. **A Survey on Data Plane Programming with P4: Fundamentals, Advances, and Applied Research** [HHM$^+$21b]. Submission to the *Elsevier Journal of Network and Computer Applications (JNCA)* on 2021-08-05. A revised version of the paper was submitted on 2022-03-10 and is still under review. The most recent version of this publication can be found in the Appendix 2.2. It is also available as preprint on arXiv: `https://arxiv.org/abs/2101.10632`.

Accepted Manuscripts (Additional Content)

8. Mark Schmidt, Frederik Hauser, Bastian Germann, and Michael Menth. **LoCoSDN: A Local Controller for Operation of Switches in Non-SDN Networks** [SHGM18]. Proceedings of the 5th *International Conference on Soft-*

ware Defined Systems (SDS), Barcelona, Spain, 2018, pp. 1-7. The published version of this publication can be found in the Appendix 3.1. The paper is also available online at the following URL: `https://doi.org/10.1109/SDS.2018.8370426`.

9. Frederik Hauser, Mark Schmidt, and Michael Menth. **A Master Course on Network Softwarization: Lectures and Practical Assignments** [HSM17a]. In Proceedings of the 1st *KuVS Fachgespräch "Network Softwarization – From Research to Application"*, Tübingen, Germany, 2017. The published version of this publication can be found in the Appendix 3.2. The paper is also available online at the following URL: `https://doi.org/10.15496/publikation-19571`.

10. Michael Menth and Frederik Hauser. **On Moving Averages, Histograms and Time-Dependent Rates for Online Measurement** [MH17b]. In Proceedings of the 8th *ACM/SPEC International Conference on Performance Engineering (ICPE)*, L'Aquila, Italy, 2017, Pages 103–114. The published version of this publication can be found in the Appendix 3.3. The paper is also available online at the following URL: `https://doi.org/10.1145/3030207.3030212`.

11. Michael Menth and Frederik Hauser. **Demo: Time Series Online Measurement for Python (TSOMpy)** [MH17a]. In Proceedings of the 8th *ACM/SPEC International Conference on Performance Engineering (ICPE)*, L'Aquila, Italy, 2017, Pages 175–176. The published version of this publication can be found in the Appendix 3.4. The paper is also available online at the following URL: `https://doi.org/10.1145/3030207.3053673`.

12. Frederik Hauser, Dominik Krauß, and Michael Menth. **FunSpec4DTMC – A Tool for Modelling Discrete-Time Markov Chains Using Functional Specification** [HKM18]. In Proceedings of the 19th *International GI/ITG Conference on Measurement, Modelling and Evaluation of Computing Systems (MMB)*, Erlangen, Germany, 2018. The published version of this publication can be found in the Appendix 3.5. The paper is also available online at the following URL: `https://doi.org/10.1007/978-3-319-74947-1_28`.

1 Introduction & Overview

Network softwarization is currently among the most relevant research fields in communication networks. It disrupts the status quo to cope with increasing demands towards flexibility, availability, bandwidths, and latencies of next-generation communication networks, e.g., as required by Internet of things, Industry 4.0, cloud computing, or 5G.

In traditional networking, hardware appliances such as switches, routers, or firewalls comprise a particular set of features. Those appliances receive an initial configuration within deployment that is adjusted over time. Functionality and configurability is limited to the supported set of features, i.e., if additional functionality is required, appliances need to be replaced or complemented by extra devices. Network softwarization aims at substituting those appliances by software- or hardware-based platforms. Functionality is only part of programs that can be acquired or developed by users. By deploying new or updated programs, the functionality of the network can be entirely changed without altering the underlying platform.

1.1 Software-Defined Networking and Data Plane Programming

Software-defined networking (SDN) and data plane programming are two novel concepts of network softwarization and the main research focus of this thesis. Figure 1.1 depicts both concepts in comparison to traditional networking.

Network appliances in traditional networking consist of a data plane and a control plane. The data plane performs high-speed packet processing and is steered by the control plane that implements functions, protocols, and interfaces. Users configure the functionality of the network appliances via command line interfaces (CLIs), web user interfaces (UIs), or application programming interfaces (APIs).

SDN splits up the close bond between control plane and data plane. Therefore, the data plane is extended by an API so that programmers can bypass the control plane of networking devices and introduce own software-based control plane algorithms. Decentral control with complex peer-to-peer protocols can be replaced by potentially simpler control through a central SDN control plane that has an encompassing view on the network. OpenFlow (OF) [MAB+08] is the most widespread SDN architecture and protocol that initiated broad application of SDN in academia and industry [onf].

However, as both the data plane functionality and API are predefined and not changeable by the users, programmability is still limited. Data plane programming overcomes this limitation by extending programmability to the data plane. Users can introduce own packet processing algorithms and data plane APIs that can be used by an SDN control plane. P4 (Programming Protocol-Independent Packet Processors) [BDG+14] is the currently most widespread concept, abstraction, and language for data plane programming.

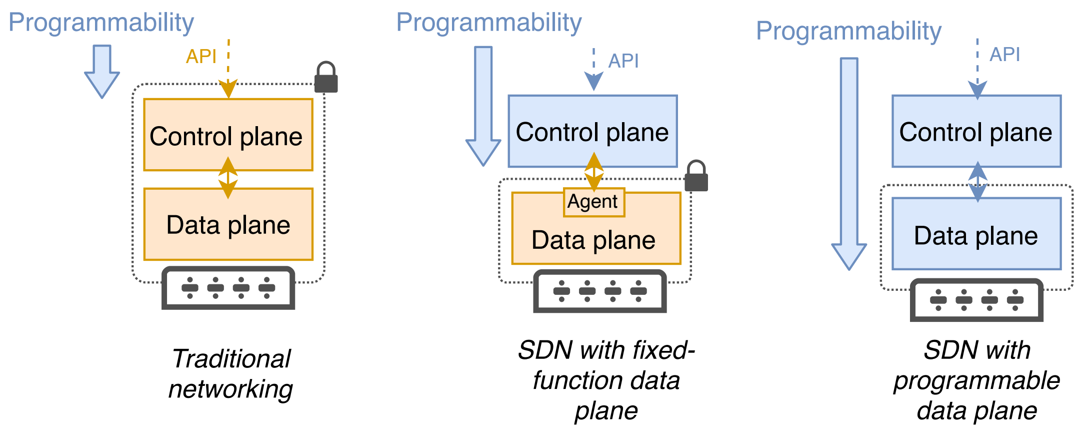


Figure 1.1: Comparison between traditional networking, SDN with fixed-function data planes, and SDN with programmable data planes (taken from [HHM+21b]).

In particular, SDN with programmable data planes entirely changes how communication networks can be built and operated. It introduces full flexibility so that algorithms, protocols, and features on all parts – the data plane, the control plane, and the API in between – can be defined by the user. Due to the availability of sophisticated hardware platforms, not only manufacturers but also users have the opportunity to design and implement custom data plane algorithms and run them on high-speed hardware. This opens up many perspectives for network innovations in academia and industry.

1.2 Research Objectives

The transition to SDN and data plane programming in existing networks raises several open issues that need to be resolved. Finding ways to integrate existing network functions, protocols, and applications in this new paradigm is one of the issues.

Network security is one particular field of network functions, protocols, and applications. Especially next-generation networks as applied in cloud computing or 5G are critical infrastructure and thereby attractive targets for cyberattacks. However, SDN and data plane programming combined with the requirements of those networks complicates or even hinders the usage of conventional network security mechanisms.

Several research studies propose novel security mechanisms that are exclusively engineered for SDN. My research follows a different approach. My three *main research objectives* were to investigate whether existing and widespread network security mechanisms are implementable in SDN and data plane programming, how potential concepts and architectures of integrations may be engineered, and if mechanisms can benefit from SDN in terms of more efficient operation, increased security, or novel features.

In contrast to novel security mechanisms designed for SDN, existing network security mechanisms profit from cross-vendor standardization, widespread use, and large operational experiences. Standard-compliant integrations allow transition strategies where SDN can be incrementally deployed within existing infrastructures.

My *subsequent research objective* was to analyze the state-of-the-art in data plane programming with P4 by an extensive literature study. The concrete goals were to review fundamentals and advancements of the technology and analyze how P4 is applied in application domains other than network security.

1.3 Research Context

The majority of my research was part of the bwNET100G+ research project [bwnb]. It was initiated in parallel with a line capacity expansion in the BelWü state research network in Baden-Württemberg from 10G to 100G. Current network concepts and systems should be reviewed and potentially redesigned to take advantage of the bandwidth increase. The project consortium includes the BelWü network provider, research groups on communication networks, and the computational centers of the Universities of Tübingen, Karlsruhe Institute of Technology, and University of Ulm. Our group conducted research on methods for flexible and intelligent network operation with the help of SDN (see [bwna]). After its completion in 2019, the project was continued under the new name bwNET2020+ [bwnc].

Additional research work that is part of this thesis was funded by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/1-2. The basis of the literature study on data plane programming with P4 took part in the "P4-SOTA" project that was funded by Siemens AG.

All research works were conducted with colleagues that were also part of the respective research projects. A description of my contributions and their shares to the overall work can be found in the appendix.

1.4 Research Results

The results of my research are part of 12 publications that can be found in the appendix. Chapter 2 covers my research results that are divided into five parts. For each part, technological foundations are introduced, results are outlined, and the significance of the results are described. An overview of the five parts and research results is given in the following.

The first part covers research works on **network access control (NAC) with 802.1X in OF-based SDN** (Section 2.1). One contribution is the integration of 802.1X for OF-based SDN [HSM17b]. We present the authentication and authorization module (AAM), a standard-compliant 802.1X authenticator implemented as SDN application to be executed on an SDN controller. To resolve shortcomings of 802.1X, it introduces support for multiple and potentially non-RADIUS authentication and authorization (AA) resources and integrates a session database for stateful tracking of user sessions to prevent account sharing. xRAC [HSM20, HM20] leverages 802.1X and OF-based SDN to extend NAC and execution control to applications. Therefore, applications are encapsulated in Docker virtualization containers, equipped with a dedicated IP address for traffic control, and executed on a managed host. AA ensures that only permitted users can start a particular application and that the application can only reach permitted network resources. This allows fine-grained control as required in high-security environments.

The second part comprises research works on the **integrations of MACsec, IPsec, and 802.1X on P4 data planes with SDN control** (Section 2.2.4). P4-MACsec [HSHM20] introduces MACsec for protecting the Ethernet layer with authentication and encryption on P4 data planes with SDN control. Instead of time-consuming setup, network links between switches are detected and monitored with a novel secured variant of LLDP, and MACsec protection is automatically deployed and operated on all detected links. P4-IPsec [HHSM20] introduces IPsec for protecting the IP layer with VPN tunnels on P4 data planes with SDN control. It follows a similar approach as in P4-MACsec and facilitates automated deployment and operation of site-to-site and host-to-site IPsec VPN tunnels. P4sec [HHM21a] combines the mechanisms of P4-MACsec and P4-IPsec, and integrates NAC with 802.1X in one system for automated deployment and operation of network security on P4 data planes with SDN control. To support distributed campus and enterprise networks with multiple sites and remote users, a novel three-tier control plane is implemented.

The third part comprises a **literature study on data plane programming with P4** (Section 2.3). The results are part of an extensive survey [HHM^{+}21b] that covers 519 scientific publications, white papers, standards, websites, and source code repositories. It presents fundamentals and advancements, gives a review of research studies that have been conducted in various application domains, and provides an extensive discussion and outlook on the technology and its current application.

The research results covered in the last two parts are additional content of this thesis. The fourth part presents research works on **transition strategies for introducing SDN** (Section 2.4). LoCoSDN [SHGM18] tries to mimic the function sets and interfaces of traditional network appliances by adding a local SDN controller to every SDN switch. The local SDN controller implements common network functionality such as L2 switching or L3 routing and provides a CLI for administrator control. The novel master's course "Network Softwarization" [HSM17a] proposes a teaching concept for university students as preparation for theses and student research projects. The fifth part includes research works on **modelling and simulation** (Section 2.5). One contribution are time-dependent statistics for online measurement [MH17b] and their implementation as Python program [HKM18]. Another contribution is FunSpec4DTMC [HKM18], a Python tool for modelling and simulation of DTMCs.

2 Results & Discussion

This chapter presents and discusses the results of this thesis. Section 2.1 presents research on *next-generation NAC with 802.1X*. In Section 2.2, research outcomes on *network security in Enterprise and Campus Networks with P4* are described. Section 2.3 covers the results of a *literature study on data plane programming with P4*. In Section 2.4, research works on *SDN transition strategies* are presented. Section 2.5 comprises research from the field of *modelling and simulation*. The results covered in the last two sections are additional content of this thesis.

2.1 Next-Generation Network Access Control with 802.1X

This section covers research on NAC with 802.1X. It introduces the foundations of 802.1X (Section 2.1.1) and control plane SDN with OF (Section 2.1.2). Afterwards, the research results of 802.1X in OF SDN (Section 2.1.3) and xRAC (Section 2.1.4) are presented.

2.1.1 NAC with 802.1X

802.1X [IEE20] is a widespread IEEE standard for port-based network access control (NAC). Network devices such as computers, printers, or IP phones are only granted network access after successful authentication and authorization (AA). The 802.1X standard was first proposed for use in IEEE 802.3 (LAN) networks [IEE01]. Later it was updated to also cover IEEE 802.11 (WLAN) networks [IEE04].

Figure 2.1 depicts the three components and the working principle of NAC with 802.1X. The *802.1X supplicant* is a software module running on the network device that requests access to a protected network. It is configured with credentials (e.g., username and password) for authentication so that AA is carried out automatically in the background without user interaction. The *802.1X authentication server* authenticates the user by verifying the provided credentials and responds with the authorization decision. The required AA data is either stored locally or accessed on a remote AA resource. The *802.1X authenticator* is part of the network access device, e.g., a switch at the edge of the network. It relays authentication between the 802.1X supplicant and 802.1X authentication server and enforces the network access decision based on the authorization response from the 802.1X authentication server. 802.1X relies on the Extensible Authentication Protocol (EAP) and Remote Authentication Dial-In User Service (RADIUS) that are both standardized in the IETF. Between the 802.1X supplicant and 802.1X authenticator, EAP data is transferred via LAN frames (EAPoL). Between the 802.1X authenticator and 802.1X authentication server, EAP data is transferred via RADIUS frames.

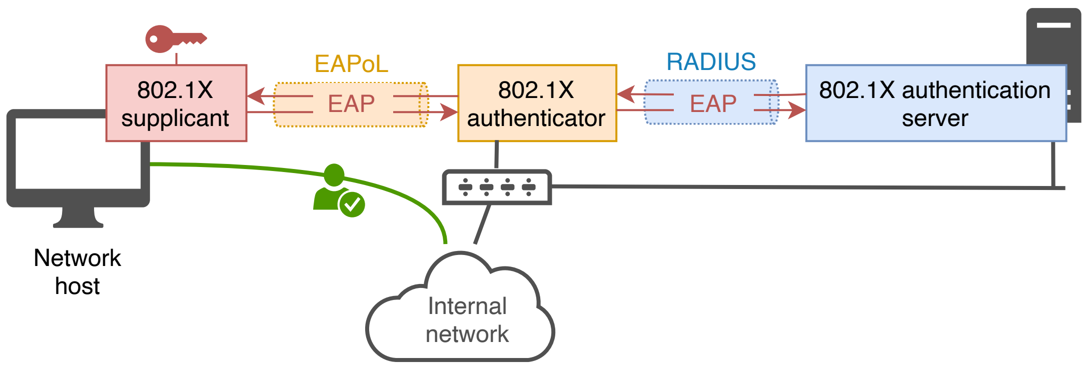
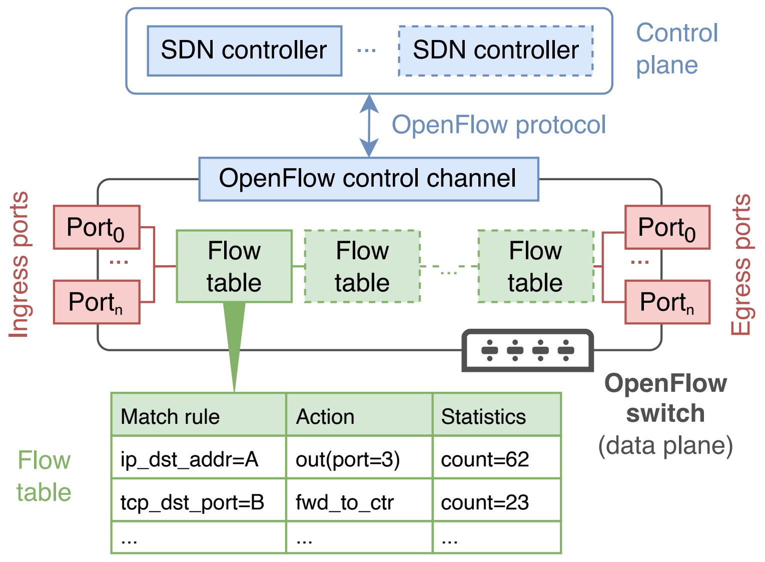


Figure 2.1: Three components and working principle of NAC with 802.1X (similar to [802]).

Details of 802.1X can be found in our papers on 802.1X for OF SDN [HSM17b] (Appendix 1.5) and xRAC [HSM20] (Appendix 1.3).

2.1.2 Control Plane SDN with OpenFlow

OpenFlow (OF) [MAB$^+$08] is the most widespread architecture and protocol for control plane programming in SDN. Users integrate custom control plane functionality that is executed on one or more SDN controllers. Those SDN controllers manage the data plane that consist of OF switches. OF switches have a fixed set of functions and an OF API for runtime programmability. OF switch functionality and the OF protocol are standardized in the Open Networking Foundation (ONF). Its most recent specification (v1.5.1) was released in 2015 [of-]. The survey of Braun and Menth [BM14] provides an encompassing overview.



Match rule	Action	Statistics
ip_dst_addr=A	out(port=3)	count=62
tcp_dst_port=B	fwd_to_ctr	count=23
...	...	...

Figure 2.2: Functional principle and architecture of OpenFlow (OF) (similar to [of-]).

Figure 2.2 depicts OF's functional principle. OF switches include a pipeline of flow tables that might comprise one or multiple flow tables. Each flow table has flow entries that consist of a match rule, an action, and statistics. Match rules are built by defining match conditions (e.g., fixed numbers, wildcard operators) for a fixed set of packet header fields (e.g., IP addresses, TCP ports) and assigning actions to them. Actions in OF include sending out packets on a particular port, forwarding packets to the SDN controller, or modifying packet headers. Statistics record information about the particular flow rule entry, e.g., the number of flow entry hits. The SDN controller manages the flow tables of OF switches, processes packets received from the OF switches, and outputs processed packets to the OF switches.

OF switches are available in software and hardware; the latter fall into two categories. *OF-only* switches exclusively support OF operation, i.e., all functions need to be implemented using the OF switch functionality and a software-based control plane. *OF-hybrid* switches offer OF programmability besides the normal switch operation, i.e., some functions of the legacy control plane (e.g., L2 forwarding) might be used in conjunction with OF (e.g., L3 routing with a custom routing protocol) and a software-based control plane. More details on OF switches can be found in our paper on LoCoSDN [SHGM18] (Appendix 3.1).

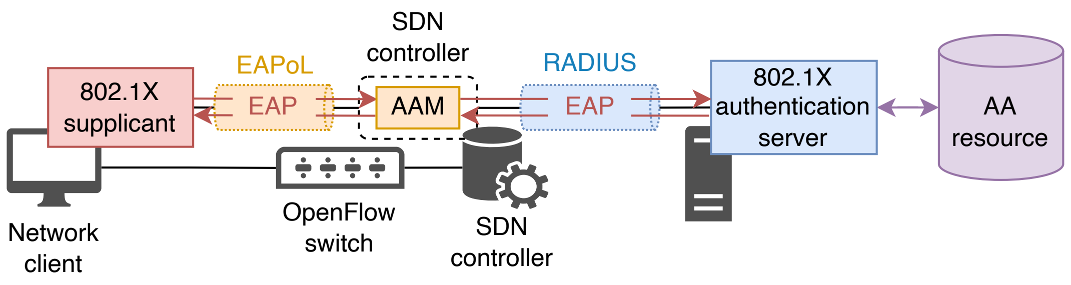
2.1.3 802.1X for OpenFlow SDN

One of the research questions of the bwNET100G+ project targeted transition strategies from legacy network switches to OF switches. While basic functions such as VLAN tagging, L2 forwarding, or access-control lists (ACLs) were already implemented for OF-based SDN, lack of support for 802.1X NAC was compensated by insecure MAC address identification or cumbersome web portal authentication.

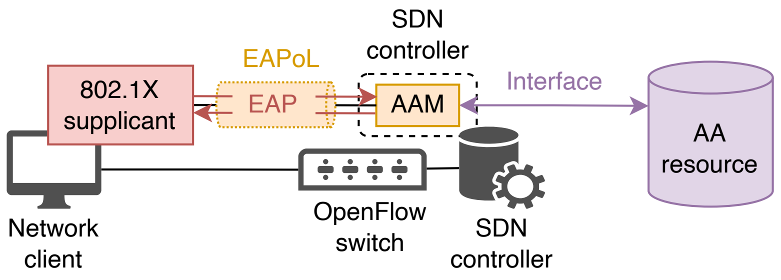
With the AAM [HSM17b], we presented the first integration of 802.1X for OF-based SDN. The AAM is an SDN application that is executed on the SDN controller. It comprises a standard-compliant 802.1X authenticator that is compatible to 802.1X supplicants on network devices and RADIUS 802.1X authentication servers. It installs flow table entries on the OF switch so that 802.1X traffic is forwarded to the AAM for processing. The AAM itself can output 802.1X traffic via the OF switch. Besides integrating an 802.1X authenticator, the AAM introduces two novelties that solve shortcomings of 802.1X.

The first novelty is *support for AA resources other than RADIUS*. Figure 2.3 depicts the two operation modes of the AAM. In authenticator mode (a), the AAM inherits the functionality of an 802.1X authenticator as described before. In authentication server mode (b), the AAM acts simultaneously as 802.1X authenticator and 802.1X authentication server so that AA resources, e.g., SQL databases, LDAP servers, or also CSV files, can be accessed without a RADIUS 802.1X authentication server. The AAM allows both modes to be used with various AA resources in parallel. The mode and AA resource to be used is either defined per port or dynamically chosen based on an anonymous identity indicator provided by the user. This allows sophisticated deployments, e.g., where NAC for students in a temporary lab course is realized using AA with simple CSV files while regular AA is still conducted via centralized RADIUS 802.1X authentication servers.

The second novelty is a *session database* that resolves limitations caused by the stateless property of RADIUS 802.1X authentication servers. Due to missing user session tracking, multiple users can complete AA using the same credentials. In the same way, changes of authorization on the 802.1X authentication server cannot be sent to the 802.1X authenticator to immediately block network access; they will be applied when the AA server is queried for the next time. Therefore, the AAM introduces a network-wide database of authenticated and authorized user entities with their sessions. The sessions are terminated by port-down notifications received from OF switches or via an



(a) Authenticator mode



(b) Authentication server mode

Figure 2.3: Two operation modes of the authentication and authorization module (AAM) (similar to [HSM17b]).

API that might be used by network control systems to remove sessions and block users when uncommon behavior was detected, e.g., by an intrusion detection system (IDS). In addition, the session database can trigger re-authentications and enforce session limits.

A prototype was implemented for the Open vSwitch, a software-based OF switch. It is managed by the Ryu SDN controller framework that runs the AAM implemented as SDN application.

Significance of the Results

The first prototype of the AAM was the result of my master thesis [Hau16]. After its finalization, I continued with concept enhancement and development of an improved prototype. The results were published in a conference paper [HSM17b] (Appendix 1.5) at the *IEEE International Conference on Communications (ICC)* in 2017. The conference talk was given by me.

The results were also presented on *IETF 96* in 2016, in a workshop of *ITG FG 5.2.4* in 2016, and in the *Matariki research workshop "Cyber Security"* in 2019. This work was the basis for xRAC (Section 2.1.4) and the integration of 802.1X in P4-based SDN (Section 2.2.4). In addition, the AAM was the use-case that initiated our considerations on local SDN controllers that we first presented in LoCoSDN (Section 2.4.1) and successfully applied in P4-MACsec (Section 2.2.2) and P4sec (Section 2.2.4).

2.1.4 xRAC: NAC and Execution Control for Containers with 802.1X

NAC with 802.1X controls the access of user hosts to a particular network, i.e., with successful AA, the user host with all its applications has access to the network. However, due to the growing requirements for network security, more fine-grained NAC would be desired. While firewalls were powerful packet filters in the past, they are hardly applicable nowadays due to traffic encryption and highly dynamic target IP addresses. Current workarounds are traffic identification using machine learning technologies and application fingerprinting via local agent tools on the network hosts. However, both are severely limited in their application and not a solid solution for security control purposes.

xRAC [HSM20] proposes a solution following an entirely different approach. It encapsulates applications within virtualization containers and controls their network access and execution with the help of 802.1X.

Figure 2.4 depicts the concept of restricted application containers (RACs) that form the basis of xRAC. RACs are virtualization containers that include an application, its dependencies, and optional configuration data. xRAC relies on operating system (OS) virtualization with Docker where the OS kernel is shared by the OS and multiple containers. Compared to system virtualization with virtual machines, container images are rather small and can be started without notable delays while resource overhead (e.g., CPU, memory) is minimal. This makes container virtualization one of the most widely used methods for developing and deploying software nowadays. RACs run atop a container runtime; their execution is managed by a container management daemon. RACs are executed on a managed host, i.e., the xRAC environment is protected against modifications by users. Also, RACs receive a unique and routable IPv6 address so that its traffic can be controlled by network components. RACs benefit from the easy software distribution in container virtualization, i.e., images can be published on a RAC image registry and synchronized to the managed host.

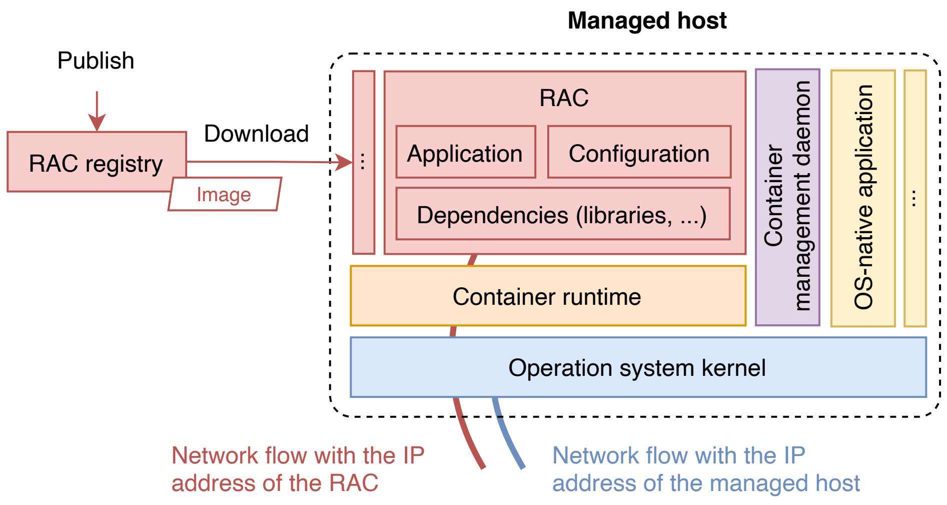

Managed host
Publish
RAC registry
Image
Download
RAC
Application
Configuration
Dependencies (libraries, ...)
Container management daemon
OS-native application
Container runtime
Operation system kernel
Network flow with the IP address of the RAC
Network flow with the IP address of the managed host

Figure 2.4: Concept of restricted application containers (RACs) that run in parallel to OS-native applications on a managed host (similar to [HSM20]).

xRAC integrates 802.1X for NAC and execution control for RACs. Authentication verifies the user credentials and integrity of the RAC image. Authorization checks if the user is permitted to execute the RAC and responds with an execution permission and additional data that describes the network resources that should be accessible for the RAC.

Figure 2.5 depicts how xRAC carries out NAC and execution control for RACs. The process is initiated by a user that issues the start of a particular RAC (1). The container management daemon requests start permission from the 802.1X container supplicant (2). Then, the 802.1X container supplicant initiates AA and performs authentication with the 802.1X authentication server via the 802.1X container authenticator (3). Authentication data comprises user credentials and the hash sum of the RAC image. The 802.1X container authenticator is an extension of the AAM (Section 2.1.3) that relays authentication data between the 802.1X container supplicant and 802.1X authentication server. As EAPoL does not allow AA of multiple RACs on a host, xRAC uses UDP as transport protocol for EAP (EAPoUDP). The 802.1X authentication server is extended by a data model and policy checks for RACs. In case of successful authentication and correct permissions, the 802.1X authentication server responds with authorization data (4). The 802.1X container authenticator applies the received information on network permissions by managing a network element that offers an API; here, an OF switch (4a). The execution permission is also forwarded by the 802.1X container authenticator to the 802.1X container supplicant that permits the start of the RAC to the container management daemon (4b). Now, the RAC but not the managed host can access the protected server.

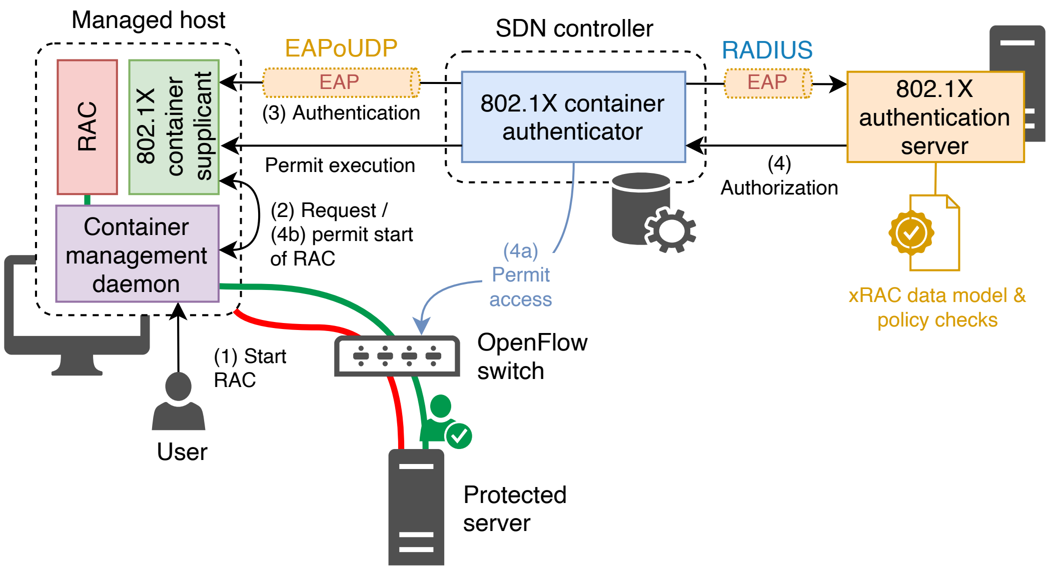

Managed host
RAC
802.1X container supplicant
Container management daemon
EAPoUDP
EAP
(3) Authentication
Permit execution
(2) Request / (4b) permit start of RAC
SDN controller
802.1X container authenticator
RADIUS
EAP
802.1X authentication server
(4) Authorization
(4a) Permit access
xRAC data model & policy checks
OpenFlow switch
(1) Start RAC
User
Protected server

Figure 2.5: Execution and NAC with 802.1X in xRAC. (similar to [HSM20]).

xRAC is highly beneficial if network access should be granted only to particular applications (RACs) but not to the managed host or other RACs. This might be the case in high-security areas without Internet access where users still require a web browser for research activities. Deploying a browser with xRAC allows that only the browser but not the managed host or other RACs have Internet access. The same applies for confidential data access where only a specific application within a RAC but not the managed host itself or other RACs should be able to access a particular server or internal network.

We implemented a prototype of xRAC without modifying the used components. Current Docker environments support dedicated and routable IPv6 addresses for containers that make them reachable for outside hosts. Also, the Docker container management daemon comprises the AuthZ framework that offers a REST interface used by the 802.1X container supplicant for AA of RACs. The necessary extensions for the data model and policy checks on the FreeRADIUS 802.1X authentication server were possible due to vendor-specific attributes (VSAs) and the unlang processing language. The 802.1X container authenticator is an extended version of the AAM (Section 2.1.3) running as an SDN application on the Ryu SDN controller. It integrates EAPoUDP and other functional extensions required for xRAC. This highly simplifies xRAC deployment in environments with existing FreeRADIUS servers or Docker hosts.

Significance of the Results

xRAC was published as conference paper [HSM20] (Appendix 1.3) and its prototype as demo paper [HM20] (Appendix 1.4) at the *IEEE/IFIP Network Operations and Management Symposium (NOMS)* in 2020. I gave the conference talks and presented the prototype demonstration. The source code of the xRAC prototype is available on GitHub [xra] with a license that prohibits any commercial usage. xRAC was also presented on the *ITG Workshop on IT Security (ITSec)* in 2020.

With the help of the technology transfer office of the University of Tuebingen, a patent application process was initiated in 2018 and is still ongoing. During the patent application, the concept was further developed and presented to several interested companies leading to helpful discussions.

Follow-up research ideas include an analysis on how xRAC can be integrated on Windows-based managed hosts. In addition, an extension by end-to-end protection of the communication between RACs and the target resource would be conceivable. After successful AA, the 802.1X container authenticator could set up and operate MACsec or IPsec protection. These automated operating mechanisms for MACsec and IPsec were developed in P4-MACsec (Section 2.2.2) and P4-IPsec (Section 2.2.3).

2.2 Securing Enterprise and Campus Networks with P4

This section covers research results on the integrations of 802.1X, MACsec, and IPsec for P4-based SDN. First, an introduction to data plane programming with P4 (Section 2.2.1) is given. Afterwards, P4-MACsec (Section 2.2.2), P4-IPsec (Section 2.2.3), and P4sec (Section 2.2.4) are presented.

2.2.1 Data Plane Programming with P4

SDN with only control plane programming, e.g., with OF (Section 2.1.2), limits the programmability to the control plane that steers fixed-function data plane devices via a predefined API. With data plane programming, programmers can also define the packet processing algorithms of data plane devices and introduce custom APIs (Chapter 1).

Out of different programming models, the protocol-independent switching architecture (PISA) emerged as the most widespread standard. P4 (Programming Protocol-Independent Packet Processors) is currently the most widely-used programming language for PISA that allows programmers to define data plane algorithms on an abstract layer. Those P4 programs are then executed on P4-capable hardware or software platforms, called P4 targets. P4 was first presented as research result [BDG+14]. Today, it is standardized within the P4 Language Consortium [p4-c] that is part of the ONF.

Figure 2.6 depicts how packets are processed by the programmable processing pipeline of P4. Its three parts are the parser, control blocks, and the deparser. The programmable *parser* allows the definition of packet header structures and parsing schemes. To provide support for complex nested packet header structures, the parser is based on a finite state machine. After parsing, the packet headers and metadata structures for user-defined metadata (e.g., intermediate results) and intrinsic metadata (e.g., ingress port and timestamps) travel through the control blocks. The control blocks comprise match-action-tables (MATs) and actions. Similar to OF, the entries in MATs decide about the action to be applied on the packet headers. Its entries are maintained via a data plane API by a control plane. In contrast to OF, all matching fields of MATs and all actions can be defined by the programmer. Additional platform-specific functionality that is not part of the P4 programming language but offered by the P4 target, e.g., stateful registers, counters, or packet encryption functions, can be used within the P4 program as P4 externs. After packet modification in the control blocks, the packet is serialized in the *deparser* so that it can be sent out.

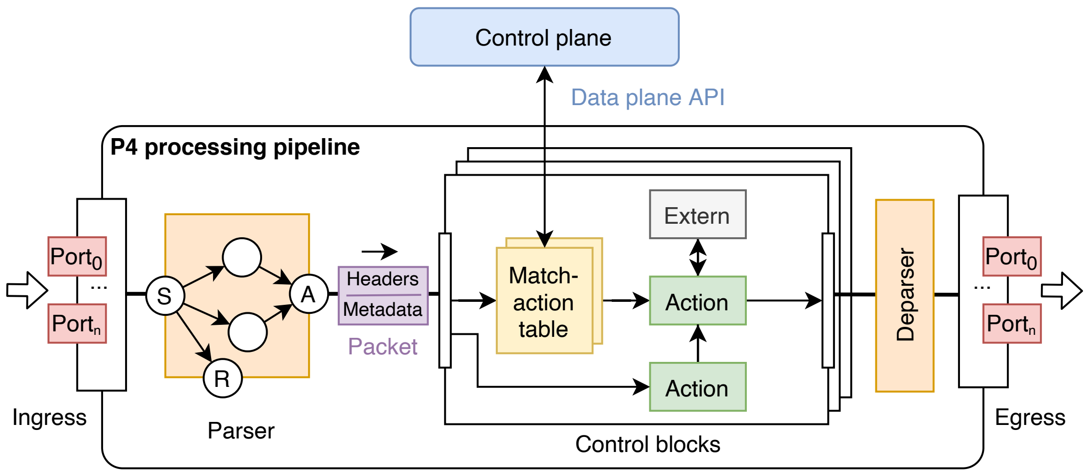


Figure 2.6: Simplified P4 processing pipeline (similar to [HHSM20]).

Figure 2.7 depicts how P4 programs are deployed to software- or hardware-based P4 targets. While software-based P4 targets are programs running on commodity computers, hardware-based P4 targets leverage network processing units (NPUs), field programmable gate arrays (FPGAs), or application-specific integrated circuits (ASICs) with P4 programmability. Although P4 aims at target-independence, differences caused by the variety of P4 targets exist. To still decouple P4 programs from the P4 targets, P4 architectures serve as an intermediate layer. P4 programs developed for a particular P4 architecture can be run on all P4 targets implementing this architecture. P4 programs are compiled by a P4 compiler; its results are data plane code to be executed by the P4 target and a data plane API definition that is used by the control plane for runtime management of the P4 target. The most commonly used data plane API is the P4Runtime API that is also standardized within the P4 Language Consortium.

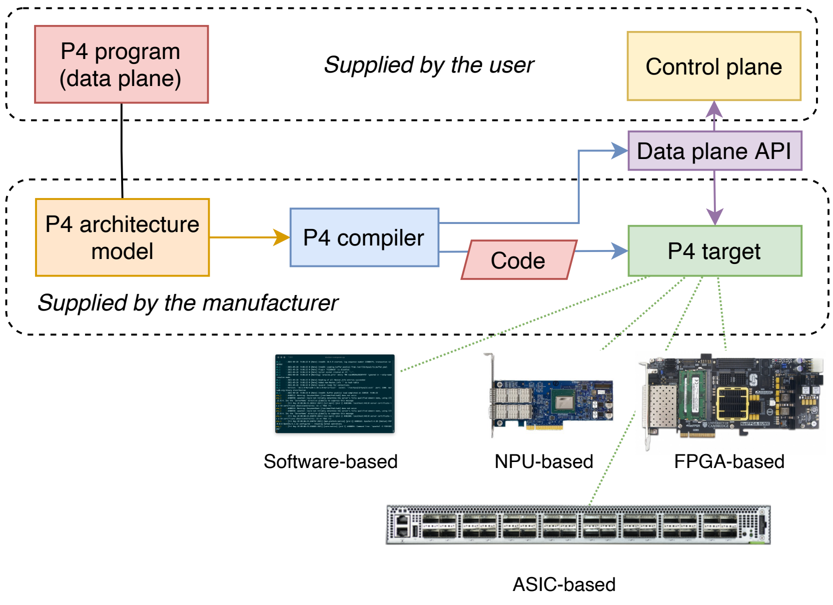


Figure 2.7: P4 program deployment (similar to [HHM+21b]).

Details on data plane programming with P4 can be found in our extensive survey [HHM+21b] (Appendix 2.2).

2.2.2 P4-MACsec

This section introduces MACsec foundations, presents the research results of P4-MACsec, and discusses the significance of P4-MACsec for my research work.

L2 Protection with MACsec

Media Access Control Security (MACsec) (IEEE 802.1AE) is a widespread standard for protecting Ethernet (IEEE 802) traffic. MACsec adds integrity, authenticity, confidentiality, and replay protection to Ethernet traffic by applying symmetric encryption and cryptographic hashing. It can be used to protect direct links between switches, links between switches and network hosts, and links between network hosts. MACsec supports different cipher suites whereas AES-GCM is the first standardized and mostly used variant. In contrast to end-to-end protection known from virtual private networks (VPNs) or application-layer security protocols such as Transport Layer Security (TLS), MACsec works in a point-to-point manner. Each networking device applies encryption/decryption so that functions such as ACLs can still be applied on the clear text packet. MACsec is part of the Linux kernel since v4.6; enterprise-grade hardware switches typically feature a MACsec hardware implementation with encryption/decryption in line rate.

MACsec sets up two unidirectional secure channels (SCs) per Ethernet link. Each SC holds a set of regularly changing security associations (SAs) with keying material. Although SC setup and SA management can be automatized, MACsec deployment still requires initial setup that cannot be automated due to missing secure topology discovery protocols.

Integration in P4-based SDN

P4-MACsec [HSHM20] introduces MACsec for P4-based SDN with automated deployment so that links between P4 targets are protected without any manual configuration. P4-MACsec comprises three parts: L2 switching with MAC address learning for L2 connectivity, a secured variant of the Link Layer Discovery Protocol (LLDP) for link discovery between P4 targets, and MACsec with automated deployment and operation. P4-MACsec features a two-tier control plane where each P4 target is managed by a local controller that connects to a central controller. Thereby, functions can be either part of the local controller, part of the central controller, or split and integrated in both tiers. This distribution of functionality reduces load on the central controller and ensures low latency.

Figure 2.8 depicts the architecture and function principle of P4-MACsec. L2 switching with MAC address learning provides the required base connectivity. The P4 processing pipeline comprises a MAC table that maps MAC addresses to ports of the switch. In case of missing entries, the MAC learning function on the local controller performs MAC address learning via packet flooding. Secure link discovery detects and monitors the links between the P4 targets. To resolve several security issues, LLDP frames are encrypted by AES-GCM with a common encryption key among all P4 targets. In addition, a sequence number against packet replay attacks is added. This function comprises a P4 data plane implementation and two control plane functions: the *link discovery function* running on the local controller and the *link discovery controller function* running on the central controller. The *link discovery function* performs link discovery by sending out encrypted LLDP packets that are received by other P4 targets. Thereby, it learns about links and notifies the *link discovery controller function* which updates the global link map. Link discovery is performed periodically and in case of port-up/-down notifications received from the P4 target. MACsec operation comprises a P4 data plane implementation and two control plane functions: the *MACsec function* running on the local controllers and the *MACsec controller function* running on the central controller. The data plane implementation relies on two P4 externs that need to be provided by the P4 target: *MACsec protect* to apply AES-GCM encryption and authentication and *MACsec validate* to apply AES-GCM decryption and authentication. MACsec operation then is triggered by the global link map. Whenever new links are discovered, the *MACsec controller function* is triggered to create new SCs with SA data and send them to the *MACsec function* of both P4 targets. The *MACsec function* then installs the SCs on the P4 target. Timeout conditions, e.g., packet counters for each SC on the data plane, trigger SA renewal.

We successfully built a software-based prototype with the Mininet network simulator and the Behavioral Model version 2 (bmv2) P4 target. bmv2 is a popular software-based prototyping platform. We extended it by the *MACsec protect* and *MACsec validate* functions that apply AES-GCM. Those functions are used within the P4 program as P4 externs. The local controllers and the central controller are implemented as Python programs. P4Runtime serves as data plane API between the P4 targets and local controllers; gRPC is used for the API between the local controllers and the central controller. We functionally validated the prototype and performed performance evaluations that show that the overhead introduced by MACsec, both on the data plane and control plane, is minimal compared to unprotected Ethernet operation.

We also investigated on a hardware prototype for the NetFPGA SUME platform, a FPGA-based P4 hardware target with four ports. The NetFPGA SUME platform is

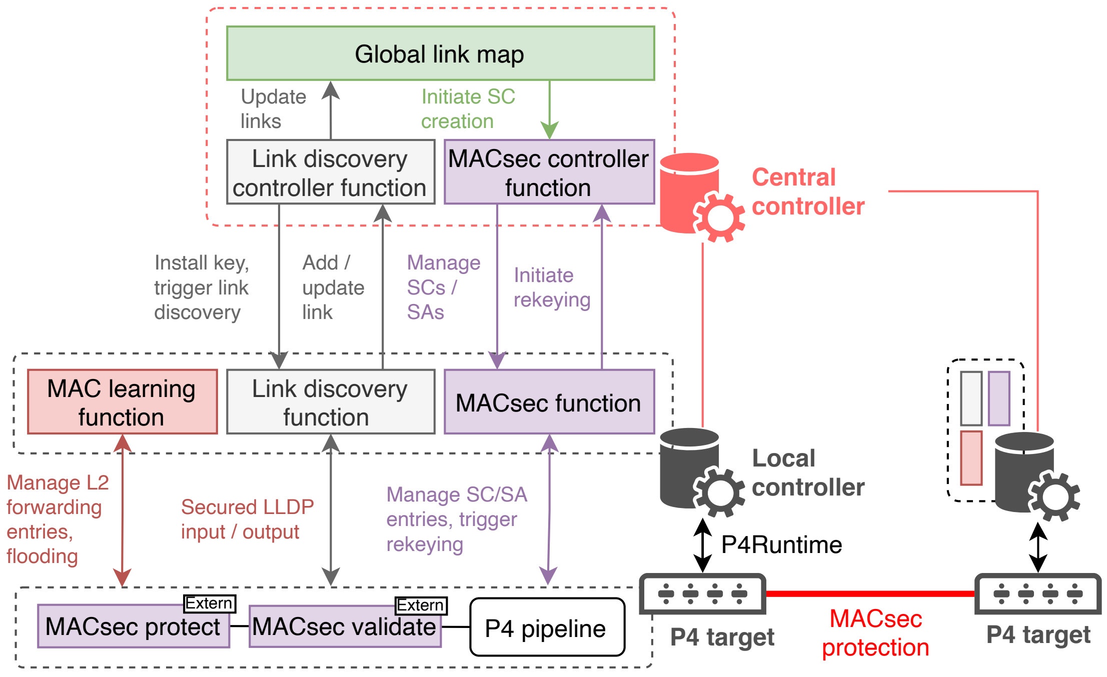


Figure 2.8: Automated operation of MACsec on links between P4 targets in P4-MACsec (similar to [HSHM20]).

natively programmed in SDNet, a vendor-specific predecessor of P4 from Xilinx, the manufacturer of the integrated FPGA. P4 programmability is facilitated by a trans-compiler that translates P4 programs into SDNet programs. We implemented the required P4 externs using a publicly available AES-GCM module on the FPGA so that it can be used within the P4 processing pipeline. However, due to platform limitations, we were only able to implement a very constrained prototype. As P4 only supports packet header processing, packet payloads have to be parsed as additional header field. The NetFPGA SUME board does not provide support for variable-length header fields, i.e., the prototype can only process fixed-size packets. In addition, the P4-NetFPGA trans-compiler lacks a packet streaming function for data exchange between the P4 processing pipeline and the P4 externs. Thereby, the final prototype was constrained to fixed-size packets with a total length not exceeding 128 bytes. This clearly disqualifies the prototype from real-world applicability.

Significance of the Results

P4-MACsec was published as journal paper [HSHM20] (Appendix 1.2) in *IEEE Access*. The source code of its prototype is available under the Apache v2 license on

GitHub [p4-b]. The integration of *MACsec protect* and *MACsec validate* functions as P4 externs was mentioned as exemplary implementation of crypto functions for the popular bmv2 target. I presented P4-MACsec at the *KuVS Fachgespräch "Network Softwarization"* in 2020.

P4-MACsec was the starting point for the development of hardware P4 prototypes within our research group. The development project on the NetFPGA SUME prototype led to extensive exchange with research colleagues from the field of embedded systems from our University. We also initiated contact with research colleagues and P4 hardware target manufacturers for further exchanges on our development experiences. The experience gained with the NetFPGA SUME platform also was the starting point for extending our scope to other P4 hardware targets. P4-MACsec was the basis for the follow-up works P4-IPsec (Section 2.2.3) and P4sec (Section 2.2.4).

2.2.3 P4-IPsec

This section introduces IPsec foundations, presents the research results of P4-IPsec, and discusses the significance of P4-IPsec for my research work.

L3 Protection with IPsec

Internet Protocol Security (IPsec) is a widespread IETF protocol suite for protecting the Internet protocol (IP). It can be used in host-to-host, host-to-site, and site-to-site scenarios. IPsec features two protocols. The Authentication Header (AH) adds authentication and integrity checks to IP packets, the Encrypted Security Payload (ESP) adds confidentiality through encryption. Two operation modes are considered by IPsec. In transport mode, IPsec protection is applied to a given IP frame. In tunnel mode, the original IP packet is encapsulated within a new IP packet for exchange between two IPsec peers. Despite the numerous operation variants, the ESP protocol in tunnel mode and site-to-site or host-to-site scenarios are the most commonly used operation patterns. Similar to MACsec, one bidirectional connection is secured by two IPsec connections, each with multiple SAs that comprise keying material. In contrast to MACsec, IPsec protection can be applied to selected flows instead of the entire traffic. Therefore, the traffic to be secured is defined in security policies (SPs) that are stored in a security policy database (SPD). Each SP maps matching patterns (e.g., destination IP address, protocol) to one of the three possible actions: apply IPsec protection, forward as is, or drop.

The SAs of each IPsec connection are part of a security association database (SAD). As in MACsec, SPs are typically defined manually while SAs are negotiated by the Internet Key Exchange (IKE), a distributed protocol for peer authentication and SA exchange. IPsec is still among the most widespread VPN technologies and thereby part of all modern operating systems and network devices such as firewalls or routers.

Integration in P4-based SDN

P4-IPsec [HHSM20] introduces IPsec for P4-based SDN. It provides support for host-to-site tunnels between remote clients and P4 targets and site-to-site tunnels between two P4 targets. P4-IPsec implements ESP in tunnel mode with support for different cipher suites. As in P4-MACsec, the cipher suites are implemented as P4 target extensions and used within P4 programs via P4 externs. For our prototype, we integrate the *NULL* and *AES-CTR* cipher suites. P4-IPsec integrates the SPD and SAD that are managed by a central controller via a data plane API without the need for distributed protocols such as IKE. IPsec functionality is part of the P4 program; thereby, any P4 target becomes a possible IPsec endpoint either for remote clients or other P4 targets. In contrast to centralized VPN concentrators, P4-IPsec allows IPsec tunnels to terminate close to the desired resource or on low utilization targets. For host-to-site VPN tunnel setup, we introduce the P4-IPsec agent tool to be executed on remote clients. It holds a management connection to the central controller to receive the required SP and SA data just as the P4 targets. Afterwards, the received configuration is applied to the IPsec functions of the remote client's OS.

Figure 2.9 depicts the architecture and function principle of P4-IPsec. The *L3 routing function* on the central controller provides the required L3 connectivity among P4 targets and remote clients. The network subnets of each P4 target are defined in a configuration on the central controller. The P4 targets implement L3 routing using longest prefix match (LPM) with the help of a MAT. Therefore, the *L3 routing function* on the central controller adds or updates LPM entries in the MATs of the P4 targets. IPsec connectivity for site-to-site and host-to-site tunnels is defined in a configuration file on the central controller. On the basis of this configuration, site-to-site tunnels between P4 targets are automatically set up and operated. Therefore, the central controller generates SP and SA data and writes it into the related MATs on the P4 target. The P4 targets receive soft and hard packet limits and keep counters for each IPsec tunnel to monitor both limits. In case the limit is reached, it notifies the central controller so that the *IPsec* *function* can perform rekeying by installing new SAs and instructing the P4 targets to change to that new SAs. Host-to-site tunnels are set up in a similar way. Here, the user requests IPsec tunnel setup via the P4-IPsec agent on the remote client. If permissions exist, the *IPsec function* again performs IPsec tunnel setup as for site-to-site tunnels. The P4 target is managed as in site-to-site operation; on the remote client, the P4-IPsec agent applies the received configuration on the host OS to set up the tunnel.

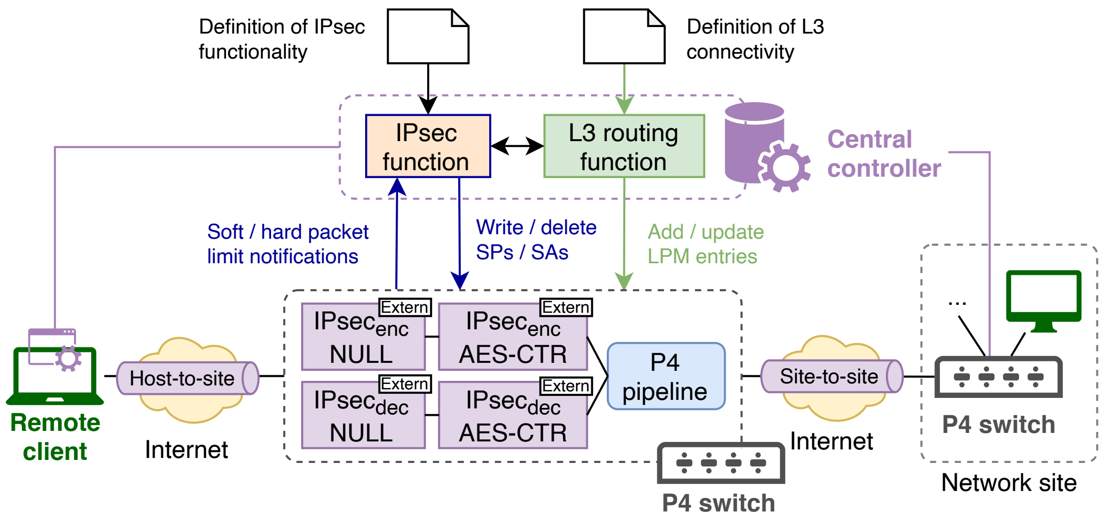


Figure 2.9: Automated operation of site-to-site and host-to-site IPsec tunnels in P4-IPsec (similar to [HHM21a]).

Similar to P4-MACsec, we successfully built a software-based prototype with the Mininet network simulator and the bmv2 P4 software target. Again, we extended the bmv2 by four extensions that integrate the encrypt/decrypt functions of the NULL and AES-CTR cipher suites. The P4-IPsec agent and central controller are implemented as Python programs. The central controller relies on the P4Runtime API for managing the P4 targets. The P4-IPsec agent establishes a connection with the central controller via gRPC.

Also similar to P4-MACsec (Section 2.2.2), we worked on a hardware prototype for the NetFPGA SUME platform but encountered the same difficulties due to the platform limitations as described before. The P4-IPsec prototype only provides support for the NULL cipher suite with fixed-length packets with a maximum length of 140 B

As one of the first research groups in Germany, we worked on a hardware prototype for the Edgecore 100BF-32X white box switch that is based on the Tofino ASIC. Today, Tofino-based white box switches are the defacto-standard P4 hardware target used in many research groups. Back then, acquisition was a complex process that involved

NDA signing and close exchange with the development team of Barefoot networks. The Edgecore 100BF-32X features 32 network ports and a main CPU module, i.e., a commodity server platform that runs a Linux-based operating system. As Tofino-based P4 hardware targets are optimized for high bandwidths, users cannot extend the platform functionalities to be used within the P4 program as P4 externs. Due to the popularity of Tofino-based P4 hardware targets, we investigated potential workarounds that forward traffic to a software-based packet processing function (PPF). The first workaround *outsources ESP processing to the main CPU module* of the white box switch. Therefore, all function calls to P4 externs are substituted by packet transfers to the server platform where the IPsec functions of the Linux kernel are used. Similar to the P4-IPsec agent, we introduced a crypto manager program that configures the kernel functions based on the IPsec configuration data received from the central controller. The second workaround *outsources ESP processing to a dedicated crypto host*. This variant works similarly except that ESP processing is handled on a more powerful external host. Again, a crypto manager tool serves as interface to the central controller. We performed performance experiments to compare and evaluate both workarounds for an IPsec tunnel with AES-GCM-256 between two hosts that are attached to the switch. While the second workaround yields higher round trip times (2 ms vs 1.5 ms), TCP goodput rates are by far higher (4 Gbit/s vs 1.4 Gbit/s). Although the throughputs in the first workaround are slower, performance might be still sufficient for many use cases.

Significance of the Results

P4-IPsec was published as journal paper [HHSM20] (Appendix 1.1) in *IEEE Access*. The source code of its prototype is available under the Apache v2 license on GitHub [p4-a]. P4-IPsec was the basis for our follow-up work P4sec.

2.2.4 P4sec

P4sec [HHM21a] integrates the previous works of 802.1X for OF-SDN (Section 2.1.3), P4-MACsec (Section 2.2.2), and P4-IPsec (Section 2.2.3) into one system. It aims at automated deployment of all three security mechanisms in distributed Campus and Enterprise networks. In contrast to the previous works, P4sec introduces a novel three-tier control plane, an implementation of 802.1X for P4-based SDN, and support for automated protection of host-to-switch links with the help of MACsec.

Figure 2.10 depicts the architecture and function principle of P4sec in an exemplary network. The network consists of two sites, each with multiple switches and local clients, and a remote client. As in P4-MACsec, all links between P4 targets are protected by automated deployment and operation of MACsec. L2 connectivity is provided by L2 forwarding using MAC address learning. As in P4-IPsec, P4 targets can be connected by site-to-site IPsec tunnels. Remote clients can establish a host-to-site IPsec tunnel to every P4 target. L3 connectivity is provided by L3 routing using LPM. As a novelty of P4sec, NAC with 802.1X can be enabled for ports of the P4 targets and links between local clients and switches can be protected by MACsec.

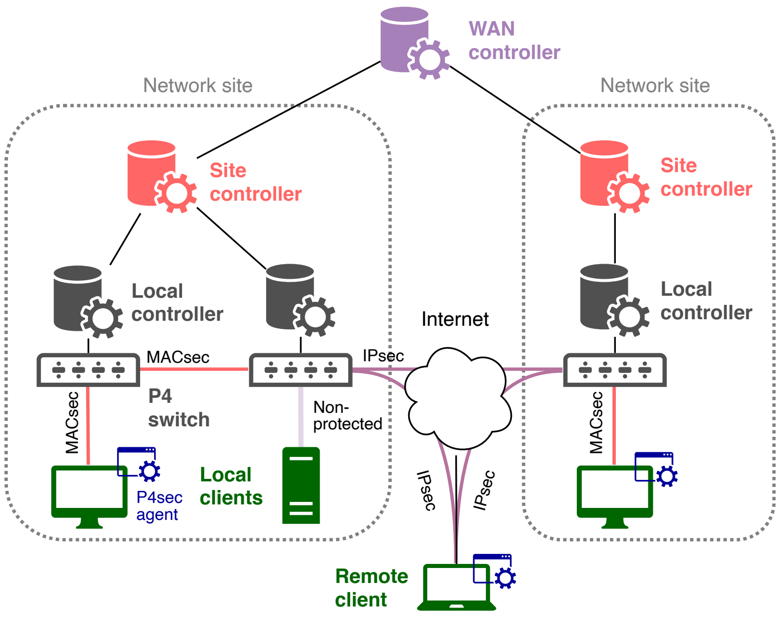


Figure 2.10: Architecture of P4sec in an exemplary network (taken from [HHM21a]).

The data plane functionality of all connectivity and security mechanisms is part of a P4 program whereas control plane functionality is achieved by a novel three-tier control plane that extends the two-tier control plane from P4-MACsec. Its architecture is shown in Figure 2.11. As for the two-tier control plane in P4-MACsec, functions can be part of one tier or split up and distributed to different tiers. For example, L2 forwarding and 802.1X are only part of the local controller, secured LLDP and MACsec are part of the local and site controller, and L3 routing and IPsec are part of all three tiers. P4Runtime is leveraged for communication between P4 targets and the local controllers; gRPC is used for the communication between the different control plane tiers. Instructions are sent top-down while notifications are delivered bottom-up. To simplify the implementation of the different control plane functions, the three-tier control plane introduces three control plane services: *execute*, *register*, and *status*.

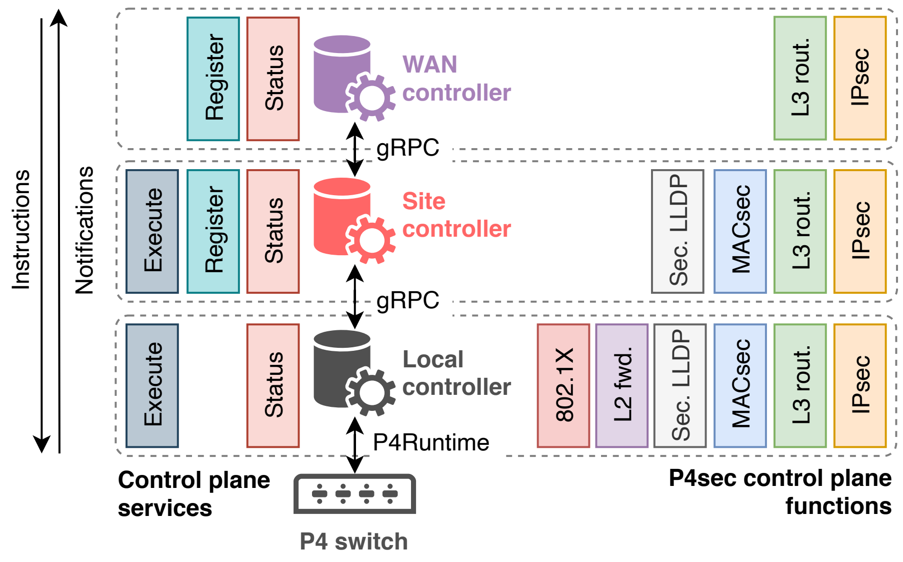


Figure 2.11: Three-tier control plane of P4sec with generic control plane services and P4sec control plane functions (taken from [HHM21a]).

The novel implementation of 802.1X adopts the principle from our past work (Section 2.1.3). However, the capabilities of P4 allowed us to shift more functionality to the data plane. For example, P4sec integrates an in-port and out-port authorizer allowing more sophisticated control of the port statuses and authenticated traffic. For automated deployment of MACsec protection on links between local hosts and switches, we extended the former P4-IPsec agent and call it P4sec agent. It includes the local client in secure link detection and MACsec operation. The P4sec agent receives the configuration data and applies it on the host, similar to IPsec tunnel setup on remote clients.

Analogue to P4-MACsec and P4-IPsec, we successfully built a software-based prototype with the help of the Mininet network simulator and the bmv2 P4 software prototyping target.

Significance of the Results

P4sec was submitted as article [HHM21a] (Appendix 2.1) to the *IEEE Networks* magazine. A revised version of the paper is currently in peer-review. The source code of its prototype is available under the Apache v2 license on GitHub [p4s].

This summarizing prototype of our works on 802.1X in OF SDN, P4-MACsec, and P4-IPsec is one of the core research outcomes of the bwNET100G+ research project.

The experienced restrictions of current P4 hardware targets was the starting point for several discussions with hardware manufacturers and research colleagues that encountered similar limitations.

2.3 Literature Study: Data Plane Programming with P4

This section covers the results of a literature study on data plane programming with P4. The P4-SOTA research project (Section 2.3.1) is described and the results of our literature study on data plane programming with P4 (Section 2.3.2) are presented.

2.3.1 P4-SOTA Research Project

Within our research activities around P4sec (Section 2.2.4) we have gained extensive experience in the field of data plane programming with P4. We implemented several prototypes using P4 software and hardware targets for our works of P4-MACsec, P4-IPsec, and P4sec. In parallel, research group colleagues developed P4 prototypes for novel mechanisms around IP multicast with Bit Index Explicit Replication (BIER).

In the scope of the "P4 State-of-the-Art" (P4-SOTA) research project funded by Siemens AG, we conducted a study combining the results of an extensive literature review with the experiences of our research group. The content scope comprised fundamentals, advances, and applied research. Deliverables were a project report, a presentation, and a Q&A session.

2.3.2 Literature Study on Data Plane Programming with P4

After completion of the P4-SOTA project in October 2019 we decided to continue research for an extensive survey paper publication. Fortunately, Vladimir Gurevich from Intel (formerly Barefoot Networks, the manufacturer of the Tofino ASIC) joined our collaboration.

Scope & Objective

While continuing research on the literature study, the current high level of activity in research around P4 has proven to be a real challenge. While the P4-SOTA report from October 2019 contained 248 references, the revised version of our survey from August 2021 now contains 519 references. 377 of them are scientific publications, the rest are white papers, source code repositories, websites, slides, and specifications.

Instead of an authorial selection of papers or restriction on particular research domains, we decided for an overall survey that considers all available resources. Figure 2.12 depicts the final structure of the survey. It consists of two main parts that are surrounded by general sections. In the first main part, an overview on the fundamentals and advancements of data plane programming with P4 is given. In the second part, the results of the survey on applied research conducted with P4 are presented.

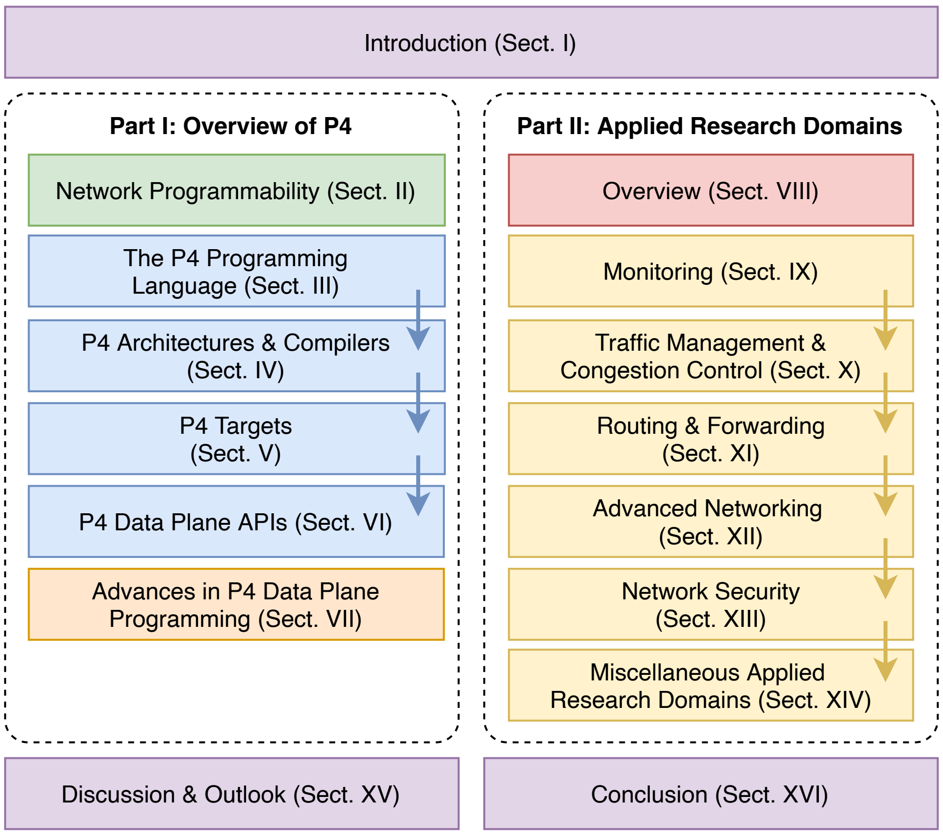

Introduction (Sect. I)
Part I: Overview of P4
Network Programmability (Sect. II)
The P4 Programming Language (Sect. III)
P4 Architectures & Compilers (Sect. IV)
P4 Targets (Sect. V)
P4 Data Plane APIs (Sect. VI)
Advances in P4 Data Plane Programming (Sect. VII)
Part II: Applied Research Domains
Overview (Sect. VIII)
Monitoring (Sect. IX)
Traffic Management & Congestion Control (Sect. X)
Routing & Forwarding (Sect. XI)
Advanced Networking (Sect. XII)
Network Security (Sect. XIII)
Miscellaneous Applied Research Domains (Sect. XIV)
Discussion & Outlook (Sect. XV)
Conclusion (Sect. XVI)

Figure 2.12: Structure of the survey on data plane programming with P4 (similar to [HHM+21b]).

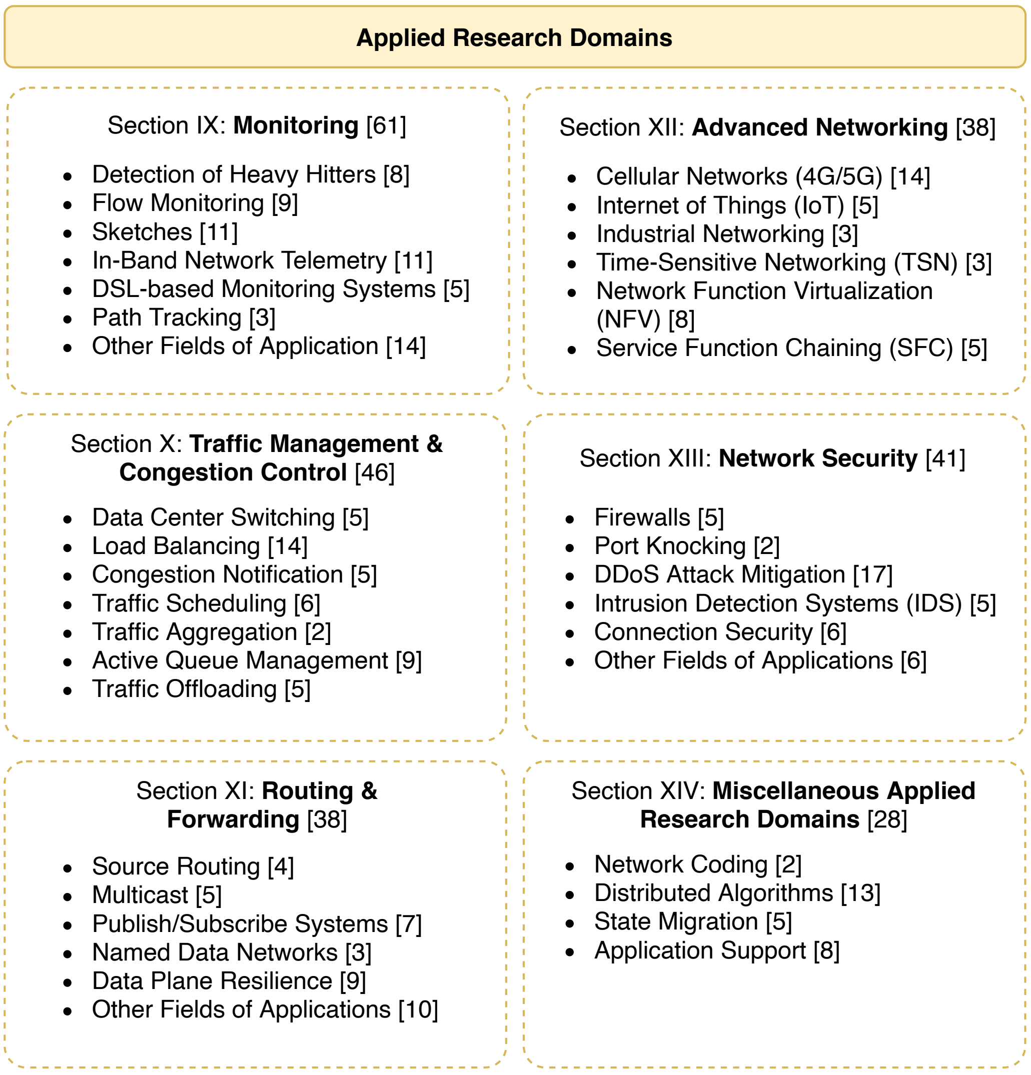
Part I: Overview of P4

The first main part of the survey covers fundamentals and advancements of data plane programming with P4. Section II ("*Network Programmability*") establishes a common definition of terms and describes the evolution from legacy networking to SDN with control plane programming and data plane programming. The two most common data plane programming models and P4 are introduced. Section III ("*The P4 Programming Language*") covers an introduction to the specification history of P4's two major versions ($P4_{14}$, $P4_{16}$), the development and deployment process of P4 programs to target platforms, and a tutorial on the programming language. Section IV ("*P4 Architectures & Compilers*") presents the concept of P4 architectures that serve as intermediate layer between P4 programs and P4 targets. The four most common P4 architectures, P4 compilers, and the concept of P4 externs are introduced. Section V ("*P4 Targets*") gives an overview on 10 P4 software targets and 7 P4 hardware target platforms that are based on FPGAs, ASICs, or NPUs. For each target, related work covering analyses, advancements, and extensions is presented. Section VI ("*P4 Data Plane APIs*") introduces APIs for runtime control of P4 data planes by a control plane. A characterization scheme, the three most common data plane APIs, and use case patterns are presented. Section VII ("*Advances in P4 Data Plane Programming*") presents research work on advances of P4 from the fields of development and deployment optimization, testing and debugging, research on P4 targets, and research on control plane operation.

Part II: Applied Research Domains

The second main part of the survey paper presents the results from a literature study on applied research in communication networks conducted with P4. It considers 241 research papers that significantly differ in length and quality. The range includes short abstracts but also long journal articles with detailed concept descriptions, prototype implementations, and performance evaluations.

To present a valuable analysis despite the significant amount of literature, we decided for a three-step procedure. First, research papers are categorized into a *two-level category scheme* that best models the topic structure of the literature. Figure 2.13 depicts the scheme comprising six main applied research domains, each with multiple categories. Within each category, we briefly summarize the core ideas of each paper where the length of this summary also reflects the level of detail given in the original paper. This categorization helps to identify the current applied research domains where P4 is used. Second, we introduce *property tables* for each applied research domain. It lists the included works by category and gives information about the publication year, P4 targets used for prototyping, and source code availability. Last, we present an *analysis* for each research domain where we discuss how the reviewed works benefit from P4's core features.



Applied Research Domains

Section IX: **Monitoring** [61]

- Detection of Heavy Hitters [8]
- Flow Monitoring [9]
- Sketches [11]
- In-Band Network Telemetry [11]
- DSL-based Monitoring Systems [5]
- Path Tracking [3]
- Other Fields of Application [14]

Section XII: **Advanced Networking** [38]

- Cellular Networks (4G/5G) [14]
- Internet of Things (IoT) [5]
- Industrial Networking [3]
- Time-Sensitive Networking (TSN) [3]
- Network Function Virtualization (NFV) [8]
- Service Function Chaining (SFC) [5]

Section X: **Traffic Management & Congestion Control** [46]

- Data Center Switching [5]
- Load Balancing [14]
- Congestion Notification [5]
- Traffic Scheduling [6]
- Traffic Aggregation [2]
- Active Queue Management [9]
- Traffic Offloading [5]

Section XIII: **Network Security** [41]

- Firewalls [5]
- Port Knocking [2]
- DDoS Attack Mitigation [17]
- Intrusion Detection Systems (IDS) [5]
- Connection Security [6]
- Other Fields of Applications [6]

Section XI: **Routing & Forwarding** [38]

- Source Routing [4]
- Multicast [5]
- Publish/Subscribe Systems [7]
- Named Data Networks [3]
- Data Plane Resilience [9]
- Other Fields of Applications [10]

Section XIV: **Miscellaneous Applied Research Domains** [28]

- Network Coding [2]
- Distributed Algorithms [13]
- State Migration [5]
- Application Support [8]

Figure 2.13: Categorization of use cases. The numbers in the square brackets indicate the number of analyzed publications in each area.

Results of the Literature Study

The survey is concluded with an extensive discussion and outlook. The results revealed that P4 emerged as the currently most widespread standard among different approaches for data plane programming. Its open availability and standardization as well as its large ecosystem with P4 targets, data plane APIs, advancements, and extensions have grown a large user group. This makes P4 attractive for research and application in academia and industry.

However, the survey also revealed that prototypes largely differ regarding their degree of applicability in real-world scenarios. The majority of prototypes of the reviewed papers were built for the bmv2 prototyping platform (Section 2.2.1) that is well-documented and compatible to any P4 program. Complexity is not constrained by hardware and bmv2 can be extended by custom functionality that can be used within P4 programs via P4 externs. Though, bmv2 is a software prototyping platform; especially its low throughput makes it not suitable for production usage.

Presented prototypes for P4 hardware targets are mostly built for white box switches with a Tofino ASIC or network interface cards (NICs) with a P4-programmable FPGA-/NPU such as the NetFPGA SUME (Section 2.2.1). Tofino-based P4 targets have throughput rates over 12.8 Tbit/s, 24-48 ports, and are built for WAN or data center applications. Optimized for high performance, program complexity is limited and programmers cannot introduce own functionality that can be used via P4 externs (Section 2.2.3). NICs with P4-programmable FPGAs or NPUs have only a few ports and are built for special-purpose server applications. On those platforms, programmers can run more complex P4 programs and integrate own functionality that can be used via P4 externs. While P4 is the native programming language for Tofino-based P4 targets, FPGA-/NPU-based P4 targets are natively programmed in a vendor-specific language (Section 2.2.2) where trans-compilers translate P4 programs into the vendor-specific format. However, the feature set might be limited so that access to all target features or the integration of custom functionality that can be used via P4 externs is only possible in the vendor-specific programming language. Whether P4 is beneficial compared to vendor-specific programming languages depends on the use case, programmer skills, and requirements towards target-independence.

However, prototypes that are not applicable to hardware targets often miss P4's original objectives of being a programmable packet header processor for highspeed targets. Those non-applicable use cases often include complex operations not only on the header but on the complete packet or rely on feature-rich externs. Still, the widespread and beneficial application of P4 shows a clear demand towards more flexible hardware platforms with more functionality (e.g., cryptographic functions) and the possibility for programmers to extend the platform with own externs.

Significance of the Results

The survey paper [HHM^{+}21b] (Appendix 2.2) is currently in peer review at the *Elsevier Journal of Network and Computer Applications (JNCA)*. Since its first version, we published preprints on arXiv [HHM^{+}21b] and received a lot of positive feedback from research colleagues. Besides being the core deliverables of the P4-SOTA project, the results from the literature study were part of numerous exchanges with colleagues and basis for two invited talks at conferences.

2.4 SDN Transition Strategies (Additional Content)

This section discusses research results on SDN transition strategies: LoCoSDN as an alternative to hybrid OF switches (Section 2.4.1) and the Master course "Network Softwarization" for education on SDN (Section 2.4.2). The presented works are additional content of this thesis.

2.4.1 LoCo SDN

Hybrid OF switches extend common network switches by OF programmability through an external control plane. Network functions from the local control plane, e.g., L2 switching, can be used in parallel with OF control, e.g., for centralized routing mechanisms. Although this simplifies SDN transitions, we experienced problematic behavior caused by function overlap in our research works on integrations of 802.1X (Section 2.1.3).

LoCoSDN [SHGM18] presents an alternative approach. Depicted in Figure 2.14, OF-only switches are coupled with a local controller that comprises common control plane features of traditional network switches such as L2 switching, L3 routing, and VLAN tagging. It offers a CLI and a web UI to administrators but can also be connected to external controllers. For this purpose, LoCo supports three operation modes. In *local controller mode*, only the local controller is utilized. In *remote controller mode*, only an external SDN controller is used. In *hybrid mode*, the OF switch is partitioned where some parts are controlled by the local controller and other parts are controlled by a remote controller.

LoCoSDN features a module system with three layers: the controller layer, the module layer, and the configuration layer. The *controller layer* comprises base functions of a

SDN controller and mechanisms for module orchestration. The *module layer* comprises the different modules, each implementing a particular function, e.g., L2 switching. The *configuration layer* comprises the configuration data of each module. This module system is also reflected in the storage model, i.e., parts of the controller layer can be upgraded without changing modules or their configuration. LoCoSDN adapts the configuration approach of most network devices that distinguishes between a startup config and a running config.

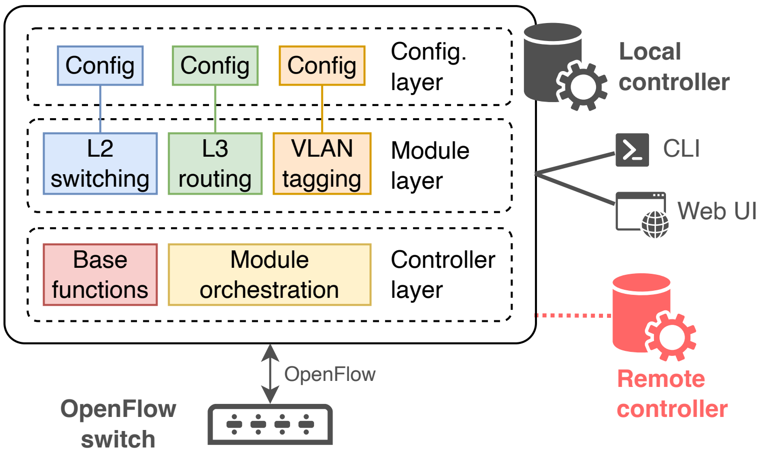


Figure 2.14: Concept of LoCoSDN (similar to [SHGM18]).

Shown in Figure 2.15, we built a hardware prototype consisting of a Raspberry Pi 3 and the Zodiac FX SDN hardware platform that appear as a single network switch. The local controller is implemented with the help of the Ryu SDN controller framework.

Significance of the Results

LoCoSDN was published as conference paper [SHGM18] (Appendix 3.1) at the *International Conference on Software Defined Systems (SDS)* in 2018. The conference talk was given by my colleague Mark Schmidt.

We applied the idea of local controllers with modular functionality and interfaces to other remote controllers in P4-MACsec (Section 2.2.2) and P4sec (Section 2.2.4). While local controllers and multi-tier control planes were a novelty in OF, they are applied in many research works conducted with P4 nowadays. The idea of using SDN control while exposing known CLIs or web UIs to the users is increasingly used by manufacturers that rely on programmable data planes to increase agility and improve time-to-market in product development.

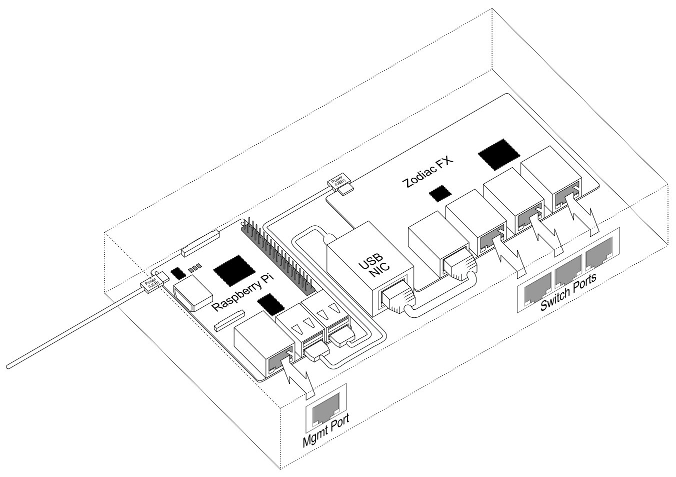


Figure 2.15: Assembled prototype hardware platform consisting of a Zodiac FX OF hardware switch and a Raspberry Pi (taken from [SHGM18]).

2.4.2 Master Course on Network Softwarization

Our research activities in the bwNET100G+ research project led to numerous topics for student research projects and master theses. Because of missing standard literature on SDN, the first students were required to familiarize themselves with the help of a set of research papers and programming tutorial recommendations provided by us. However, we observed highly different results in terms of concept understanding, depth of knowledge, and programming skills.

In order to create a homogeneous knowledge and experience base, the course "Network Softwarization" [HSM17a] was initiated in 2017 by my colleague Mark Schmidt and me under the supervision of Michael Menth. It was held in the summer terms of 2017, 2018, 2019, and 2021. This special course addresses master students with extensive knowledge and practical experiences in the field of communication networks. The course was designed for advanced master students as preparation for a thesis or student research project in that field. It has a scope of 3 ECTS with 90 minutes course time per week.

The course syllabus comprises lectures with slide presentations and practical assignments in the form of two programming projects. Figure 2.16 depicts how the contents of the lectures were aligned with each of the four passes of the course. While OF SDN was the main focus of the first pass, topics changed to P4 and network function virtualization (NFV) with its application, e.g., in cloud computing or in-band network telemetry (INT).

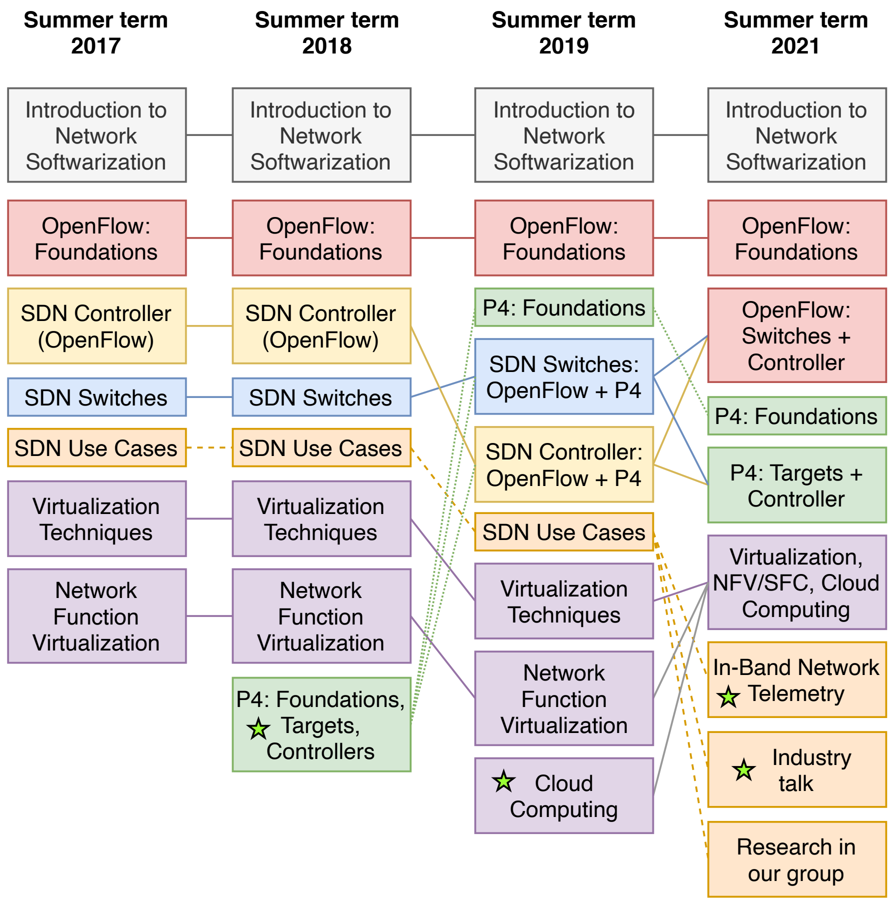

Summer term 2017
Summer term 2018
Summer term 2019
Summer term 2021
Introduction to Network Softwarization
OpenFlow: Foundations
SDN Controller (OpenFlow)
SDN Switches
SDN Use Cases
Virtualization Techniques
Network Function Virtualization
P4: Foundations, Targets, Controllers
P4: Foundations
SDN Switches: OpenFlow + P4
SDN Controller: OpenFlow + P4
Cloud Computing
OpenFlow: Switches + Controller
P4: Targets + Controller
Virtualization, NFV/SFC, Cloud Computing
In-Band Network Telemetry
Industry talk
Research in our group

Figure 2.16: Overview on content blocks of the course "Network Softwarization". Topics that were newly introduced in the respective semester are marked with an asterisk.

In the two programming projects, the students are required to apply the concepts and techniques learned in the lecture. While the programming projects of the first two passes were both on OF, now the first programming project covers OF while the second one covers P4. To foster team work, the students work in groups of two on the projects. Each programming project comprises a group test and programming assignments. We dedicate parts of the weekly timeslot for Q&A sessions, the group tests, and result presentations. The group test verifies that both team members have familiarized themselves with the programming assignments. Therefore, the students receive 15-20 related questions as basis for individual preparation; about 5 of them are asked in a short oral query. Successfully passing the oral query is a requirement for getting a grade for the programming project. An overall score of 60% is required to take part in the final exam where scores over 60% are translated into bonus points. To deal with difficult environment setup for the programming projects, we provided a virtual machine (VM) with all required software and configuration.

Significance of the Results

The concept and experiences of our Master course “Network Softwarization” was published as teaching report [HSM17a] (Appendix 3.2) at the *KuVS Fachgespräch “Network Softwarization – From Research to Application”* in 2017. I presented the results at the workshop.

In 2021, the course is held for the fourth time. While the first two passes were solely held by Mark Schmidt and me, more and more colleagues joined the teaching team. For its current pass, I only took part in the content restructurization with no involvement in teaching.

Over the semesters, we received good feedback from the participating students and recorded generally excellent test results. While we typically started with 20-30 students, many of them dropped out prior to the submission of the first programming project so that only 8-10 students took part in the final exam. However, the majority of those course graduates started a master thesis or student research project in our group. Compared to the situation before the first completion of the course, the homogeneous and in-depth knowledge and programming experiences largely decreased the individual supervision efforts for master theses and led to excellent thesis and student research project outcomes.

For me, the positive teaching experiences from that course were also the trigger for a higher education didactics training to acquire the “Baden-Württemberg Certificate for Higher Education Didactics” [Hoc]. Thereby, the course was part of many discussions, presentations, and also a collegiate hospitation by a fellow student. I have interacted with many teaching colleagues from different fields on experiences and open issues of the course.

2.5 Modelling & Simulation (Additional Content)

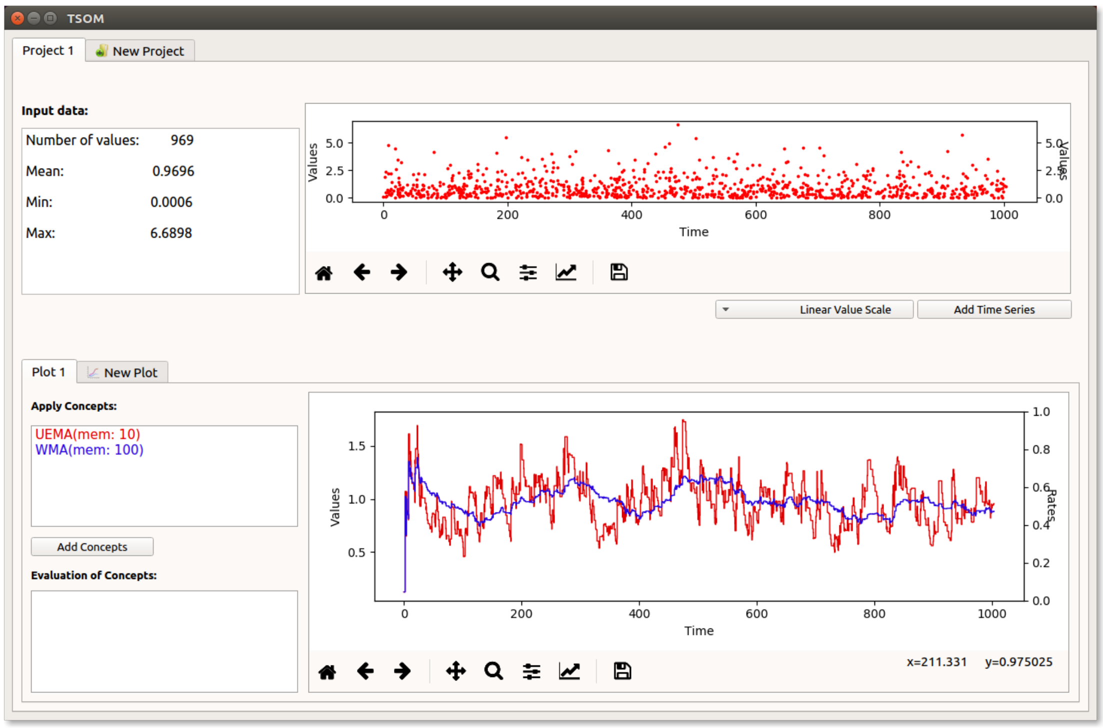
Prior to my PhD studies, I was student assistant for the course ”Modelling & Simulation I/II”. My main task was to introduce the R programming language for modelling, statistical analyses, and data visualization in the form of a tutorial and exercises. In my first PhD year, I was co-supervisor of the course and revised the discrete-event simulation tool that is the basis for several exercises.

These teaching activities were accompanied and followed by thematically subsequent research works. This section discusses research results on time-dependent statistics (Section 2.5.1) and discrete-time Markov chain (DTMC) modelling (Section 2.5.2). The presented works are additional content of this thesis.

2.5.1 Time-Dependent Statistics for Online Measurement

Time-dependent statistics for online measurement [MH17b] are crucial for adaptive control and behavior tracking of technical systems. In contrast to standard average calculation, time-dependent statistics give greater weights to recent samples where older samples have smaller or zero weights. In our research work, we defined a framework for moving averages, presented four moving averages (MAs) for evenly spaced time series and three MAs for unevenly spaced time series, and analyzed their characteristics. As a superior alternative to the current MAs with exponential weights, we propose the novel Unbiased Exponential Moving Average (UEMA) for evenly spaced time series and the Unbiased Time-Exponential Moving Average (UTEMA) for unevenly spaced time series. The first extension to MAs are moving histograms (MHs). Here, UEMA and UTEMA are applied on histograms for getting time-dependent quantiles. The second extensions of MAs are time-dependent rate measurements (TDRMs). We present and compare different approaches, and propose the UTEMA-based mechanism TDRM-UTEMA as a superior alternative.

TSOMpy [MH17a] is a Python tool that implements all 23 MA, MH, and TDRM measurement methods presented in our research paper. It also includes testing and plotting functions that can be used within a graphical user interface (GUI) to apply the measurement methods to evenly or unevenly spaced time series. These time series can be either generated via several distributons or imported from data files. One or multiple measurement methods can be applied, and their output can be evaluated and visualized in plots. Figure 2.17 depicts an exemplary screenshot of the TSOMpy tool where two MAs are applied to a generated sample data set.



Input data:

Number of values:	969
Mean:	0.9696
Min:	0.0006
Max:	6.6898

Apply Concepts:

UEMA(mem: 10)
WMA(mem: 100)

Evaluation of Concepts:

Figure 2.17: Screenshot from the TSOMpy tool. Two MAs are applied on a generated sample data set. The samples and MA outputs are visualized in plots.

Significance of the Results

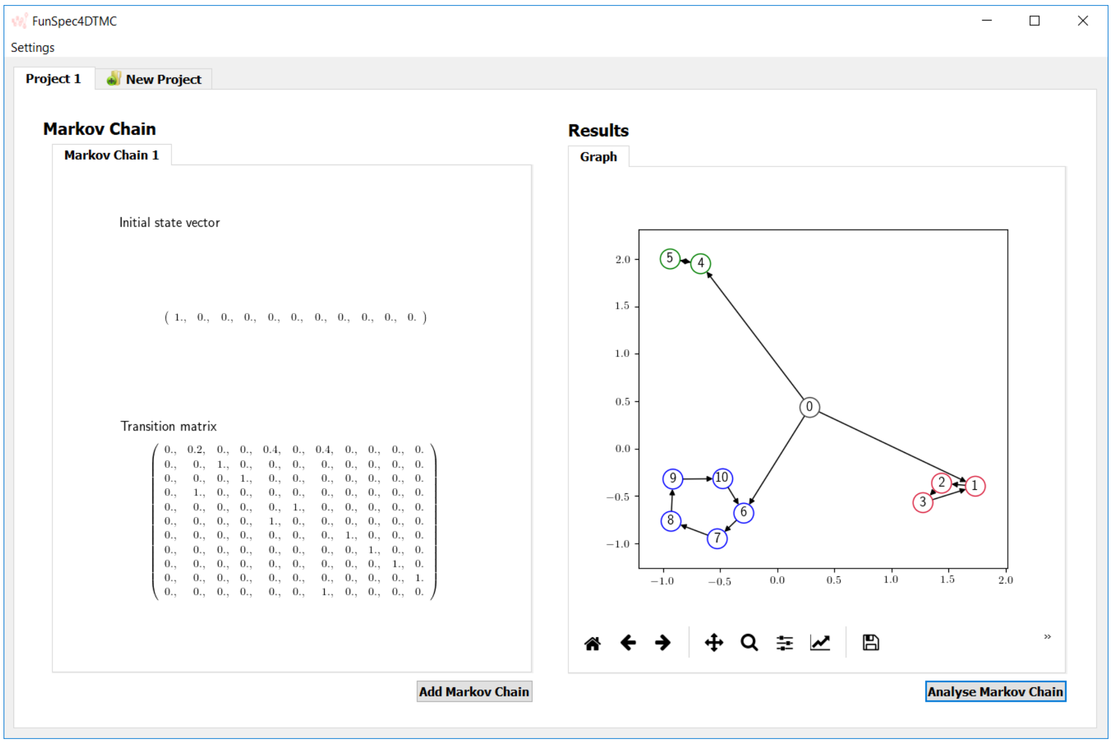
The results of our research on time-dependent statistics were published as conference paper [MH17b] (Appendix 3.3) and the TSOMpy tool as demo paper [MH17a] (Appendix 3.4) at the *ACM/SPEC International Conference on Performance Engineering (ICPE)* in 2017. I gave the conference talk and presented the demonstration at the conference. We released the source code of TSOMpy under the GPLv3 license on GitHub [tso].

The results were applied in several research works, mainly in the field of SDN. Examples are an offloading mechanism for firewalls based on OF [HSMM17] and a classification mechanism for network function offloading [DK20].

2.5.2 FunSpec4DTMC – Modelling of Discrete-Time Markov Chains

FunSpec4DTMC [HKM18] is a Python tool for modelling and analysis of discrete-time Markov chains (DTMCs) in an interactive, GUI-based process. Besides various known direct and indirect computation methods for DTMCs, it implements the novel functional specification and forward algorithm that was developed and published in a former work of my PhD supervisor [Men11].

FunSpec4DTMC features a four-step analysis process. In the first step, the DTMC model is defined within a GUI or imported from a data file. As an alternative for teaching, the parameters of an exemplary DTMC system can be specified. In the second step, the given DTMC model input is parsed and validated. In the third step, computation methods for DTMC metrics can be applied. In the final step, computation outputs can be visualized. Figure 2.18 depicts an exemplary GUI screen of FunSpec4DTMC where a DTMC is defined using an initial state vector and transition matrix. After calculation, its state transitions are visualized in a graph.



Initial state vector

$$\begin{pmatrix} 1., & 0., & 0., & 0., & 0., & 0., & 0., & 0., & 0., & 0., & 0. \end{pmatrix}$$

Transition matrix

$$\begin{pmatrix}
0., & 0.2, & 0., & 0., & 0.4, & 0., & 0.4, & 0., & 0., & 0., & 0. \\
0., & 0., & 1., & 0., & 0., & 0., & 0., & 0., & 0., & 0., & 0. \\
0., & 0., & 0., & 1., & 0., & 0., & 0., & 0., & 0., & 0., & 0. \\
0., & 1., & 0., & 0., & 0., & 0., & 0., & 0., & 0., & 0., & 0. \\
0., & 0., & 0., & 0., & 0., & 1., & 0., & 0., & 0., & 0., & 0. \\
0., & 0., & 0., & 0., & 1., & 0., & 0., & 0., & 0., & 0., & 0. \\
0., & 0., & 0., & 0., & 0., & 0., & 0., & 1., & 0., & 0., & 0. \\
0., & 0., & 0., & 0., & 0., & 0., & 0., & 0., & 1., & 0., & 0. \\
0., & 0., & 0., & 0., & 0., & 0., & 0., & 0., & 0., & 1., & 0. \\
0., & 0., & 0., & 0., & 0., & 0., & 0., & 0., & 0., & 0., & 1. \\
0., & 0., & 0., & 0., & 0., & 0., & 1., & 0., & 0., & 0., & 0.
\end{pmatrix}$$

Figure 2.18: Screenshot from the FunSpec4DTMC tool. The possible transitions of a DTMC specified by its initial state vector and transition matrix is shown.

Significance of the Results

FunSpec4DTMC was published as demo paper [HKM18] (Appendix 3.5) at the *International GI/ITG Conference on Measurement, Modelling and Evaluation of Computing Systems (MMB)* in 2018. The demonstration was presented by the student Dominik

Krauß that was supervised by me. We released the source code of FunSpec4DTMC under the GPLv3 license on GitHub [fun]. Today, the tool is mainly used for teaching purposes in our lecture “Modelling & Simulation I/II”.

Bibliography

[802] Wikimedia Commons: Diagram showing protocols involved in wired 802.1X authentication. https://commons.wikimedia.org/wiki/File:802.1X_wired_protocols.png. Last accessed: 2021-11-27.

[BDG$^+$14] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming Protocol-independent Packet Processors. *ACM SIGCOMM Computer Communications Review (CCR)*, 44, 2014.

[BM14] Wolfgang Braun and Michael Menth. Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices. *Future Internet (FI)*, 6(2):302–336, 2014.

[bwna] bwNET100G+ Research Project Description. https://www.belwue.de/ueberuns/archiv/projekte.html#bwnet100g. Last accessed: 2021-11-27.

[bwnb] Website of the bwNET100G+ Research Project. http://bwnet100g.de/. Last accessed: 2021-11-27.

[bwnc] Website of the bwNET2020+ Research Project. https://bwnet.belwue.de/. Last accessed: 2021-11-27.

[DK20] Raphael Durner and Wolfgang Kellerer. Network Function Offloading Through Classification of Elephant Flows. *IEEE Transactions on Network and Service Management (TNSM)*, 17(2):807–820, 2020.

[fun] GitHub: FunSpec4DTMC. https://github.com/uni-tue-kn/funspec4dtmc. Last accessed: 2021-11-27.

[Hau16] Frederik Hauser. Authentifizierung und Autorisierung in kabelgebundenen OpenFlow-basierten Campus-Netzwerken mit 802.1X als Netzwerkapplikation. Master's thesis, University of Tuebingen, Sand 13, 72076 Tübingen, 6 2016.

[HHM21a] Frederik Hauser, Marco Häberle, and Michael Menth. P4sec: Automated Deployment of 802.1X, IPsec, and MACsec Network Protection in P4-Based SDN. Technical report, University of Tuebingen, 2021.

[HHM$^+$21b] Frederik Hauser, Marco Häberle, Daniel Merling, Steffen Lindner, Vladimir Gurevich, Florian Zeiger, Reinhard Frank, and Michael Menth. A Survey on Data Plane Programming with P4: Fundamentals, Advances, and Applied Research, 2021. Preprint available on arXiv: `https://arxiv.org/abs/2101.10632`.

[HHSM20] Frederik Hauser, Marco Häberle, Marc Schmidt, and Michael Menth. P4-IPsec: Site-to-Site and Host-to-Site VPN With IPsec in P4-Based SDN. *IEEE Access*, 8:139567–139586, 2020.

[HKM18] Frederik Hauser, Dominik Krauß, and Michael Menth. FunSpec4DTMC – A Tool for Modelling Discrete-Time Markov Chains Using Functional Specification. In *International GI/ITG Conference on Measurement, Modelling and Evaluation of Computing Systems (MMB)*, pages 332–337. Springer International Publishing, 2018.

[HM20] Frederik Hauser and Menth Menth. Demo: Execution and Access Control for Restricted Application Containers on Managed Hosts (xRAC). In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 1–2, 2020.

[Hoc] Hochschuldidaktikzentrums Baden-Württemberg. Baden-Württemberg-Zertifikat für Hochschuldidaktik. https://www.hdz-bawue.de/das-angebot-des-hdz/baden-wuerttemberg-zertifikat/. Last accessed: 2021-11-27.

[HSHM20] Frederik Hauser, Mark Schmidt, Marco Häberle, and Michael Menth. P4-MACsec: Dynamic Topology Monitoring and Data Layer Protection With MACsec in P4-Based SDN. *IEEE Access*, 8:58845–58858, 2020.

[HSM17a] Frederik Hauser, Mark Schmidt, and Michael Menth. A Master Course on Network Softwarization: Lectures and Practical Assignments. In *1.*

KuVS Fachgespräch "Network Softwarization – From Research to Application", 2017.

[HSM17b] Frederik Hauser, Mark Schmidt, and Michael Menth. Establishing a session database for SDN using 802.1X and multiple authentication resources. In *IEEE International Conference on Communications (ICC)*, pages 1–7, 2017.

[HSM20] Frederik Hauser, Mark Schmidt, and Michael Menth. xRAC: Execution and Access Control for Restricted Application Containers on Managed Hosts. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 1–9, 2020.

[HSMM17] Florian Heimgaertner, Mark Schmidt, David Morgenstern, and Michael Menth. A software-defined firewall bypass for congestion offloading. In *International Conference on Network and Service Management (CNSM)*, pages 1–9, 2017.

[IEE01] IEEE. Standard for Port Based Network Access Control. *IEEE Std 802.1X-2001*, pages 1–140, 2001.

[IEE04] IEEE. Standard for Local and metropolitan area networks - Port-Based Network Access Control. *IEEE Std 802.1X-2004 (Revision of IEEE Std 802.1X-2001)*, pages 1–175, 2004.

[IEE20] IEEE. Standard for Local and Metropolitan Area Networks–Port-Based Network Access Control. *IEEE Std 802.1X-2020 (Revision of IEEE Std 802.1X-2010 Incorporating IEEE Std 802.1Xbx-2014 and IEEE Std 802.1Xck-2018)*, pages 1–289, 2020.

[MAB$^+$08] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communications Review (CCR)*, 38(2):69–74, March 2008.

[Men11] Michael Menth. Description and Analysis of Markov Chains Based on Recursive Stochastic Equations and Factor Distributions. *World Journal of Modelling and Simulation (WJMS)*, 7, 02 2011.

[MH17a] Michael Menth and Frederik Hauser. Demo: Time Series Online Measurement for Python (TSOMpy). In *ACM/SPEC on International Conference on Performance Engineering (ICPE)*, page 175–176, 2017.

[MH17b] Michael Menth and Frederik Hauser. On Moving Averages, Histograms and Time-DependentRates for Online Measurement. In *ACM/SPEC on International Conference on Performance Engineering (ICPE)*, page 103–114, 2017.

[of-] OpenFlow Switch Specification Version 1.5.1. `https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf`. Last accessed: 2021-11-27.

[onf] ONF: OpenFlow: Catalyst that Kickstarted the SDN Transformation. `https://opennetworking.org/news-and-events/blog/openflow-catalyst-that-kickstarted-the-sdn-transformation/`. Last accessed: 2021-11-27.

[p4-a] GitHub: P4-IPsec. `https://github.com/uni-tue-kn/p4-ipsec`. Last accessed: 2021-11-27.

[p4-b] GitHub: P4-MACsec. `https://github.com/uni-tue-kn/p4-macsec`. Last accessed: 2021-11-27.

[p4-c] Website of the P4 Language Consortium. `https://p4.org/`. Last accessed: 2021-11-27.

[p4s] GitHub: P4sec. `https://github.com/uni-tue-kn/P4sec`. Last accessed: 2021-11-27.

[SHGM18] Mark Schmidt, Frederik Hauser, Bastian Germann, and Michael Menth. LoCoSDN: A local controller for operation of of switches in non-SDN networks. In *International Conference on Software Defined Systems (SDS)*, pages 1–7, 2018.

[tso] GitHub: TSOMpy. `https://github.com/uni-tue-kn/TSOMpy`. Last accessed: 2021-11-27.

[xra] GitHub: xRAC. `https://github.com/uni-tue-kn/xrac`. Last accessed: 2021-11-27.

Personal Contribution

Accepted Manuscripts (Core Content)

1. **P4-IPsec: Site-to-Site and Host-to-Site VPN With IPsec in P4-Based SDN** [HHSM20]

Scope of the joint work	Research work on an integration of IPsec in P4-based SDN. Developed as follow-up work to P4-MACsec [HSHM20] in the context of the bwNET100G+ research project.
Names of collaborators and their shares	Marco Häberle: Development of an elementary prototype in his master thesis. Assistance in performance evaluation experiments. Review of technological and architectural aspects for the publication. Mark Schmidt: Co-supervision of the master thesis (focus on the prototype implementation). Michael Menth: Scientific supervision of the master thesis. Editorial assistance on the publication.
Importance of own contributions to the joint work	Co-supervision of the master thesis (focus on concept and architecture). Lead author of the paper, taking up most of the write-up. Complete rework of the paper and reimplementation of the prototype (introduction of site-to-site mode, integration of SPD and SAD) for a paper revision requested by the journal.

2. **P4-MACsec: Dynamic Topology Monitoring and Data Layer Protection With MACsec in P4-Based SDN** [HSHM20]

Scope of the joint work	Research work on an integration of MACsec in P4-based SDN. Developed in the context of the bwNET100G+ research project based on the findings of our previous works on OF-based SDN [HSM17b, SHGM18, HSM20].
Names of collaborators and their shares	Mark Schmidt: Joint identification and development of the research topic. Co-supervision of the master thesis of Joshua Hartmann (focus on the prototype implementation). Marco Häberle: Co-revision of the prototype for an open-source release in conjunction with the publication. Assistance in performance evaluation experiments. Review of technological and architectural aspects. Joshua Hartmann: Development of an elementary prototype in his master thesis. Michael Menth: Scientific supervision of the master thesis. Editorial assistance on the publication.
Importance of own contributions to the joint work	Co-supervision of the master thesis of Joshua Hartmann (focus on the concept and architecture). Lead author of the paper, taking up most of the write-up. Co-revision of the original architecture for the publication and open-source release.

3. **xRAC: Execution and Access Control for Restricted Application Containers on Managed Hosts** [HSM20]

Scope of the joint work	Research work in the context of the bwNET100G+ research project based on the findings of our previous work on integrating 802.1X in OF-based SDN [HSM17b].
Names of collaborators and their shares	Mark Schmidt: Joint identification and development of the research topic. Co-supervision of the master thesis. Assistance on several technological aspects. Julian Rilli: Development of an elementary prototype in his master thesis. Michael Menth: Scientific supervision of the master thesis. Editorial assistance on the publication.
Importance of own contributions to the joint work	Co-supervision of the master thesis. Lead author of the paper, taking up most of the write-up. Revision of major parts of the original concept and architecture also within the scope of a patent application process. Preparation and presentation of the conference talk.

4. **Demo: Execution and Access Control for Restricted Application Containers on Managed Hosts (xRAC)** [HM20]

Scope of the joint work	Demonstration of the xRAC [HSM20] prototype in the context of the bwNET100G+ research project.
Names of collaborators and their shares	Michael Menth Editorial assistance on the publication.
Importance of own contributions to the joint work	Revision of the prototype implementation for an open-source release. Lead author of the paper, taking up most of the write-up. Preparation and presentation of the conference demo.

5. **Establishing a session database for SDN using 802.1X and multiple authentication resources** [HSM17b]

Scope of the joint work	Extended concept and prototype that was developed in my master thesis [Hau16] in the context of the bwNET100G+ research project.
Names of collaborators and their shares	Mark Schmidt Identification and development of the research topic. Supervision of my master thesis. Michael Menth Scientific supervision of the master thesis. Editorial assistance on the publication.
Importance of own contributions to the joint work	Development of the prototype. Lead author of the paper, taking up most of the write-up. Preparation and presentation of the conference talk.

Submitted Manuscripts (Core Content)

6. **P4sec: Automated Deployment of 802.1X, IPsec, and MACsec Network Protection in P4-Based SDN** [HHM21a]

Scope of the joint work	Integration of our previous works on 802.1X [HSM17b], MACsec [HSHM20], and IPsec [HHSM20] in a unified prototype as final project task of the bwNET100G+ research project.
Names of collaborators and their shares	Marco Häberle Co-supervision of the master thesis (focus on implementation). Feedback on the publication. Arwed Mett: Development of the prototype in his master thesis. Michael Menth Scientific supervision of the master thesis. Editorial assistance on the publication.
Importance of own contributions to the joint work	Co-supervision of the master thesis (focus on concept and architecture). Lead author of the paper, taking up most of the write-up.

7. **A Survey on Data Plane Programming with P4: Fundamentals, Advances, and Applied Research** [HHM+21b]

Scope of the joint work	Literature study on the state-of-the-art of data plane programming with P4 covering fundamentals, advances, and applied research. Follow-up work emerged from the "P4-SOTA" research project.
Names of collaborators and their shares	Marco Häberle, Daniel Merling, Steffen Lindner: Analysis, classification and summarization of applied research works. Writing input and feedback on the foundation chapters. Help in structure definition and review of the complete manuscript. Vladimir Gurevich: Writing input and feedback on the foundation chapters. Florian Zeiger, Reinhard Frank: Principals of the P4-SOTA research project. Feedback on the structure and write-up of the manuscript. Michael Menth: Scientific supervision of the research project. Editorial assistance on the publication.
Importance of own contributions to the joint work	Responsible project manager for the P4-SOTA research project. Coordination of all activities around this large writing project. Lead author of the paper, taking over most of the write-up.

Accepted Manuscripts (Additional Content)

8. **LoCoSDN: A local controller for operation of switches in non-SDN networks** [SHGM18]

Scope of the joint work	Research work on local controllers in OF-based SDN identified and developed in the context of the bwNET100G+ research project.
Names of collaborators and their shares	Mark Schmidt: Co-supervision of the student research project. Assistance in manuscript writing. Preparation and presentation of the conference talk. Bastian Germann: Development of an elementary prototype in a student research project. Michael Menth: Scientific supervision of the student research project. Editorial assistance on the publication.
Importance of own contributions to the joint work	Co-supervision of the student research project. Extensive revision of the original concept and architecture. Lead author of the paper, taking over most of the write-up.

9. **A Master Course on Network Softwarization: Lectures and Practical Assignments** [HSM17a]

Scope of the joint work	University course with practical programming assignments for preparing master students for a student research project or master thesis in the context of the bwNET100G+ research project.
Names of collaborators and their shares	Mark Schmidt: Joint development, lecturing, and supervision of the course. Michael Menth: Supervision of the lecture. Editorial assistance on the publication.
Importance of own contributions to the joint work	Joint development, lecturing, and supervision of the course. Lead author of the paper, taking over most of the write-up. Preparation and presentation of the workshop talk.

10. **On Moving Averages, Histograms and Time-Dependent Rates for Online Measurement** [MH17b]

Scope of the joint work	Research work on time-dependent statistics for online measurement.
Names of collaborators and their shares	Michael Menth: Topic creator and lead author of the paper, taking over most of the write-up.
Importance of own contributions to the joint work	First implementation of all measurement algorithms in R. Assistance on evaluation experiments. Feedback on the structure and write-up of the manuscript. Preparation and presentation of the conference talk.

11. **Demo: Time Series Online Measurement for Python (TSOMpy)** [MH17a]

Scope of the joint work	Implementation of the algorithms and GUI-based demonstrator for the time-dependent statistic presented in the previous publication [MH17b]
Names of collaborators and their shares	Michael Menth: New implementation of all algorithms in Python. Dominik Krauß: Extension of the implementation by a GUI and various functions for dialogue-based data input and visualization.
Importance of own contributions to the joint work	Supervision of the GUI implementation. Assistance in the elaboration of the publication. Preparation and presentation of the conference demo.

12. **FunSpec4DTMC – A Tool for Modelling Discrete-Time Markov Chains Using Functional Specification** [HKM18]

Scope of the joint work	Implementation of the algorithms developed in a past work [Men11] of Michael Menth.
Names of collaborators and their shares	Dominik Krauß: Development of the program in his bachelor thesis. Assistance with feedback on conceptual and technological aspects of the manuscript. Preparation and presentation of the conference demo. Michael Menth: Scientific supervision of the bachelor thesis. Editorial assistance on the publication.

Importance of own contributions to the joint work	Co-supervision of the bachelor thesis. Lead author of the paper, taking over most of the write-up.

Publications

1 Accepted Manuscripts (Core Content)

1.1 P4-IPsec: Site-to-Site and Host-to-Site VPN With IPsec in P4-Based SDN

Received July 5, 2020, accepted July 16, 2020, date of publication July 29, 2020, date of current version August 10, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3012738

P4-IPsec: Site-to-Site and Host-to-Site VPN With IPsec in P4-Based SDN

FREDERIK HAUSER, (Graduate Student Member, IEEE), MARCO HÄBERLE, (Student Member, IEEE), MARK SCHMIDT, (Member, IEEE), AND MICHAEL MENTH, (Senior Member, IEEE)
Chair of Communication Networks, University of Tübingen, 72076 Tübingen, Germany

Corresponding author: Frederik Hauser (frederik.hauser@uni-tuebingen.de)

This work was supported by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/1-2 and the bwNET100G+ project which is funded by the Ministry of Science, Research and the Arts Baden-Württemberg (MWK). The authors alone are responsible for the content of this paper.

ABSTRACT In this work, we present P4-IPsec, a concept for IPsec in software-defined networks (SDN) using P4 programmable data planes. The prototype implementation features ESP in tunnel mode and supports different cipher suites. P4-capable switches are programmed to serve as IPsec tunnel endpoints. We also provide a client agent to configure tunnel endpoints on Linux hosts so that site-to-site and host-to-site application scenarios can be supported which are the base for virtual private networks (VPNs). While traditional VPNs require complex key exchange protocols like IKE to set up and renew tunnel endpoints, P4-IPsec benefits from an SDN controller to accomplish these tasks. One goal of this experimental work is to investigate how well P4-IPsec can be implemented on existing P4 switches. We present a prototype for the BMv2 P4 software switch, evaluate its performance, and publish its source code on GitHub [1]. We explain why we could not provide a useful implementation with the NetFPGA SUME board. For the Edgecore Wedge 100BF-32X Tofino-based switch, we presented two prototype implementations to cope with a missing crypto unit. As another contribution of this paper, we provide technological background of P4 and IPsec and give a comprehensive review of security applications in P4, IPsec in SDN, and IPsec data plane implementations. According to our knowledge, P4-IPsec is the first implementation of IPsec for P4-based SDN.

INDEX TERMS IPsec, P4, software-defined networking, VPN.

I. INTRODUCTION

Virtual Private Networks (VPNs) extend private networks across public networks by adding authentication and encryption to network traffic. Internet Protocol Security (IPsec) is one of the oldest, but still most widespread VPN protocols. Standardized by the IETF, it introduces protection on the Internet Protocol (IP) layer. Due to its large distribution, many implementations for network appliances and operating systems are available. Although it is criticized for its complexity, proven deployment patterns allow efficient and reliable operation.

IPsec tunnel setup requires user configuration plus keying material that is exchanged by IPsec peers via the Internet Key Exchange (IKE) protocol. The complexity grows with the number of IPsec peers, especially in highly dynamic environments such as campus or enterprise networks with many users and sites. Several works investigate how to leverage the centralized control plane of software-defined networking (SDN) to simplify IPsec operation. However, the possibilities for IPsec deployment in SDN were limited. Typical SDN switches have a fixed function data plane that does not provide support for IPsec. As a result, IPsec data plane processing needs to be moved to an additional software-based packet processing function (PPF). Besides being an additional component, this adds latency as traffic needs to be forwarded back and forth. Programmable data planes as offered by P4 are a game changer. Data plane behavior can be described in a high-level programming language. Those network programs can be executed by software or hardware devices. For IPsec, this means that instead of shifting IPsec functionality to PPFs, functions can be implemented directly on the data plane of SDN switches. In our previous work P4-MACsec [2], we introduced MACsec for P4-based SDN.

The associate editor coordinating the review of this manuscript and approving it for publication was Congduan Li.

We proposed a data plane implementation in P4 and introduced a novel concept for automated deployment and operation of MACsec.

In this paper, we present the first integration of IPsec VPN for P4-based SDN. We give an introduction on the technological background and provide an extensive survey on related work in that field. We present an IPsec data plane implementation that integrates IPsec components and processes with constructs and components under the given constraints of the P4 data plane programming language. Cryptographic operations for authentication, encryption, and decryption are implemented in P4 externs where IPsec components such as the Security Policy Database (SPD) and Security Association Database (SAD) are part of the P4 processing pipeline. P4 switches that implement the functionality of P4-IPsec can be deployed in host-to-site and site-to-site VPN scenarios. The control plane functions for IPsec operation are part of a central SDN controller that maintains IPsec tunnels without the help of distributed key exchange protocols such as IKE. As these components are steered by a centralized control plane through an authenticated and encrypted control connection, complex IKE-based key exchange protocols are substituted by controller-based tunnel setup and renewal procedures. For host-to-site operation, we introduce a client agent for Linux operating systems that runs on the roadwarrior hosts. It establishes an interface to the central SDN controller via a gRPC connection. To investigate how well P4-IPsec can be implemented on existing P4 targets, we work on three prototypes. We successfully implement a prototype for the Behavioral Model version 2 (BMv2) P4 software target and conduct a performance evaluation. We release the source code of our prototype with its testbed environment under the Apache v2 license on GitHub [1]. In addition, we report on implementation experiences for the NetFPGA SUME board and Edgecore Wedge 100BF-32X P4 switch. For the latter, we present two workaround implementations and compare them in performance experiments.

P4-IPsec introduces several benefits over traditional IPsec operation. First, we improve scalability by making switches and roadwarrior hosts stateless components whose functionality is only managed by an SDN controller. Second, we improve flexibility by converting P4 targets into IPsec endpoints, i.e., IPsec tunnels can terminate close to the network hosts that should be made accessible via the VPN. This limits the size of the perimeter and improves security through better isolation. Last, we encourage open networking research and operation. Network functionality can be modified in agile development processes, source code can be audited and improved by a larger audience.

The rest of the paper is organized as follows. Section II gives an overview on IPsec, VPN, and data plane programming with P4. In Section III, we describe related work on P4-based network security applications, IPsec in SDN, and IPsec data plane implementations. Section IV presents the architecture of P4-IPsec. In Section V, we describe the prototypical implementation of P4-IPsec with Mininet and BMv2. Section VI presents the performance evaluation of that prototype. In Section VII, we report implementation experiences for the NetFPGA SUME board and Edgecore Wedge 100BF-32X P4 switch. Section VIII concludes this work. The appendices include a list of the acronyms used in the paper.

II. TECHNICAL BACKGROUND

We give an introduction to VPN with IPsec and data plane programming with P4.

A. IPsec VPN

Internet Protocol Security (IPsec) is a widespread VPN protocol suite. It applies authentication and encryption on the Internet Protocol (IP) in host-to-host, site-to-site, and host-to-site communication scenarios. RFC 4301 [3] is the latest version of its specification.

1) PROTOCOLS

IPsec comprises the *Authentication Header (AH)* and *Encrypted Secured Payload (ESP)* protocol. AH [4] protects IP packets by sender authentication and packet integrity validation. It applies a hash function with a shared key (e.g., HMAC-SHA256) to calculate Integrity Check Values (ICVs) and adds packet sequence numbers to protect against replay attacks. ESP [5] protects the confidentiality of IP packets by symmetric encryption. As for AH, it also adds sender authentication, packet integrity validation, and protection against replay attacks. ESP supports symmetric ciphers such as Triple Data Encryption Standard (3DES), Blowfish, and Advanced Encryption Standard (AES). Ciphers that only apply encryption are combined with an authentication function. AES in cipher block chaining (CBC) or counter (CTR) mode are examples for such ciphers that might be combined with secure hash algorithm (SHA) for authentication. Authenticated encryption (AE) ciphers such as AES in galois/counter mode (GCM) [6] include both packet encryption and authentication. IPsec provides support for IP Payload Compression (IPComp) [7] so that the payload of IP packets can be compressed before encryption.

2) OPERATION MODES

IPsec can be deployed in either *transport* or *tunnel* operation mode. Transport mode protects IP traffic that is exchanged between two network hosts (host-to-host scenario). An AH or ESP header is inserted between the IP header and the IP payload. Tunnel mode protects IP traffic in host-to-host, host-to-site, and site-to-site communication scenarios. Figure 1 depicts how tunnel mode with ESP is applied to an IP packet. A new outer IP header with the IP addresses of the IPsec peers is created. The original IP packet is inserted between the ESP header and the ESP trailer. Encryption protects the original IP packet while authentication is applied to the complete ESP packet.

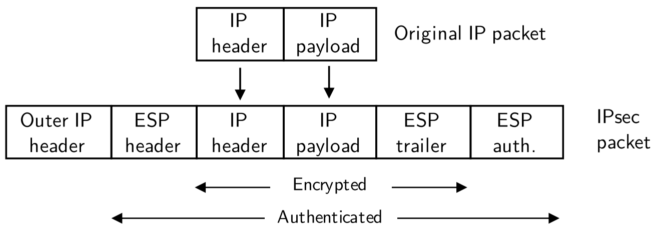


FIGURE 1. Tunnel mode with ESP. The original IP packet is inserted between the ESP header and the ESP trailer. The inner IP packet is encrypted while the complete ESP packet is authenticated.

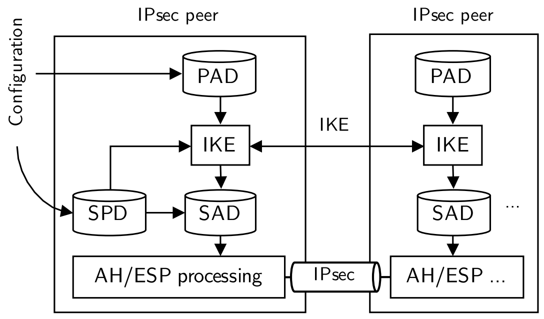


FIGURE 2. IPsec packet processing between two IPsec peers. Each peer features a SPD, SAD, PAD, and AH/ESP processing functions. The SPD and PAD are configured manually, where SAD entries are managed by the IKE daemon.

3) CORE COMPONENTS

We describe the core components of IPsec implementations that are part of hosts or gateways. The *Security Policy Database (SPD)* holds security policies that decide on traffic protection using IPsec. Entries have match keys, e.g., IP src/dst address, IP protocol, and TCP/UDP port, with an assigned action. IPsec allows three actions: DROP (discard packet), BYPASS (no protection), and PROTECT (apply IPsec protection). In case the table yields no match, the DROP action is applied. SPD entries for IPsec connections point to the protocol (AH/ESP), the operation mode (transport/tunnel), and the cipher suite. An IPsec tunnel between two peers is described by two unidirectional security associations (SAs). An IPsec SA contains all required data for AH/ESP processing, e.g., cipher keys, valid sequence numbers, and SA lifetimes for rekeying and tear down. SAs are part of the *Security Association Database (SAD)*. With the information from the SAD, packets then can be processed by *ESP/AH processing*. Although manual configuration of SAs is possible, they are typically configured between IPsec peers with the help of the Internet Key Exchange (IKE) protocol [8] that was introduced with IPsec. It authenticates both peers, sets up a secure channel for key exchange, and negotiates SAs. Today, its successor Internet Key Exchange v2 (IKEv2) [9] should be used. It is less complex and solves incompatibility issues of IKE. IKE relies on the *Peer Authentication Database (PAD)* for authentication of other IPsec peers.

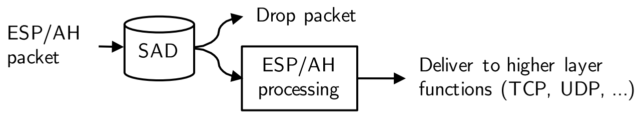


FIGURE 3. IPsec ingress processing. Arriving packets with an ESP/AH header are processed with the help of the SAD. ESP/AH processing relies on data in the SAD. In case of no match, the packet is dropped.

4) PACKET PROCESSING

IPsec differentiates between ingress and egress processing of packets. Figure 3 depicts ingress processing. Arriving packets that have an ESP/AH header are processed with the help of the SAD. If the SAD has an entry for the corresponding SA, the SA data is forwarded to the ESP/AH processing function that removes IPsec protection. Afterwards, the packet is forwarded to default network processing.

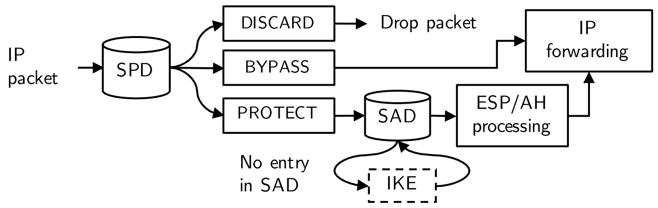


FIGURE 4. IPsec egress processing. The SPD matches packets and maps them to the actions DISCARD, BYPASS, and PROTECT. In case of BYPASS, the packet is passed to IP forwarding. In case of PROTECT, the SAD is searched for a corresponding entry for ESP/AH processing. In case of no match, the packet is dropped.

Figure 4 depicts egress processing where IP packets are matched with SPD entries as explained before. In case of PROTECT, data for ESP/AH processing is selected from the SAD. If the SAD has no matching entry, SA setup is requested from the IKE daemon. In case of BYPASS, the packet is passed to IP forwarding. In case of DISCARD, the packet is dropped.

5) DISCUSSION

Among more recent alternatives such as OpenVPN and WireGuard, IPsec is still one of the most widespread VPN technologies nowadays. IPsec implementations are part of most operating systems for computers, servers, and mobile devices. Most network hardware appliances, e.g., firewalls, routers, or security appliances, include an IPsec implementation.

However, IPsec is highly criticized for its complexity. The most encompassing analysis was performed by Ferguson and Schneier [10] in 2003. The authors mainly criticize the redundancy of functionality caused by AH, ESP, and the two operation modes, the complex key exchange with IKE, and the complex configuration caused by the SPD and SAD. However, those issues can be easily solved. Transport mode and AH should be avoided. Instead, AE ciphers that combine encryption and authentication should be used in conjunction with ESP with tunnel mode. IKE should be substituted by a less complex protocol for key exchange. In P4-IPsec, we follow those recommendations and restrict the IPsec implementation to ESP and tunnel mode with controller-based SA management without IKE.

B. DATA PLANE PROGRAMMING WITH P4

SDN introduces network programmability by shifting control plane functions to a software-based controller that determines the packet processing behavior of network devices. OpenFlow (OF) [11] is the most widely used SDN approach. It relies on data plane devices with a fixed set of functions and a southbound interface to the SDN controller. The SDN controller defines how these functions are applied to network packets. Programmable data planes extend network programmability to data plane functionality. Packet processing can be defined on an abstract layer using a dedicated programming language. Thereby, packet processing behavior is decoupled from the underlying hardware. This new principle facilitates open network research with support for agile development processes and flexible deployment options. Bifulco and Rétvári [12] give an overview on programmable data planes. Target platforms include software targets, network interface cards (NICs), NICs with a field programmable gate array (FPGA) unit, and hardware appliances with network processing units (NPUs). P4 is the most widely used data plane programming language nowadays. Initially presented as a research paper in 2014, the project is now standardized by the P4 Language Consortium under the Open Network Foundation (ONF). Its latest specification is version 16 ($P4_{16}$) [13].

1) PROCESSING PIPELINE

Figure 5 depicts a simplified view on the packet processing pipeline of P4. It consists of three core abstractions that help to express forwarding behavior.

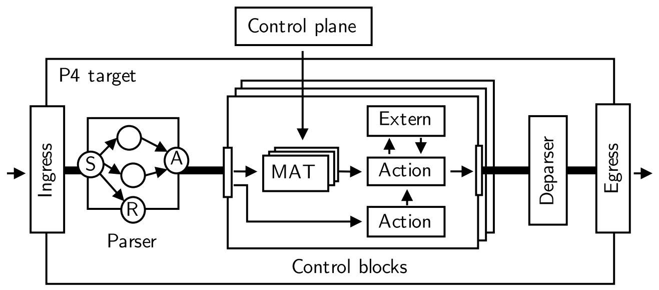


FIGURE 5. Simplified view on the P4 processing pipeline of $P4_{16}$. It comprises the parser, control blocks, and the deparser. Each control block may include MATs, actions, and externs.

a: PARSER

The parser extracts header fields of packets into internal data structures. P4 does not include predefined header types, i.e., programmers need to define packet formats and extraction behavior. Packet header formats are defined using P4 header types such as fixed- and variable-length bit strings or integers. The extraction behavior of the parser is expressed as finite state machine (FSM). Parsing is initiated in the state *start*, possible outcome states are *accept* (proceed in packet processing) and *reject* (drop the packet). Custom states that are positioned between start and end states implement the extraction of header data. The transitions between those states are formulated using conditions. For example, after successfully parsing an IP header, state transitions to TCP or UDP parsing might follow.

b: CONTROL BLOCKS

Control blocks are functions that modify packet headers and metadata. The P4 processing pipeline can include multiple control blocks that are typically separated by queues or buffers. Packet processing in control blocks is stateless: the outcome of packet processing applied to one packet can not influence packet processing applied on a subsequent packet. Actual packet processing is implemented in *actions*, code fragments within control blocks that implement read/write operations with functions provided by P4, e.g., setting header fields or adding/removing headers. Actions can be called from other actions, explicitly with the start of the control block, or implicitly by *match-action tables (MATs)*. MATs map match keys to particular actions with associated parameters. When applying a MAT to packets, the header and metadata are matched in exact, ternary, or in longest prefix manner against the keys of the MAT. If matching yields a particular row entry, the specified action is called with the associated parameters. If there is no match, a default action is applied. P4 programs only contain the declaration of MATs, their entries are maintained by a control plane via an application programming interface (API) in runtime. Some targets may provide additional functions for packet processing, e.g., particular functions such as checksum generation, or stateful components such as counters, meters, and registers. These components can be used within P4 programs as so-called *externs*. Externs have an interface with defined instantiation methods, functions, and parameters. After import and declaration, they can be used in control blocks just like any other P4 function.

c: DEPARSER

The deparser reassembles the packet header and payload and serializes it to be sent out via an egress port.

2) DEPLOYMENT MODEL

Software or hardware platforms that execute P4 programs are called P4 targets. Common software P4 targets are the BMv2 [14] software target, eBPF packet filters, and the T_4P_4S [15] software target that includes hardware interfaces via Data Plane Development Kit (DPDK) [16] and Open Data Plane (ODP) [17]. The hardware P4 targets include FPGA-based targets and NICs, NPU-based NICs, and whitebox switches featuring the Tofino application-specific integrated circuit (ASIC) from Barefoot Networks. P4 programs are implemented for a particular P4 architecture.

P4 architectures can be seen as programming models that represent the logical view of a P4 processing pipeline. They serve as an intermediate layer to decouple P4 programs from P4 targets, i.e., P4 programs that are implemented for a particular P4 architecture can be deployed to all P4 targets that implement this architecture. A front-end compiler translates P4 programs into a target-independent high-level intermediate representation (HLIR). Afterwards, the HLIR is compiled to the particular P4 target using a back-end compiler that is provided by the manufacturer.

3) CONTROL PLANE API: P4Runtime

The runtime behavior of P4 targets can be controlled by managing MATs or stateful components (e.g., counters, meters, registers, or externs) that are part of the P4 program. P4Runtime API [18] is a target- and program-independent API standardized by the P4 Language Consortium. P4Runtime uses gRPC for communication between the control plane and P4 targets and protobuf [19] data structures for packet serialization/parsing. gRPC connections can be secured with Transport Layer Security (TLS) and mutual authentication with certificates. In P4Runtime, the SDN controller establishes gRPC connections to preconfigured P4 targets. P4Runtime supports P4 object access (e.g., on MATs and externs), session management (master/slave controllers), role-based access control, and a packet-in/-out mechanism to receive and send out packets via the control plane. The PI Library is the reference implementation of the P4Runtime server that is part of P4 targets. It implements generic functionality for internal P4 objects such as MATs. This functionality can be extended by target- or architecture-specific configuration objects. *p4runtime_lib* [20] is an exemplary implementation of the P4Runtime API in Python to be used for building SDN controllers. P4Runtime API plugins are also available for common SDN controllers such as ONOS or OpenDaylight.

4) APPLICATION DOMAINS

Most research works on P4-based network applications target data center or wide area networks. In traffic management and congestion control, P4 is leveraged to implement new congestion notification mechanisms, novel traffic scheduling mechanisms, or novel mechanisms for active queue management. In routing and forwarding, special routing and forwarding mechanisms, publish-subscribe systems, or novel concepts from the area of named data networks are implemented. A large focus also lies on monitoring, where several works implement monitoring systems, sketch-based monitoring mechanisms, and in-band network telemetry (INT) systems. Besides, P4 is used in data center scenarios to implement switching, load balancing, network function virtualization (NFV), and service function chaining (SFC) mechanisms.

III. RELATED WORK

We describe related work on network security applications built with P4, IPsec in SDN, and implementation of IPsec packet processing.

A. NETWORK SECURITY APPLICATIONS WITH P4

Although network security is not the prevalent application domain of P4, some scientific work has been published in this field. We describe related work on firewalls, DDoS mitigation mechanisms, and other security applications.

1) FIREWALLS

Vörös and Kiss [21] introduce a P4-based firewall for filtering IPv4, IPv6, TCP, and UDP packets. It includes a ban list for instant drop, counters, e.g., for measuring the packet rate or unsuccessful connection attempts, and MATs for applying whitelist firewall rules. P4Guard [22] follows a similar approach. Its authors focus on simplified updated processes by deploying recompiled versions of the P4 program. Ricart-Sanchez *et al.* [23] implement a P4-based firewall for 5G networks. It includes parser definitions for filtering GPRS tunneling protocol data. CoFilter [24] introduces a hash function for efficient flow identification. It is built as P4 action and uses hashes instead of 5-tuples for flow identification to save table space. Including the function directly on the packet processing devices keeps latency low. Zaballa *et al.* [25] and Almaini *et al.* [26] introduce port knocking on P4 switches.

2) DDoS MITIGATION MECHANISMS

Paolucci *et al.* [27], [28] propose a DDoS mitigation mechanism that runs on P4 switches. A stateful mechanism detects and blocks DDoS port scan attacks with incremental TCP and UDP destination port numbers. Dimolianis *et al.* [29] also implement a DDoS attack mitigation mechanism that runs completely on P4 switches. The collected flow data is mapped to distinct time intervals where DDoS attacks are detected by analyzing the symmetry ratio of incoming and outgoing traffic. TDoSD@DP [30] implements a mitigation scheme against DDoS attacks on SIP proxies. The authors introduce a simple state machine that monitors SIP message sequences. Valid sequences of INVITE and BYE messages keep the port open. Febro *et al.* [31] implement another DDoS mitigation mechanism for SIP INVITE DDoS attacks. P4 switches keep per-port counters for INVITE or REGISTER packets that are monitored by an SDN controller to detect DDoS attacks. LAMP [32] implements cooperative mitigation of application layer DDoS attacks via in-band signaling with P4. Afek *et al.* [33] implement known mitigation mechanisms for SYN and DNS spoofing in DDoS attacks in P4. Lapolli *et al.* [34] describe a novel algorithmic approach based on the Shannon entropy to detect and stop DDoS attacks on P4 switches. Kuka *et al.* [35] introduce an FPGA-based system for DDoS attack mitigation. P4 is used to extract header data from packets and send it to an SDN controller where DDoS attack identification is implemented. Mi and Wang [36] propose a similar approach where collected data is sent to a deep learning module that runs on a server in the network.

3) OTHER SECURITY APPLICATIONS

Lewis *et al.* [37] implement an IDS offloading mechanism in P4. A rule parser translates Snort IDS rules into MAT entries for a P4 switch. Then, IDS pipeline stages decide if packets should be forwarded, dropped, or sent to an external IDS for analysis. Poise [38] is a security-related network control system that translates high-level policies into P4 programs for network control. In P4-MACsec [2], we implement IEEE 802.1AE (MACsec) in P4 and introduce an automated deployment mechanism provisions MACsec on detected links between P4 switches. Link monitoring is implemented using a novel variant of Link Layer Discovery Protocol (LLDP). It relies on encrypted payloads and sequence numbers to protect against LLDP packet manipulation and replay attacks.

B. IPsec IN SDN

Several works investigate the application of SDN to IPsec operation. We describe and discuss operation modes, southbound interfaces, and use cases.

1) OPERATION MODES

Related work can be categorized by three different operation modes that are depicted in Figure 6.

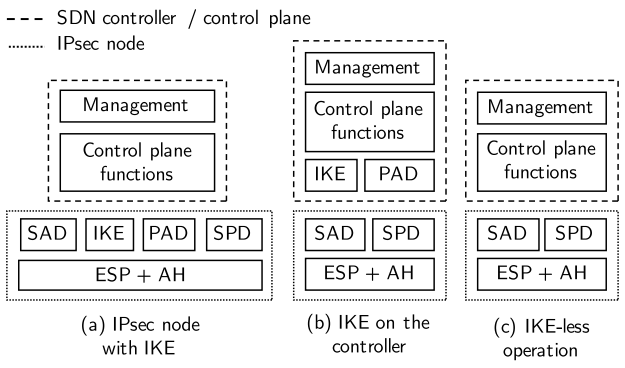


FIGURE 6. Operation modes for data plane management of IPsec. In (a), IKE is part of the IPsec node. In (b), IKE is part of the control plane. In (c), IKE is substituted by controller-based SA management.

a: IPsec NODE WITH IKE

In the first operation mode, IPsec processing nodes feature an IKE daemon, SDN assists in preconfiguration. Aragon *et al.* [39], [40] propose that an SDN controller preconfigures authentication keys in the PAD. Carrel and Weiss [41] propose that an SDN controller distributes Diffie-Hellman public values to all associated IPsec data plane nodes. Guo *et al.* [42] propose a similar approach that is compatible to older IKE daemons that only support IKEv1. Lopez-Millan *et al.* [43] propose an *IKE mode*, where the SDN controller only provides information for the configuration of the SPD, PAD, and IKE daemon. All proposals aim to reduce the message exchanges in an IKE process by preconfiguring it by an SDN controller.

b: IKE ON THE SDN CONTROLLER

In the second operation mode, the IKE daemon is part of an SDN controller. Son *et al.* [44] present an approach where the IKE daemon running on the SDN controller performs key exchange with peers and manages the SAD of the IPsec data plane nodes. This approach even supports migration schemes so that the SA can be transferred to other IPsec data plane nodes, e.g., in fail-over or load-balancing operations. Vajaranta *et al.* [45] describe a similar approach where IKE is executed as a network function that can be scaled up by creating additional instances.

c: IKE-LESS OPERATION

In the third operation mode, SAs are maintained without IKE. Lopez-Millan *et al.* [43] describe an *IKE-less* mode where the SA maintenance is delegated to an SDN controller. Here, the IPsec processing nodes only implement IPsec logic where the complete key management logic is moved to the SDN controller. As there is no IKE, no PAD is required. The authors differentiate between a proactive mode, where SPD and SAD are preconfigured by the SDN controller, and a reactive mode, where only the SPD is preconfigured by the SDN controller. Several works [39], [46]–[48] propose SA management without IKE. The SDN controller generates keying material and sets up SAs in the SAD of associated IPsec data plane nodes. Gunleifsen *et al.* [49] introduce a key management server that creates and distributes IPsec SAs for encryption virtual network functions (VNFs). In consecutive works, Gunleifsen *et al.* [50], [51] name this concept as *Software-Defined Security Associations (SD-SAs)*. Encryption VNFs only perform IPsec processing. SAs are created and distributed by an authentication center.

2) DISCUSSION ON OPERATION MODES

We briefly discuss the benefits and drawbacks of the three operation modes. The first operation mode benefits from easy migration. As legacy IPsec devices already feature an IKE daemon, they can be easily extended by an interface to profit from SDN-assisted operation of IPsec (see [43]). The second operation mode especially introduces flexibility and scalability. Separating IPsec processing and SA establishing to different entities improves scalability (see [45]). The third operation mode removes the overhead of peer-to-peer key exchange with IKE. In SDNs, IKE might be unnecessary as both IPsec peers are controlled by an SDN controller. Besides, IKE requires connectivity for IKE packet exchange between both peers which could be not given in particular scenarios (see [51]). Lopez-Millan *et al.* [43] show in an analytical evaluation that IKE-based and IKE-less operation of IPsec have approximately the same process loads in terms of messages and configuration data exchange.

3) SOUTHBOUND PROTOCOLS

On legacy network devices that feature IPsec devices, SNMP (e.g., [52]) is used for basic configuration and monitoring. Guo *et al.* [42] extend this usage in making an IKE daemon manageable by SNMP as well. Alharbi *et al.* [53] use SSH as a southbound interface to manage and monitor IPsec data plane nodes. Marin-Lopez *et al.* [46] use NETCONF with YANG configuration models. In addition to the southbound protocol, they consider east-/westbound interfaces for controller-to-controller communication via different domains. Aragon *et al.* [39] use OAuth 2.0 to deliver configuration data within authorization messages. Braadland [47] extends OpenFlow (OF) using experimenter messages for IPsec management. Sajassi *et al.* [48] leverage BGP. Li and Mao [54] use a custom southbound protocol to interface an IPsec extension module on an Open vSwitch. Li *et al.* [55] propose a custom southbound protocol with notification, configuration, and query messages that are transmitted via TCP or TLS. Lopez-Millan *et al.* [43] use NETCONF with YANG models as southbound protocol. Gunleifsen *et al.* [50], [51] rely on REST with JSON.

4) USE CASES

Use cases that benefit from controller-based operation of IPsec are SD-WAN, cloud provider networks, and dynamic VPN setup.

a: SD-WAN

Large organizations with distributed locations require network connectivity between the different sites. As dedicated links are expensive, site-to-site IPsec-VPNs over provider networks are increasingly used. However, manually setting up VPN connections between all branches is time-intensive and complex. SD-WAN [42], [53], [54] proposes IPsec data plane functionality as part of hardware appliances or software modules at the perimeter of the different sites of the organization. Then, a centralized SDN controller automatically sets up and maintains IPsec-VPN connections.

b: CLOUD PROVIDER NETWORKS

Often, internal services offered by a public or private cloud provider need to be accessed from within networks of an organization. Again, site-to-site IPsec-VPN tunnels are a cost-efficient alternative to dedicated links. Administrators define IPsec-VPN gateways via a cloud management interface. Then, the cloud orchestrator deploys IPsec-VPN gateways as VNFs on the cloud provider's infrastructure. Its runtime operation is managed by a SDN controller. In addition, controller-based operation of IPsec can be also used to dynamically connect different cloud networks by a multi-cloud orchestrator [56]. Gunleifsen *et al.* [49], [50] propose hop-by-hop protection for SFCs using IPsec and controller-based operation.

c: DYNAMIC VPN SETUP

Managing many IPsec-VPN connections to different hosts or services on a client host can be cumbersome. Dynamic VPN setup performed by a SDN controller takes over the tasks of tunnel setup and management. Van der Pol *et al.* [57] present a concept where users request VPN access to a particular network device from the SDN controller. It then automatically sets up a VPN tunnel to the remote domain. Aragon *et al.* [39] combine dynamic VPN setup with authentication and authorization to automatically deploy IPsec-VPN tunnels between IoT network devices. This introduces several advantages over traditional deployment. First, the control plane has an encompassing view on the network topology with all devices. It can monitor usage and detect outages for reliable operation. Second, the centralized control plane features northbound interfaces for management applications and southbound interfaces for controlling data plane devices. Instead of manual per-device configuration, VPNs are operated via a management layer with policy languages that allow rule validation. Last, the centralized control plane offers flexibility so that the VPN operation can be extended by other mechanisms, e.g., user authentication with 802.1X [54].

C. IMPLEMENTATION OF IPsec PACKET PROCESSING

With P4-IPsec, we present the first data plane implementation of IPsec in P4. We give an overview on IPsec data plane implementations as related work.

1) SOFTWARE IMPLEMENTATIONS WITH HARDWARE ACCELERATION

IPsec software programs represent the simplest packet processing implementations. Their I/O performance depends on the hardware, the chosen cryptographic algorithm, and the average packet size. For Linux host systems, optimization techniques such as DPDK [16], Netmap [58], and PF_RING [59] tweak network stack processing to increase packet I/O rates. Other works propose to increase IPsec packet I/O by using multiple CPU cores [60], [61] or the GPU [62]. Gallenmüller *et al.* [63] compare several mechanisms in an extensive study. Most of the described optimization mechanisms are only applicable to Linux operating systems.

IPsec packet I/O of software implementations can be improved by offloading crypto operations or IPsec operations to hardware. For the former, current CPU architectures provide hardware acceleration for common crypto operations. AES-NI [64] or ARMv8 Cryptographic Extensions [65] are examples of AES instruction sets that replace pure software implementations. System on a chip (SoC) platforms or circuit boards may contain chips for offloading cryptographic processing. Examples are the Marvell Cryptographic Engines Security Accelerator (CESA) or Intel QuickAssist [66]. Such processors can be also part of extension circuit boards that are connected to the mainboard via PCI. FPGAs might be also used for implementing crypto operations, several vendors (e.g., [67]) supply implementations of cryptographic algorithms as program cores. For the latter, IPsec hardware accelerators are available as ASIC [68], [69], NPU [70], [71], accellerated processing unit (APU) [72], or FPGA [73], [74].

2) HARDWARE IMPLEMENTATIONS

Proprietary IPsec hardware concentrators, e.g., as sold by Cisco or Juniper, are optimized for high IPsec I/O rates and, therefore, might implement a larger degree of the overall IPsec processing operations in hardware (e.g., on ASICs). Due to their disclosed architectural details, we cannot get insight into technical details. In addition, encompassing IPsec implementations for FPGAs exist [75], [76] where only the SPD and SAD are managed by an SDN controller.

3) IMPLEMENTATIONS ON PROGRAMMABLE DATA PLANES

In 2016, a Xilinx employee reported on the P4-Development mailing list [77] that IPsec was successfully implemented in PX [78], a high-level domain specific programming language for programmable data planes. Crypto primitives are implemented via an extern mechanism similar to P4 externs. The authors report that the crypto primitives were programmed as Register Transfer Level (RTL) designs targeting FPGAs. The authors report that the principle should be the same for P4, but it was not ported so far.

IV. CONCEPT

We describe the concept of P4-IPsec. We give an overview, discuss design choices, and describe its data plane and control plane in detail.

A. OVERVIEW

Figure 7 gives an overview on the functionality of P4-IPsec.

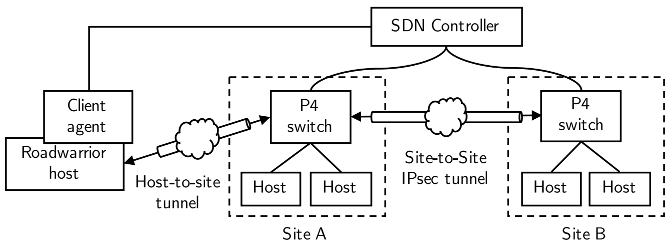


FIGURE 7. Overview on the functionality of P4-IPsec. In host-to-site operation, roadwarrior hosts run a client agent to set up an IPsec tunnel to a P4 switch via the SDN controller. In site-to-site operation, the SDN controller sets up IPsec tunnels for pairs of P4 switches.

P4-IPsec supports two IPsec tunnel operation modes: *host-to-site* and *site-to-site*. In host-to-site mode, roadwarrior hosts establish IPsec tunnels to get access to internal networks. Roadwarrior hosts run a client agent that interacts with the SDN controller for tunnel setup. In site-to-site mode, two internal networks are connected via an IPsec tunnel that is established between two P4 switches. As a core principle of P4-IPsec, every P4 switch implements the same IPsec functionality, i.e., it can act as both IPsec tunnel endpoint for roadwarrior hosts in host-to-site mode and for other P4 switches in site-to-site mode. This facilitates very flexible deployments where IPsec tunnels do not necessarily terminate at a central VPN concentrator but can be distributed to many P4 switches instead.

B. DESIGN CHOICES

P4 programs describe the packet forwarding behavior of switches or routers. Thereby, an implementation of IPsec in P4 is limited to data-plane-centric parts. Additional mechanisms such as IKE need to be part of an SDN controller implementing the control plane and interfacing the P4 switch. For P4-IPsec, our integration of IPsec in P4, we make the following design choices:

1) USE OF IKE-LESS OPERATION MODE

Referring to the results of Lopez-Millan *et al.* [43] (see Section III-B.1), we choose to implement IKE-less SA management via the SDN controller. Our proposed P4 processing pipeline comprises equivalent representations for the SAD and SPD that are both maintained by the SDN controller. Due to the lack of IKE, no PAD is required. Selecting IKE-less operation mode does not exclude an integration of IKE on the SDN controller at a later stage.

2) RESTRICTION TO ESP IN TUNNEL MODE

To keep our proposed concept as minimalistic as possible, we adopt the recommendations of Ferguson and Schneier [10] and restrict our implementation to ESP in tunnel mode.

3) IMPLEMENTATION OF CIPHER SUITES WITH EXTERNS

P4 does not provide functions for encryption, decryption, and message authentication. In contrast to related work, we decide against offloading IPsec processing to external software-based processing nodes and implement cipher suites with the help of P4 externs (see Section II-B). This should decrease the latency introduced by external processing while keeping the overall system more minimalistic. Each cipher suite is implemented by two externs; one that implements encryption functionality, and one that implements decryption functionality.

4) PROTOTYPE SIMPLIFICATIONS

We limit P4-IPsec to IPv4 and omit support for IPv6. We also omit support for IPComp. For applicability in experiments, we implement simple L3 forwarding based on longest-prefix matching (LPM). Clearly, this is not a requirement from the IPsec standard.

C. DATA PLANE OF P4-IPsec

We first give an overview on the P4 processing pipeline of P4-IPsec. For the sake of simplicity in presentation, we combine functions into function blocks and describe them later in detail.

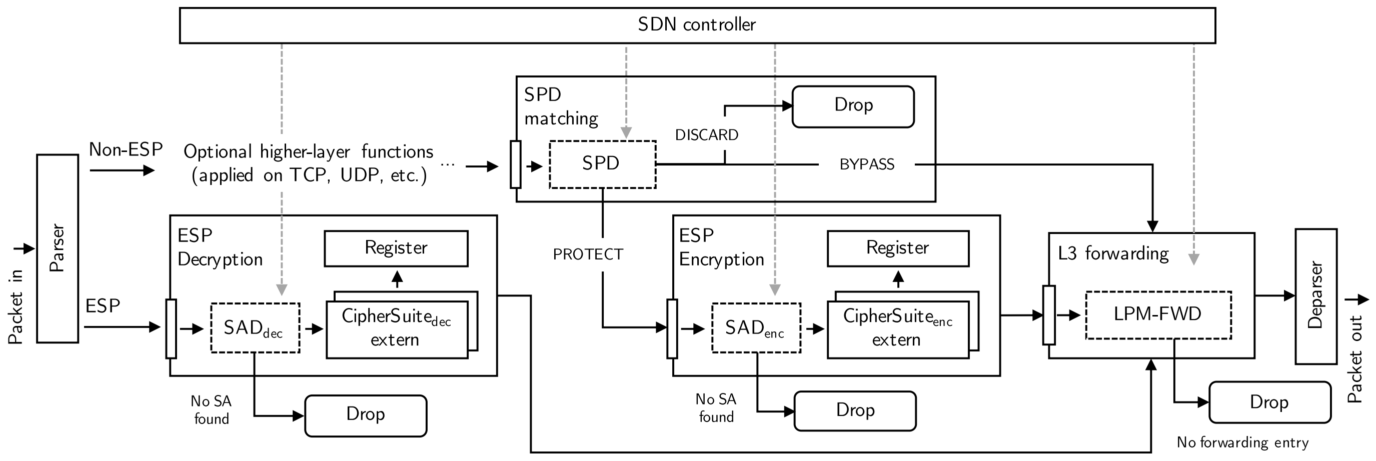


FIGURE 8. Data plane processing pipeline of P4-IPsec. For ease of understanding, related functionalities are grouped together as function block.

1) P4 PROCESSING PIPELINE

Figure 8 depicts the P4 packet processing pipeline of P4-IPsec. It consists of a parser, a deparser, and four function blocks in between. When a packet arrives via the ingress, the P4 parser first extracts the packet headers. In case of a header other than ESP, the parser forwards the packet to optional higher-layer functions that operate on protocol layers such as TCP or UDP. Afterwards, the SPD matching function block processes the packet. Following the IPsec standard, entries in the SPD determine about the action to be executed on the packet. In case of DISCARD, the packet is dropped. In case of BYPASS, the packet is passed to the L3 forwarding function block. In case of PROTECT, the packet is passed to the ESP encrypt function block. In the ESP encryption function block, encryption using SA data from the SAD_{enc} MAT is applied to the IP packet. In the L3 forwarding function block, the packet is forwarded based on the rules defined in the forwarding MAT. Going back to the parser again: if the packet has an ESP header, it is forwarded to the ESP decryption function block. It validates the packet's authenticity, decrypts the ESP message, and extracts the original IP packet that is then passed to the L3 forwarding function block. In case of missing entries in the SPD, SAD_{dec}, SAD_{enc}, or LPM-FWD MAT, the packet is dropped. As the final step, the deparser reassembles all headers and re-calculates the IPv4 checksum as some fields, e.g., the time to live (TTL), are changed. Runtime behavior of the data plane can be managed by manipulating the MATs via an SDN controller.

2) FUNCTION BLOCK: L3 FORWARDING

Figure 9 depicts the function block of *L3 forwarding*. It implements packet forwarding to the next hop via a particular output port of the P4 switch. The LPM-FWD MAT matches packets using their IPv4 destination addresses to two actions: *forward_packet* and *drop*. The forward_packet action receives the MAC address of the next hop and the output port as parameters from the MAT. Then, it sets the MAC

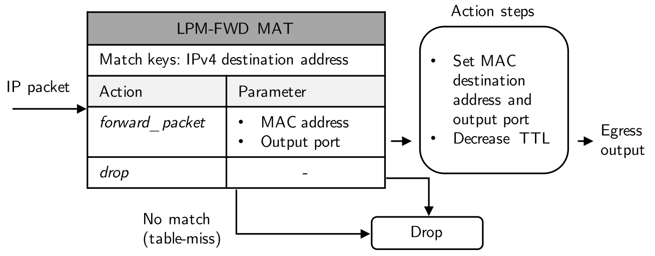


FIGURE 9. L3 forwarding function block. IP packets are processed by the LPM-FWD MAT that either applies the forward_packet or drop action. In case of no match, the drop action is applied.

destination address of the packet to the MAC address of the next hop, decreases the TTL by 1 in the IP header, and sets the output port. Afterwards, the packet is forwarded to the deparser and sent out via the egress. *drop* directly discards the packet; this action is also applied if no match in the LPM-FWD MAT is found.

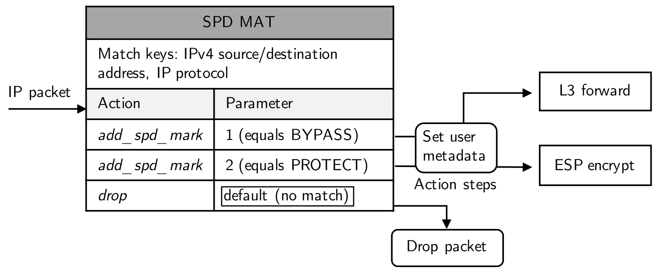


FIGURE 10. SPD matching function block. IP packets are processed by the SPD MAT. The add_spd_mark action adds the given parameter to the user metadata of the packet. It decides if the packet is protected by IPsec (PROTECT) or forwarded without protection (BYPASS) in later stages.

3) FUNCTION BLOCK: SPD MATCHING

Figure 10 depicts the function block of *SPD matching*. We introduce a security policy (SP) MAT that resembles the SPD from the IPsec standard (see Section II-A). It matches given packets with SPD rules and adds a mark to the user metadata of each packet that is used in further processing within the P4 processing pipeline. We implement IPv4 source and destination address and IP protocol as exemplary match keys. Due to P4's flexibility in defining packet parsing, more match keys, e.g., TCP/UDP ports or even application-layer ports, could be added easily. Actions are either *add_spd_mark* or *drop*. The *add_spd_mark* action adds "spd_mark = 1" for BYPASS or "spd_mark = 2" for PROTECT to the user metadata field of the packet. *drop* directly discards the packet; this action is also applied if no match is found.

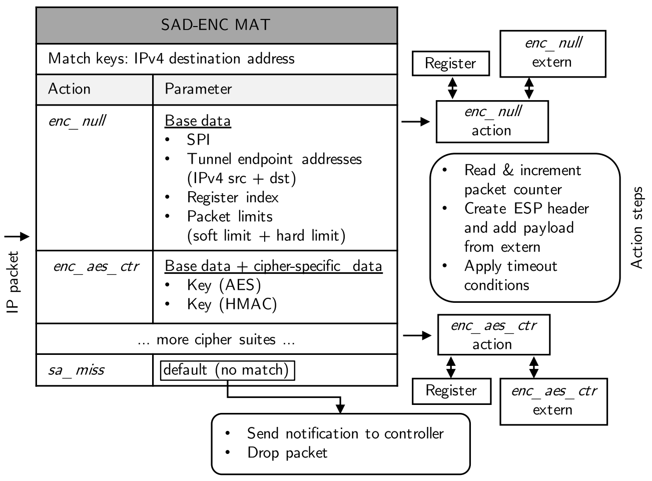

SAD-ENC MAT
Match keys: IPv4 destination address
Action
Parameter
enc_null
Base data
SPI
Tunnel endpoint addresses (IPv4 src + dst)
Register index
Packet limits (soft limit + hard limit)
enc_aes_ctr
Base data + cipher-specific data
Key (AES)
Key (HMAC)
... more cipher suites ...
sa_miss
default (no match)
IP packet
Register
enc_null extern
enc_null action
Read & increment packet counter
Create ESP header and add payload from extern
Apply timeout conditions
Action steps
enc_aes_ctr action
Register
enc_aes_ctr extern
Send notification to controller
Drop packet

FIGURE 11. ESP encryption function block. IP packets are processed by the SAD-ENC MAT. It holds entries for each SA with the corresponding data that is required for applying the associated cipher suite externs for encryption.

4) FUNCTION BLOCK: ESP ENCRYPTION

Figure 11 depicts the function block of *ESP encryption*. We introduce a SAD-ENC MAT that resembles the SAD from the IPsec standard (see Section II-A). Each entry in the MAT represents a particular SA that is identified by the IPv4 destination address, i.e., packets are matched based on their IPv4 destination address.

We implement cipher suites as actions that rely on externs and registers. Representing a variety of cipher suites, we implement the *AES-CTR* and *NULL* cipher suites as examples. The *NULL* cipher suite is intended for testing purposes only. It uses the identity function instead of encrypting data and skips calculating an integrity check value. Cipher suite actions receive two types of parameters: base data that is required by all cipher suites and cipher-specific data.

We first describe base data. The *Security Parameter Index (SPI)* is part of the ESP header. It identifies the SA. The *tunnel endpoint addresses* (IPv4 source/destination address) identify the source and destination of the IPsec tunnel. Both are part of the new outer IPv4 header that encapsulates the ESP frame. The *register index* points to a particular index that holds the packet counter for the particular SA used by the cipher suite extern. *Packet limits* declare timeout conditions in terms of packet count thresholds for SAs. If a soft limit is reached, rekeying is triggered. If a hard limit is reached, packets that belong to that SA are dropped.

The NULL cipher suite is an example that only requires this set of base data. Typical cipher suites that implement particular encryption and authentication mechanisms require additional parameters such as keys, initialization vectors (IVs), or even additional constructs to keep cipher state, e.g., registers. AES-CTR, as an example for such a cipher suite, requires a key for AES and a key for keyed-hash message authentication code (HMAC).

The functionality within the cipher suite action is as follows. First, the packet counter for the particular SA is read from the register and incremented. Second, an ESP header is created with the SPI and sequence number of the packet. For the creation of the ESP packet, the action passes the original IP packet, the newly created ESP header, and the required keys of the cipher suite to the corresponding extern. The cipher suite extern performs encryption/authentication and responds with the ESP packet. Fourth, the new outer IP packet is created with the tunnel endpoint addresses. It encapsulates the newly created ESP packet. Last, timeout conditions are checked. The user metadata structure includes flags for *soft_limit_reached* and *hard_limit_reached* that are set in case of matching conditions. For soft_limit_reached, the packet is forwarded to the SDN controller to trigger tunnel renewal. In case of hard_limit_reached, the packet is dropped.

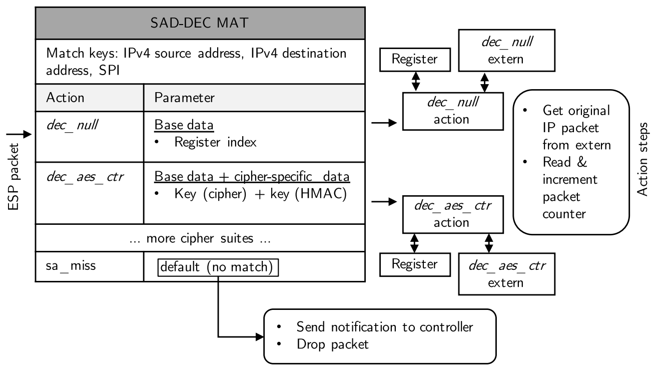

SAD-DEC MAT
Match keys: IPv4 source address, IPv4 destination address, SPI
Action
Parameter
dec_null
Base data
Register index
dec_aes_ctr
Base data + cipher-specific data
Key (cipher) + key (HMAC)
... more cipher suites ...
sa_miss
default (no match)
ESP packet
Register
dec_null extern
dec_null action
Get original IP packet from extern
Read & increment packet counter
Action steps
dec_aes_ctr action
Register
dec_aes_ctr extern
Send notification to controller
Drop packet

FIGURE 12. ESP decryption function block. ESP packets are processed by the SAD-DEC MAT. It holds entries for each SA with the corresponding data that is required for applying the associated cipher suite externs for decryption.

5) FUNCTION BLOCK: ESP DECRYPTION

Figure 12 depicts the function block of *ESP decryption*. We introduce the SAD-DEC MAT that resembles the decryption SAD from the IPsec standard (see Section II-A). Each entry in the MAT represents a particular SA that is identified by the outer IPv4 source and destination address (tunnel endpoints), and the SPI. As in the function block of ESP Encryption, cipher suites are implemented as actions that rely on externs and registers with a different set of action parameters.

The functionality within the cipher suite action is as follows. First, the packet counter for the particular SA is read from the register and incremented. Second, the original IP packet is extracted from the ESP packet. Therefore, the action passes the ESP packet and the required keys of the cipher suite to the corresponding extern. The cipher suite extern performs decryption/authentication and responds with the original IP packet. Last, timeout conditions are checked as described in ESP encryption.

D. CONTROL PLANE OPERATION OF P4-IPsec

We first give an overview on the control plane operation of P4-IPsec. We describe how configuration data is generated on the SDN controller and how it is set up in both host-to-site and site-to-site operation mode.

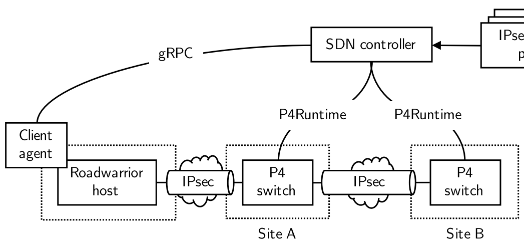


FIGURE 13. Control plane operation in P4-IPsec. The client agent on the roadwarrior host holds a control channel via gRPC to the SDN controller. P4 switches are connected via P4Runtime. The SDN controller includes IPsec tunnel profiles with configuration data for tunnel setup and management.

1) OVERVIEW

Figure 13 depicts the control plane interaction in the two operation modes of P4-IPsec, host-to-site and site-to-site mode. In both operation modes, IPsec tunnels are set up by the SDN controller on the basis of IPsec tunnel profiles. Those can be manually defined by an administrator or generated by another software component, e.g., a network operation platform. In host-to-site operation mode, the SDN controller interacts with the client agent via a gRPC tunnel and with the P4 switch via P4Runtime. In site-to-site operation mode, the SDN controller interacts with both P4 switches via P4Runtime. On roadwarrior hosts, configuration data is converted into *ip xfrm* commands that set up the tunnel. For P4 switches, the SDN controller directly writes to MATs and receives notifications, e.g., if an SA needs to be renewed, via P4Runtime. For the sake of simplicity, we restrict our implementation to proactive IPsec tunnel setup. In site-to-site mode, IPsec tunnels are set up and kept alive for all configured P4 switches. For host-to-site mode, the client agent presents a selection of available tunnels. The user then can select one or multiple IPsec tunnel profiles to be set up by the SDN controller.

This mechanism can be extended or substituted by more sophisticated approaches such as on-demand VPN setup. Predefined conditions (e.g., a request for a network resource in an internal network) may trigger IPsec tunnel setup via the SDN controller.

2) IPsec TUNNEL PROFILES

An IPsec tunnel between two peers consists of two unidirectional SAs, each identified by a unique SPI. Due to their direction, the first peer of an SA is called "left" where the second peer of the SA is called "right". We denote the SA from the left to the right peer as SPI_i and the SA from the right to the left peer as SPI_j, respectively. Each SA requires two MAT entries: one for encrypting ESP packets in the SAD-ENC MAT and one for decrypting ESP packets in the SAD-DEC MAT.

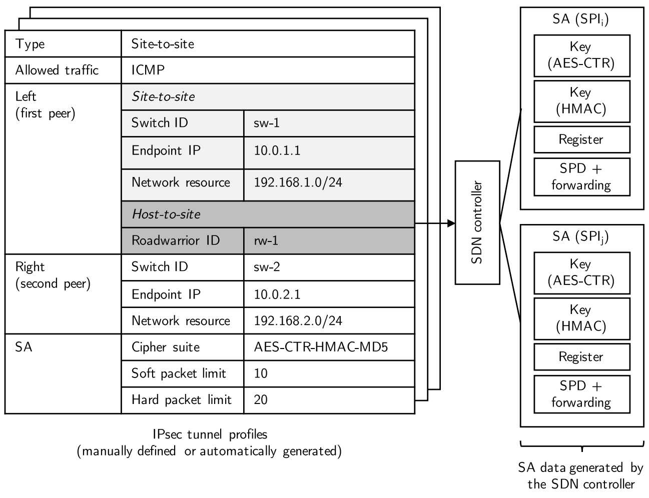


FIGURE 14. IPsec tunnel profiles with the associated SA data generated by the SDN controller. The configuration data in the IPsec tunnel profiles depends on the operation mode (host-to-site or site-to-site). SA data depends on the cipher suite that is defined in the IPsec tunnel profile.

Figure 14 depicts how the SDN controller generates configuration data for the roadwarrior hosts or P4 switches. IPsec tunnel profiles are the basis for any IPsec tunnel. As basic information about the tunnel, they include information about the type of IPsec tunnel (host-to-site or site-to-site) and the allowed traffic that is set to PROTECTED via SPD rules. The left peer (first) can be a P4 switch (in site-to-site operation mode) or a roadwarrior host (in host-to-site operation mode). In case of site-to-site operation mode, this field holds the switch ID (unique identifier of the P4 switch), endpoint IP (public IP address of the P4 switch), and network resource (internal network behind the P4 switch). In case of host-to-site operation mode, this field only holds the roadwarrior ID. The right peer (second) is always a P4 switch. Therefore, it holds the same data as in the left peer field in site-to-site operation mode as described before. The SA field holds the cipher suite and soft/hard packet limits. On the basis of an IPsec tunnel profile, the SDN controller generates configuration data for both SAs. In case of the AES-CTR-HMAC-MD5 cipher suite, SA data includes keys for AES-CTR and HMAC, register indexes, and configuration data for the SPD and forwarding function block. In case of the NULL cipher suite, keying material is not needed.

In our prototype, IPsec tunnel profiles are manually defined by an administrator. In practice, they can be generated by a software component, e.g., a network operation platform, on the basis of user/device profiles, groups, network resources, and permission models.

3) CONTROLLER CONNECTION

We describe the management connections in site-to-site and host-to-site operation mode.

a: SITE-TO-SITE OPERATION MODE (P4Runtime)

Site-to-site operation mode relies on P4Runtime for managing the P4 switches. Explained in Section II-B.3, the control plane connection to the P4 switches is established by the SDN controller. Therefore, it holds a list of connection data (name, address, port identity) of all assigned P4 switches.

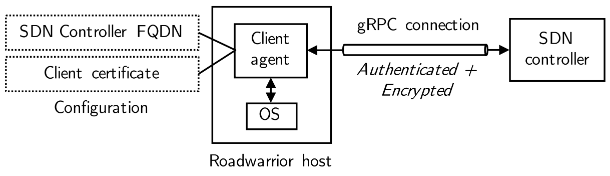


FIGURE 15. Control plane connection between the client agent and SDN controller. The client agent depends on the FQDN of the SDN controller and a client certificate to establish a gRPC connection to the SDN controller.

b: HOST-TO-SITE OPERATION MODE (gRPC)

Figure 15 depicts the connection between the client agent running on the roadwarrior host and the SDN controller. The required configuration data for the start of the client agent are the FQDN of the SDN controller and the client certificate. At start, the client agent establishes a gRPC tunnel to the SDN controller. The gRPC tunnel is protected with TLS, i.e., the client agent and SDN controller perform a mutual authentication using certificates and establish an encrypted connection. Certificates can be created and deployed to all roadwarrior hosts running the client agent and the SDN controller with a public key infrastructure (PKI). Roadwarrior host access can be removed by simply revoking the associated client certificate. In addition, gRPC provides support for optional multifactor authentication with token-based authentication via the Google Authenticator service. After connection setup, the client agent and SDN controller exchange configuration and signaling data via the gRPC tunnel. The client agent implements interfaces to interact with the roadwarrior host's operating system for configuration and signaling. The management connection to the P4 switch as a remote peer is established with P4Runtime as described before.

Host-to-site operation mode requires that the SDN controller is dual-homed. It has an interface to a management network where it holds P4Runtime connections to the P4 switches and another interface that makes it accessible via the Internet for client agents running on roadwarrior hosts. On the latter interface, it has an IP address that is publically

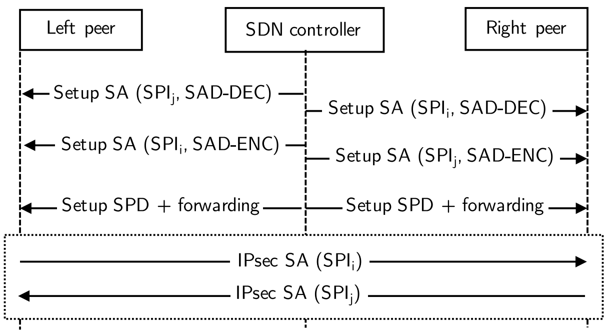


FIGURE 16. Setup procedure for a bidirectional IPsec tunnel on two peers. The SDN controller sets up decryption and encryption for both SAs followed by SPD and forwarding entries.

reachable via the Internet. Although mutual certificate-based authentication protects against malicious P4-IPsec agents, this public interface should be protected as every publically available web service, e.g., with a firewall.

4) TUNNEL MANAGEMENT OPERATIONS

We describe the elementary management operations of tunnel setup, tunnel renewal, and tunnel deletion that apply to both operation modes.

a: TUNNEL SETUP

IPsec tunnel setup is a three-step process. First, the SDN controller sets up both SAs in the SAD-DEC MATs of both peers. Second, the SDN controller sets up both SAs in the SAD-ENC MATs of both peers. Setting up SA entries for decryption first ensures that no ESP packets get lost if one peer immediately starts to send ESP packets. Last, the SDN controller sets up the SPD and installs forwarding rules if required.

b: TUNNEL RENEWAL

IPsec SAs have a limited lifetime, i.e., keying material needs to be renewed on a regular basis. Both client agent and P4 switch notify the SDN controller if an SA needs to be renewed. For the client agent, this notification is triggered by the kernel implementation of IPsec that sends expiration messages in the case that soft and hard timeout limits are reached. For the P4 switches, this is implemented using packet counters in registers that are checked with each packet processing within an ESP encryption or decryption function block. We adopt the principle presented by Lopez-Millan *et al.* [43] that implements tunnel renewal without risking packet loss. When the SDN controller received the SA expire notification, it generates a new SA that is identified by a new SPI. Then, tunnel renewal follows the principle of tunnel setup as described before. First, the new SA is installed in the SAD-DEC MAT. Second, the existing SA in the SAD-ENC MAT is replaced by the new SA via a modify operation. Last, as it can be ensured that no packets are encrypted using the previous SA, its entry can be removed from the SAD-DEC MAT.

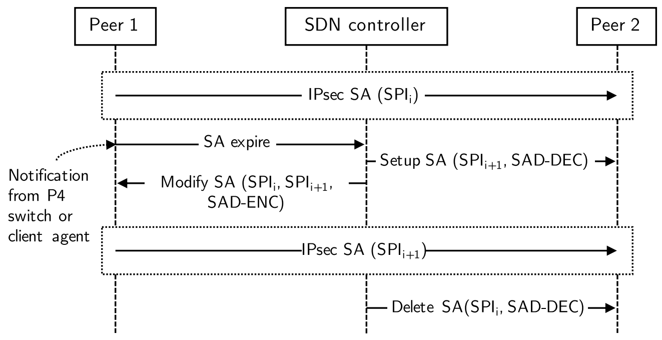

Peer 1
SDN controller
Peer 2
IPsec SA (SPI$_i$)
SA expire
Setup SA (SPI$_{i+1}$, SAD-DEC)
Notification from P4 switch or client agent
Modify SA (SPI$_i$, SPI$_{i+1}$, SAD-ENC)
IPsec SA (SPI$_{i+1}$)
Delete SA(SPI$_i$, SAD-DEC)

FIGURE 17. Renewal procedure for a unidirectional SA within an IPsec tunnel on two peers. After receiving an SA expire message, the SDN controller installs a new decryption SA on the remote peer. Afterwards, it replaces the expired encryption SA with new SA data. As a cleanup step, the old decryption SA is removed.

c: TUNNEL DELETION

If an IPsec tunnel should be deleted, the SDN controller removes the associated entries in the SPD, SAD-ENC, and SAD-DEC MAT. This is triggered on the SDN controller, e.g., when an IPsec profile is removed.

V. PROTOTYPICAL IMPLEMENTATION

We describe our software-based prototype of P4-IPsec. We present the testbed environment and outline the three parts of the implementation in detail. We publish the source code for both parts under the Apache v2 license on GitHub [1].

A. TESTBED ENVIRONMENT

Our prototypical implementation of P4-IPsec includes a softwarized testbed environment. We use Mininet to create network topologies that consist of BMv2 P4 switches and network hosts. We build it with Vagrant [79], a tool that simplifies the creation and management of virtual environments. All resources and setup steps are part of a configuration file. Executing *vagrant up* in a console within the repository folder automatically sets up and launches the testbed environment. It includes a virtual machine running Ubuntu 16.04 with all dependencies: libyang, sysrepo, mininet, protobuf, gRPC, PI/P4Runtime, BMv2, and P4C. The versions of all components can be found in the setup scripts.

B. DATA PLANE IMPLEMENTATION

We implement the P4 data plane implementation for the BMv2 P4 software target. We extend its *simple_switch* architecture by externs programmed in C++ for the AES-CTR-HMAC-MD5 and NULL IPsec cipher suites. Each cipher suite is implemented by two externs, one for encryption and one for decryption. For AES-CTR-HMAC-MD5, we use OpenSSL to apply AES-CTR for encryption/decryption and HMAC-MD5 for packet authentication. We implement the P4 processing pipeline as P4$_{16}$ program. It relies on the cipher suite externs and uses registers to store packet counters for the SAs. We run the P4 program on our extended *simple_switch* P4 target. We encapsulate our modified simple_switch P4 target within the *simple_switch_grpc* P4 target so that P4Runtime API can be used for interaction with the SDN controller.

C. CLIENT AGENT

We implement the client agent as Python 3.6 command line tool for Linux hosts. We integrate a gRPC client using the gRPC library [80] as an interface to the SDN controller. For IPsec tunnel setup, the client agent translates configuration data from the SDN controller into particular *XFRM* commands from the *iproute2* tool to configure IPsec on the roadwarrior host. In addition, it sets up IP routes for routing IP traffic via the IPsec tunnel. The received and applied configuration data is cached so that proper teardown configuration can be applied in case of tunnel shutdown. We implement rekeying with the help of *Netlink* [81]. The client agent monitors Netlink messages by listening on the corresponding Netlink socket and binding to the XFRMNLGRP_EXPIRE address so that XFRM Expire messages can be received. When receiving an XFRM Expire message, it extracts parameters such as SPI and IP addresses of the tunnel endpoints and notifies the SDN controller for tunnel renewal.

D. SDN CONTROLLER

We implement the SDN controller as a command line tool in Python 2.7. We use the p4runtime_lib [20] to integrate the interface to the P4 switch and the gRPC library to integrate the interface to the client agent. The SDN controller features a simple command line interface (CLI) for development and testing purposes that displays information about all active IPsec tunnels. P4Runtime and p4runtime_lib facilitate easy implementation of individual SDN controllers for prototypes. Nevertheless, those functions could be also integrated into existing SDN controllers such as ONOS or OpenDaylight.

VI. PERFORMANCE EVALUATION WITH THE SOFTWARE SWITCH BMv2

We describe the testbed environment and report the experiment results performed with the P4-IPsec software prototype introduced in Section V.

A. METHODOLOGY

We conduct the performance experiments in the testbed environment presented in Section V-A. The testbed runs on a Lenovo Thinkpad T480s (Intel i5-8250U CPU, 16 GB RAM) with Manjaro Linux. The Vagrant file in the repository includes the version numbers of all software components from the testbed environment.

Figure 18 depicts the experiment setup. S_1 and S_2 are BMv2 P4 switches, H_1 and H_2 are Linux hosts that are attached to them. S_1 and S_2 are connected via two unidirectional virtual links. We do not configure any additional delay or bandwidth limitations on these links. The traffic between H_1 and H_2 is forwarded by S_1 and S_2. We set up IPsec tunnels with different cipher suites and conduct TCP

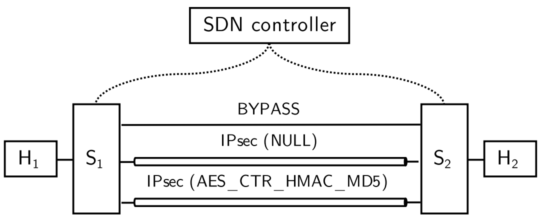


FIGURE 18. Experiment setup for evaluation of the P4-IPsec prototype on the BMv2 switches S_1 and S_2.

goodput measurements between H_1 and H_2 using iperf in version 3.7.

The results in the following represent measured average values with confidence intervals for a significance of $\alpha = 5\%$. Thus, the true averages lie within the displayed ranges with a probability of $1 - \alpha$.

B. DATA PLANE EVALUATION

We investigate how P4-IPsec's data plane implementation affects the overall throughput on the BMv2 P4 software target.

1) EXPERIMENT DESCRIPTION

We analyze TCP goodput for three different configurations. In the first scenario, we install BYPASS rules in the SPD so that traffic is only forwarded and not handled by IPsec. In the second scenario, we establish an IPsec tunnel between S_1 and S_2 with the NULL cipher suite. In the third scenario, we establish an IPsec tunnel between S_1 and S_2 with the AES_CTR_HMAC_MD5 cipher suite. Each experiment comprises 20 runs, each with a duration of 60 s and an MTU set to 1450 B. We configure soft and hard packet limits for rekeying to 50000 and 51000 resulting in an average of six rekeyings per run, three for each of the two SAs of the IPsec tunnel.

2) RESULTS & DISCUSSION

Figure 19(a) depicts the results. For forwarding without IPsec (BYPASS), TCP goodput is 48.90 Mbit/s. For IPsec forwarding with the NULL cipher suite, TCP goodput is 47.25 Mbit/s. For IPsec forwarding with the AES_CTR_HMAC_MD5 cipher suite, TCP goodput is 47.21 Mbit/s. Hence, the experimental results show similar TCP goodput rates between 47.21 Mbit/s and 48.90 Mbit/s for all three scenarios. The drop in performance is caused by IPsec processing, while the differences between both IPsec cipher suites are negligible. As all three results are still similar, we allocate the moderate overall TCP goodput to the runtime performance of the BMv2 P4 target. The low throughput of BMv2 is due to the fact that its use is intended for testing and not for production purposes. Thus, the results show that the overhead of our IPsec implementation on BMv2 only slightly reduces the TCP goodput and the impact of encryption/decryption operations is negligible on this platform.

C. CONTROL PLANE EVALUATION

We investigate how P4-IPsec's controller-based operation of IPsec affects the time needed for IPsec tunnel setup and renewal.

1) EXPERIMENT DESCRIPTION

In common IPsec deployment, SAs are set up between two IPsec peers using IKE message exchange. In P4-IPsec, an SDN controller sets up and renews IPsec tunnels, which may take longer due to controller operation and table updates on P4 switches.

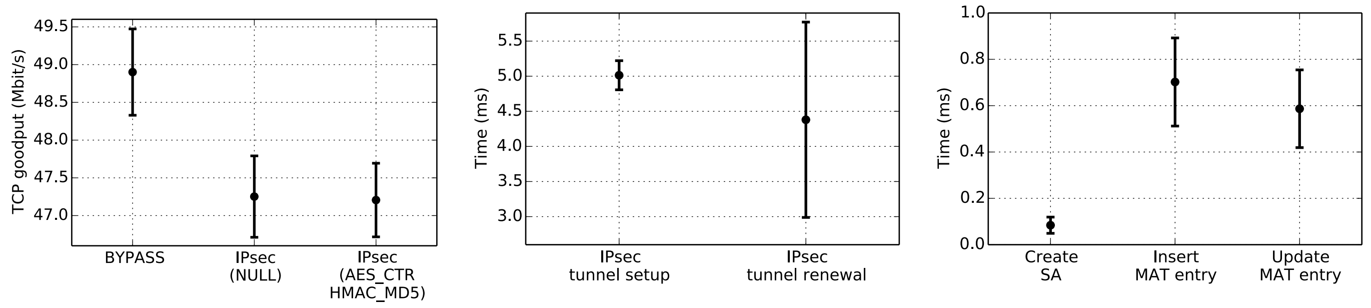
For *IPsec tunnel setup*, the SDN controller generates two unidirectional SAs, installs them on both P4 switches, and modifies the SPD MATs of both peers so that traffic is protected using IPsec. In our experiment, we measure the time for IPsec tunnel setup. It starts when the southbound connections between the SDN controller and the two P4 switches are established and ends with the last confirmation of the MAT modifications on the two P4 switches. For *IPsec tunnel renewal*, the SDN controller generates one unidirectional SA and installs it on both P4 switches. Tunnel renewal is triggered by a P4 switch if the packet counter of a SA reaches the soft packet limit. The measurement is started on the SDN controller when it receives the soft timeout notification from one P4 switch and it is stopped when the SDN controller has received all confirmations of the P4 switches about all MAT modifications. The details of both operations including sequence diagrams can be found in Section IV-D.4. We recorded measurement data for IPsec tunnel setup and tunnel renewal within the experiment on TCP goodput for the AES_CTR_HMAC_MD5 cipher suite as described before.

In the testbed environment, latency on the management link between the SDN controller and P4 switches is very low as all components run on the same host and as we have not configured any extra delay on the links. In real-world deployments, link latencies are significant, but they impact both IKE message exchange and controller-based operation of IPsec. By keeping the link latencies minimal, we derive an upper bound on potentially additional latency due to controller operation and MAT modification on P4 switches.

2) RESULTS & DISCUSSION

Figure 19(b) depicts the measured averages for IPsec tunnel setup and renewal times. Tunnel setup takes 5.02 ms while the time for tunnel renewal is 4.38 ms. These results show that the control plane overhead is low.

To further investigate both operations, we also analyze the durations of three major components: creating SA data, inserting new MAT entries, and updating existing MAT entries. Figure 19(c) depicts their average times. Creating keying material for a unidirectional SA takes 0.084 ms. Installing a new MAT entry via a write operation takes 0.702 ms. Updating an existing MAT entry takes 0.587 ms. Thus, the effort for key generation is almost negligible compared to MAT modifications.



(a) TCP goodput measured by iperf3 on the receiving host. Three variants are considered: without IPsec forwarding (BYPASS), with IPsec forwarding but without encryption, and with IPsec forwarding using the AES_CTR_HMAC_MD5 cipher suite.

(b) IPsec tunnel setup and renewal times. Times are measured on the SDN controller from initiation to completion of these processes.

(c) Times for creation of a single SA, insertion of a single MAT entry, and update of a single MAT entry. Times are measured on the SDN controller from initiation to completion of these processes.

FIGURE 19. Measurement results for experiments with P4-IPsec using BMv2 software switches in a virtual environment with almost zero link delays. Average values are shown with confidence intervals for a significance of $\alpha = 5\%$.

VII. IMPLEMENTATION ON HARDWARE P4 TARGETS

In the following, we describe implementation experiences for the NetFPGA SUME board and Edgecore Wedge 100BF-32X P4 switch.

A. NetFPGA SUME

We give an overview of the platform and describe implementation experiences.

1) OVERVIEW ON PLATFORM & DEVELOPMENT

NetFPGA SUME is an open source hardware development board for prototyping network applications. Its main part is a Xilinx Virtex-7 690T FPGA with 4 SFP+ ports that acts as programmable data plane. It supports throughput rates up to 100 GBit/s. The NetFPGA SUME board can be programmed via the Software Defined Specification Environment for Networking (SDNet) [78], a proprietary predecessor of P4 from Xilinx. Support for P4 programmability was introduced with the P4-NetFPGA tool [82]. First, a P4-to-SDNet compiler translates $P4_{16}$ programs into SDNet. Then, the SDNet compiler generates hardware description language (HDL) blocks in Verilog that can be validated in generic and platform-specific FPGA simulations. Finally, the HDL representation is synthesized into a hardware design to program the FPGA. In addition to the P4 program, custom functions can be implemented in a HDL and included in the hardware design. Programmers may implement custom HDL blocks or integrate semiconductor intellectual property cores that can be used as P4 externs in the P4 program. The P4-NetFPGA tool only supports the SimpleSumeSwitch architecture, i.e., P4 programs defined for more sophisticated architectures such as Portable Switch Architecture (PSA) need to be transformed to this architecture.

2) IMPLEMENTATION EXPERIENCES

We report on implementation experiences about porting our software-based implementation P4-IPsec for the NetFPGA SUME board. First, P4-NetFPGA is currently limited to *packet header manipulation*. P4-IPsec requires modifications of packet payloads, i.e., we were required to parse packet payloads as an additional header field. As P4-NetFPGA does not support parsing variable-length header fields, the implementation is limited to packets with a fixed length. Second, P4-NetFPGA lacks a packet streaming function for *data exchange between the P4 pipeline and P4 externs*. Instead, data between the P4 pipeline and externs is currently exchanged via blocks of bits. As this data transfer needs to be executed within one clock cycle of the FPGA, the data size is limited. We observed that this limit is about 10 kbit for one function call. This limits the maximum packet size to be processed through a P4 extern to about 600 B. During the synthesis, the Vivado suite optimizes the hardware implementation through several algorithms. In various experiments, we observed a practical upper bound of about 140 B for packets. Either the hardware implementation did use more resources than offered by the FPGA, or data transfer and calculation within the P4 extern exceeded one clock cycle. A packet streaming function was announced in 2018, but is still not available. Last, we encountered several *more general problems* with P4-SDNet and the NetFPGA SUME board. Probably due to a bug, we were not able to access the values of an LPM table for IP routing with our SDN controller. We solved that problem by using exact matching tables instead, an approach that is not acceptable for a production implementation. In addition, we experienced several stability problems. No matches in MATs were found when data was written to hardware registers. Finally, we missed many important details in the documentation. With hope for improved support, we managed to implement a very limited prototype. It only allows to apply the NULL cipher on fixed-length packets that do not exceed a total length of 140 B.

Scholz *et al.* [83] report on implementation experiences of cryptographic hashing functions in P4 data planes. The NetFPGA SUME board is also one of the platforms examined where the authors present results that correspond to our results. As a workaround, the authors propose to move the externs subsequent to the synthesized P4 program. However, the workaround can be applied only if the P4 program does not rely on the output of the extern. This makes it inapplicable to P4-IPsec. Besides, implementing this workaround requires extensive knowledge about HDLs and FPGA programming.

B. EDGECORE WEDGE WITH TOFINO

We give an overview of the platform and describe why a direct adoption of P4-IPsec is not feasible. We present two workaround implementations and evaluate their performance in experiments.

1) OVERVIEW ON PLATFORM & DEVELOPMENT

The Edgecore Wedge 100BF-32X [84], features 32 QSFP28 network ports with throughput rates up to 100 Gbit/s. The QSFP28 ports interface the Tofino switching ASIC from Barefoot Networks which is fully programmable with P4. The Tofino ASIC connects to a CPU module via PCIe. It features an Intel Pentium D1517 processor (1.6 GHz, 4 cores), 8 GB RAM, and a 32 GB SSD. For programming and managing the Tofino ASIC, the CPU module runs the Barefoot P4 Software Development Environment on top of a Linux-based operating system. It loads and manages P4 programs during execution, provides management APIs (e.g., P4Runtime), and exposes an interface for network packet exchange between the P4 processing pipeline and the CPU module. Due to its optimization for high-speed packet processing with bandwidths up to multiple Tbit/s in data centers or core networks, user-defined P4 externs that may contain computation-intense functions are not supported.

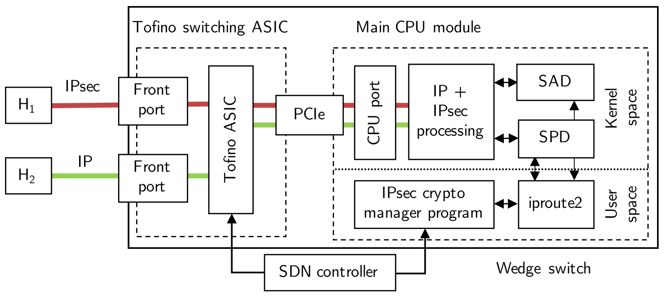


FIGURE 20. First workaround implementation. We relocate IPsec processing to the main CPU module that interfaces the Tofino switching ASIC via a PCIe CPU port.

2) WORKAROUND IMPLEMENTATIONS

In our first workaround implementation, we *relocate the P4 externs of P4-IPsec to the main CPU module*. Figure 20 depicts the concept. We replace all P4 extern function calls in the P4 processing pipeline by packet transfers via the CPU port to the main CPU module. On the main CPU module, we use the IPsec kernel functions of the Linux operating system for IPsec processing. We implement a simple IPsec crypto manager program that translates P4-IPsec configuration from the SDN controller into IPsec configuration for the Linux host. We implement the IPsec crypto manager in Python 3. It relies on iproute2 commands to manage the SPD and SAD of the Linux host.

We briefly evaluate this first workaround implementation with experiments on latency and TCP goodput. As depicted in Figure 20, we attach two physical hosts running Ubuntu 16.04 LTS via 100 Gbit/s links to the front ports of the Wedge switch. We enable IPsec on the link between H_1 and the switch while the link between the switch and H_2 remains unprotected. First, we measure the latency that is introduced by IPsec processing and packet exchange with the CPU module. We send 100 ICMP echo requests from H_1 to H_2 and measure an average round-trip time of about 1.5 ms. Second, we investigate the maximum TCP goodput. We generate TCP transmissions with iperf3 in three experiments, each performed with five runs and a duration of 30 s. For getting a reference, we measure the maximum TCP goodput between the P4 processing pipeline and the main CPU module. Therefore, we assign an IP address to the virtual network interface of the CPU port on the main CPU module and run iperf3 measurements between H_1 and the main CPU module. We measure an average TCP goodput of about 3.3 Gbit/s. We consider this as upper bound for the main CPU module. Now, we measure TCP goodput between H_1 and H_2. When using the NULL cipher suite, we measure an average TCP goodput of about 2 Gbit/s. For the AES-GCM-256 cipher suite, the average TCP goodput drops to about 1.4 Gbit/s. We repeat the experiment for 16 concurrent IPsec tunnels and calculate the average of 10 runs with a duration of 300 s. The maximum TCP goodput remains at 2 Gbit/s for IPsec with the NULL cipher suite and 1.4 Gbit/s for IPsec with the AES-GCM-256 cipher suite. We attribute the large differences in TCP goodput to the rather slow CPU with a base frequency of 1.6 GHz. Still, we consider this a very reasonable performance that might be sufficient for scenarios where only few shared network resources should be accessed sporadically by roadwarrior hosts.

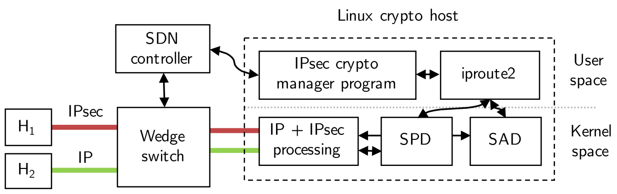


FIGURE 21. Second workaround implementation. We forward IPsec-related flows to a Linux crypto host.

In our second workaround implementation, we *forward IPsec-related flows to a crypto host*. This approach is used by several past works on integrating IPsec with fixed function SDN data planes (e.g., [45]). As depicted in Figure 21, we set up a Linux crypto host for offloading IPsec processing. We deploy the IPsec crypto manager program from the previous workaround implementation as an interface between the crypto host and SDN controller. We deploy a simple P4 program on the Wedge switch that forwards IPsec flows based

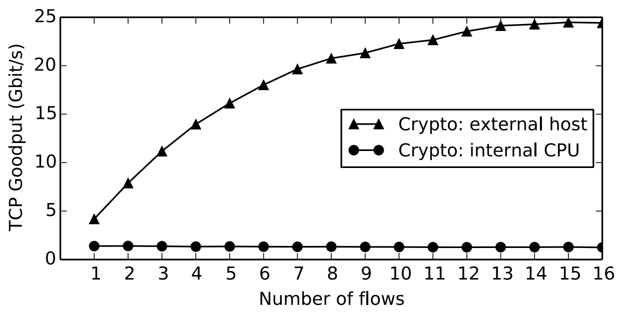


FIGURE 22. **Average TCP goodput for both workaround implementations and 1-16 IPsec tunnels with the AES-GCM-256 cipher suite.**

on a MAT. The SDN controller writes/edits the forwarding MAT on the Wedge switch and sends configuration messages to the IPsec crypto manager program.

For a simple evaluation, we set up a crypto host with an Intel Xeon Gold 6134 CPU (8 cores, 16 threads), 128 GB RAM, and a 240 GB SSD, running Ubuntu 18.04 LTS. We perform the same experiments as for the first workaround implementation. The round-trip time is about 2 ms which is slightly larger than in the previous approach. Figure 22 compares TCP goodput results for IPsec tunnels with the AES-GCM-256 cipher suite of both workaround implementations. For a single IPsec tunnel with the AES-GCM-256 cipher suite, we measure an average TCP goodput of about 4 Gbit/s. It can be increased by running multiple connections over the same crypto host. For 16 parallel IPsec tunnels, we measure an overall average TCP goodput of about 24 Gbit/s. This effect can be attributed to receive-side scaling (RSS) of the network interface card, which can leverage multiple cores, but only one per IPsec tunnel. In case of multiple IPsec tunnels, the overall TCP goodput can be increased through RSS by leveraging the processing power of more than a single core. Crypto capacity can be scaled up by increasing the number of crypto hosts connected to the switch. TCP goodput on each crypto host can be further improved by optimization techniques as presented in Section III-C.

Chen [85] presents an implementation of AES encryption for Tofino-based P4 switches. It uses a novel Scrambled Lookup Table technique that allows throughput rates of up to 10.92 Gbit/s for AES-128. However, the current concept is limited to blockwise encryption of packets with a maximum payload size of 16 bytes so that it is in its current form not a suitable base for IPsec support. If subsequent versions of this work introduce block chaining, integrating P4-IPsec could be an interesting follow-up work.

VIII. CONCLUSION

In this work, we proposed the first implementation of IPsec in P4. The proposed data plane implementation features ESP in tunnel mode and provides support for multiple cipher suites with the help of P4 externs. P4-IPsec supports automated operation of IPsec in host-to-site and site-to-site scenarios. IPsec tunnels are set up and managed by an SDN controller based on predefined tunnel profiles. For interaction with remote hosts in host-to-site scenario, we introduce a client agent for Linux hosts. We introduced the fundamentals of IPsec and data plane programming with P4, gave an extensive review on related work, and presented the architecture of P4-IPsec.

P4 programmable data planes open up the possibility of implementing IPsec on SDN-capable data plans for the first time. However, the implementation on P4 switches is still challenging. For the BMv2 software switch, the implementation was straightforward, but moderate data rates make its practical application difficult. However, the controller-supported signaling was not a bottleneck. Due to the platform limitations of the NetFPGA SUME board, we were not able to build a working prototype. With the Tofino-based Wedge switch, we were successful. Even though it does not support P4 externs, we presented two workaround implementations that leverage the main CPU module for crypto functions or an external crypto host, respectively.

We have shown that security use cases can benefit from P4, but crypto functions are still missing on P4 hardware switches. Therefore, we advocate for P4 hardware targets that either include P4 externs for those operations or offer powerful interfaces so that developers can run individual functions on the CPU module of such switches. Such features have the potential to massively foster the deployment of P4 targets in practice and stimulate further network research.

LIST OF ACRONYMS

SDN	software-defined networking
ONF	Open Network Foundation
OF	OpenFlow
ODP	Open Data Plane
NFV	network function virtualization
VNF	virtual network function
SFC	service function chaining
BMv2	Behavioral Model version 2
MAT	match-action table
VPN	Virtual Private Network
IP	Internet Protocol
IPsec	Internet Protocol Security
ESP	Encrypted Secured Payload
AH	Authentication Header
PPF	packet processing function
SP	security policy
SPD	Security Policy Database
SA	security association
SAD	Security Association Database
PAD	Peer Authentication Database
SPI	Security Parameter Index
IKE	Internet Key Exchange
IKEv2	Internet Key Exchange v2
IPComp	IP Payload Compression
AE	authenticated encryption
ICV	Integrity Check Value
IV	initialization vector

3DES	Triple Data Encryption Standard
AES	Advanced Encryption Standard
CBC	cipher block chaining
CTR	counter
GCM	galois/counter mode
HMAC	keyed-hash message authentication code
SHA	secure hash algorithm
TLS	Transport Layer Security
PKI	public key infrastructure
SoC	system on a chip
FPGA	field programmable gate array
NPU	network processing units
ASIC	application-specific integrated circuit
NIC	network interface card
NPU	network processing unit
APU	accellerated processing unit
DPDK	Data Plane Development Kit
SDNet	Software Defined Specification Environment for Networking
HDL	hardware description language
HLIR	high-level intermediate representation
RTL	Register Transfer Level
PSA	Portable Switch Architecture
TTL	time to live
INT	in-band network telemetry
LPM	longest-prefix matching
CLI	command line interface
FSM	finite state machine
API	application programming interface
LLDP	Link Layer Discovery Protocol
MACsec	Media Access Control Security
RSS	receive-side scaling

ACKNOWLEDGMENT
The authors would like to thank the anonymous reviewers for their valuable comments. The authors alone are responsible for the content of this paper.

REFERENCES

[1] *P4-IPsec Repository on GitHub*. Accessed: Jul. 26, 2020. [Online]. Available: https://github.com/uni-tue-kn/p4-ipsec
[2] F. Hauser, M. Schmidt, M. Haberle, and M. Menth, "P4-MACsec: Dynamic topology monitoring and data layer protection with MACsec in P4-based SDN," *IEEE Access*, vol. 8, pp. 58845–58858, 2020.
[3] K. Seo and S. Kent, *Security Architecture for the Internet Protocol*, document RFC 4301, Dec. 2005.
[4] S. Kent, *IP Authentication Header*, document RFC 4302, Dec. 2005.
[5] S. Kent, *IP Encapsulating Security Payload (ESP)*, document RFC 4303, Dec. 2005.
[6] J. Viega and D. McGrew, *The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP)*, document RFC 4106, Jun. 2005.
[7] A. Shacham, B. Monsour, R. Pereira, and M. Thomas, *IP Payload Compression Protocol (IPComp)*, document RFC 3173, Sep. 2001.
[8] D. Carrel and D. Harkins, *The Internet Key Exchange (IKE)*, document RFC 2409, Nov. 1998.
[9] C. Kaufman, E. Paul Hoffman, Y. Nir, P. Eronen, and T. Kivinen, *Internet Key Exchange Protocol Version 2 (IKEv2)*, document RFC 7296, Oct. 2014.
[10] N. Ferguson and B. Schneier, *A Cryptographic Evaluation of IPsec*. Santa Clara, CA, USA: Counterpane Internet Security, 2000.
[11] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, p. 69, 2008.
[12] R. Bifulco and G. Retvari, "A survey on the programmable data plane: Abstractions, architectures, and open problems," in *Proc. IEEE 19th Int. Conf. High Perform. Switching Routing (HPSR)*, Jun. 2018, pp. 1–7.
[13] *P4_16 Language Specification, Version 1.2.1*. Accessed: Jul. 26, 2020. [Online]. Available: https://p4.org/p4-spec/docs/P4-16-v1.2.1.html
[14] *BMv2*. Accessed: Jul. 26, 2020. [Online]. Available: https://github.com/p4lang/behavioral-model
[15] *T4P4S*. Accessed: Jul. 26, 2020. [Online]. Available: http://p4.elte.hu/
[16] *DPDK*. Accessed: Jul. 26, 2020. [Online]. Available: https://www.dpdk.org/
[17] *ODP*. Accessed: Jul. 26, 2020. [Online]. Available: https://opendataplane.org/
[18] *P4 Runtime Specification, Version 1.2.0*. Accessed: Jul. 26, 2020. [Online]. Available: https://p4.org/p4runtime/spec/v1.2.0/P4Runtime-Spec.html
[19] *Google Protocol Buffers*. Accessed: Jul. 26, 2020. [Online]. Available: https://developers.google.com/protocol-buffers/
[20] *P4 Language Tutorials*. Accessed: Jul. 26, 2020. [Online]. Available: https://githubcomlangtutorialstreemasterutils runtimelib
[21] A. Kiss, "Security middleware programming using P4," in *Security Middleware Programming Using*, T. Tryfonas, Ed. Cham, Switzerland: Springer, 2016, pp. 277–287.
[22] R. Datta, S. Choi, A. Chowdhary, and Y. Park, "P4Guard: Designing p4 based firewall," in *Proc. MILCOM - IEEE Mil. Commun. Conf. (MILCOM)*, Oct. 2018, pp. 1–6.
[23] R. Ricart-Sanchez, P. Malagon, J. M. Alcaraz-Calero, and Q. Wang, "Hardware-accelerated firewall for 5G mobile networks," in *Proc. IEEE 26th Int. Conf. Netw. Protocols (ICNP)*, Sep. 2018, pp. 446–447.
[24] J. Cao, J. Bi, Y. Zhou, and C. Zhang, "CoFilter: A high-performance switch-assisted stateful packet filter," in *Proc. SIGCOMM*, 2018, pp. 9–11.
[25] E. O. Zaballa, D. Franco, Z. Zhou, and S. Michael Berger, "P4Knocking: Offloading host-based firewall functionalities to the network," in *Proc. Conf. Innov. Clouds, Internet Netw. Workshops (ICIN)*, 2020, pp. 7–12.
[26] A. Almaini, A. Al-Dubai, I. Romdhani, and M. Schramm, "Delegation of authentication to the data plane in software-defined networks," in *Proc. IEEE Int. Conferences Ubiquitous Comput. Commun. (IUCC) Data Sci. Comput. Intell. (DSCI) Smart Comput., Netw. Services (SmartCNS)*, Oct. 2019, pp. 58–65.
[27] F. Paolucci, F. Cugini, and P. Castoldi, "P4-based multi-layer traffic engineering encompassing cyber security," in *Proc. Opt. Fiber Commun. Conf.*, 2018, pp. 1–3.
[28] F. Paolucci, F. Civerchia, A. Sgambelluri, A. Giorgetti, F. Cugini, and P. Castoldi, "P4 Edge node enabling stateful traffic engineering and cyber security," *IEEE/OSA J. Opt. Commun. Netw.*, vol. 11, no. 1, pp. 84–95, Oct. 2019.
[29] M. Dimolianis, A. Pavlidis, and V. Maglaris, "A multi-feature DDoS detection schema on p4 network hardware," in *Proc. 23rd Conf. Innov. Clouds, Internet Netw. Workshops (ICIN)*, Feb. 2020, pp. 1–6.
[30] A. Febro, H. Xiao, and J. Spring, "Telephony denial of service defense at data plane (TDoSDDP)," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp. (NOMS)*, 2018, pp. 1–6.
[31] A. Febro, H. Xiao, and J. Spring, "Distributed SIP DDoS defense with p4," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2019, pp. 1–8.
[32] G. Grigoryan and Y. Liu, "LAMP: Prompt layer 7 attack mitigation with programmable data planes," in *Proc. IEEE 17th Int. Symp. Netw. Comput. Appl. (NCA)*, Nov. 2018, pp. 1–4.
[33] Y. Afek, A. Bremler-Barr, and L. Shafir, "Network anti-spoofing with SDN data plane," in *Proc. IEEE Conf. Comput. Commun.*, May 2017, pp. 1–9.
[34] C. Lapolli. Ã., J. A. A. Marques, and L. P. Gaspary, "Offloading Real-time DDoS Attack Detection to Programmable Data Planes," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manage. (IM)*, Oct. 2019, pp. 19–27.
[35] M. Kuka, K. Vojanec, J. Kucera, and P. Benacek, "Accelerated DDoS attacks mitigation using programmable data plane," in *Proc. ACM/IEEE Symp. Architectures Netw. Commun. Syst. (ANCS)*, Sep. 2019, pp. 1–3.
[36] Y. Mi and A. Wang, "ML-pushback: Machine learning based pushback defense against DDoS," in *Proc. 15th Int. Conf. Emerg. Netw. EXperiments Technol.*, Dec. 2019, p. 80.

[37] B. Lewis, M. Broadbent, and N. Race, "P4ID: P4 enhanced intrusion detection," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2019, pp. 1–4.

[38] Q. Kang, L. Xue, A. Morrison, Y. Tang, A. Chen, and X. Luo, "Programmable in-network security for context-aware BYOD policies," in *Proc. Secur. Symp.*, 2020, pp. 1–7.

[39] S. Aragon, M. Tiloca, M. Maass, M. Hollick, and S. Raza, "ACE of spades in the IoT security game: A flexible IPsec security profile for access control," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, May 2018, pp. 1–9.

[40] S. Aragon, M. Tiloca, and S. Raza, *IPsec profile of ACE*. Fremont, CA, USA: Internet Engineering Task Force, 2017,

[41] D. Carrel and B. Weis, *IPsec Key Exchange using a Controller*. Fremont, CA, USA: Internet Engineering Task Force, 2019.

[42] X. Guo, K. Yang, A. Galis, X. Cheng, B. Yang, and D. Liu, "A policy-based network management system for IP VPN," in *Proc. Int. Conf. Commun. Technol. Process.*, 2003, pp. 1630–1633.

[43] G. Lopez-Millan, R. Marin-Lopez, and F. Pereniguez-Garcia, "Towards a standard SDN-based IPsec management framework," *Comput. Standards Interface*, vol. 66, Oct. 2019, Art. no. 103357.

[44] J. Son, Y. Xiong, K. Tan, P. Wang, Z. Gan, and S. Moon, "Protego: Cloud-scale multitenant IPsec gateway," in *Proc. USENIX Annu. Tech. Conf.*, 2017, pp. 473–485.

[45] M. Vajaranta, J. Kannisto, and J. Harju, "IPsec and IKE as functions in SDN controlled network," in *Proc. Int. Conf. Netw. Syst. Secur. (NSS)*, 2017, pp. 521–530.

[46] R. Lopez, G. Lopez-Millan, and F. Pereniguez-Garcia, *Software-Defined Networking (SDN)-based IPsec Flow Protection*. Fremont, CA, USA: Internet Engineering Task Force, 2019.

[47] A. S. Braadland, "Key Management for data plane encryption in SDN using wireGuard," M.S. thesis, Dept. Inf. Secur. Commun. Technol., Norwegian Univ. Sci. Technol., Norway, China, 2017.

[48] A. Sajassi, A. Banerjee, S. Thoria, D. Carrel, B. Weis, and J. Drake, *Secure EVPN*. Fremont, CA, USA: Internet Engineering Task Force, 2019.

[49] H. Gunleifsen, V. Gkioulos, and T. Kemmerich, "A tiered control plane model for service function chaining isolation," *Future Internet*, vol. 10, no. 6, p. 46, Jun. 2018.

[50] H. Gunleifsen, T. Kemmerich, and V. Gkioulos, "A Proof-of-Concept demonstration of isolated and encrypted service function chains," *Future Internet*, vol. 11, no. 9, p. 183, Aug. 2019.

[51] H. Gunleifsen, T. Kemmerich, and V. Gkioulos, "Dynamic setup of IPsec VPNs in service function chaining," *Comput. Netw.*, vol. 160, pp. 77–91, Sep. 2019.

[52] *Cisco: IPsec Management Configuration Guide*. Accessed: Jul. 26, 2020. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/sec_conn_imgmt/config%uration/xe-16/sec-ipsec-management-xe-16-book/sec-ipsec-snmp-supp.html

[53] A. Alharbi, A. Bahnasse, M. Talea, H. A. Oulahyane, and F. E. Louhab, *Smart SDN Policy Management Based VPN Multipoint*. Cham, Switzerland: Springer, 2019, pp. 250–263.

[54] Y. Li and J. Mao, "SDN-based access authentication and automatic configuration for IPSec," in *Proc. 4th Int. Conf. Comput. Sci. Netw. Technol. (ICCSNT)*, Dec. 2015, pp. 996–999.

[55] W. Li, F. Lin, and G. Sun, "SDIG: Toward software-defined IPsec gateway," in *Proc. IEEE 24th Int. Conf. Netw. Protocols (ICNP)*, Nov. 2016, pp. 1–8.

[56] M. Mechtri, I. Houidi, W. Louati, and D. Zeghlache, "SDN for inter cloud networking," in *Proc. IEEE SDN for Future Netw. Services (SDN4FNS)*, Nov. 2013, pp. 1–7.

[57] R. van der Pol, B. Gijsen, P. Zuraniewski, D. F. C. Romáo, and M. Kaat, "Assessment of SDN technology for an easy-to-use VPN service," *Future Gener. Comput. Syst.*, vol. 56, pp. 295–302, Mar. 2016.

[58] L. Rizzo, "Netmap: A Novel Framework for Fast Packet I/O," in *Proc. Annu. Tech. Conf.*, 2012, pp. 101–112.

[59] *N. Pf_ring*. Accessed: Jul. 26, 2020. [Online]. Available: https://www.ntop.org/products/packet-capture/pf_ring/

[60] W. Li, S. Hu, G. Sun, and Y. Li, "Adaptive load balancing on multi-core IPsec gateway," in *Proc. Int. Conf. Algorithms Archit. Parallel Process. (ICAPP)*, 2018, pp. 1–7.

[61] G. Xie, H. Jiang, and K. Salamatian, "Load balancing by ruleset partition for parallel IDS on multi-core processors," in *Proc. Int. Conf. Comput., Commun. Netw. (ICCCN)*, 2013, pp. 1–7.

[62] S. Han, K. Jang, K. Park, and S. Moon, "PacketShader: A GPU-accelerated software router," in *Proc. ACM SIGCOMM Conf. SIGCOMM*, 2010, pp. 195–206.

[63] S. Gallenmuller, P. Emmerich, F. Wohlfart, D. Raumer, and G. Carle, "Comparison of frameworks for high-performance packet IO," in *Proc. ACM/IEEE Symp. Architectures for Netw. Commun. Syst. (ANCS)*, May 2015, pp. 29–38.

[64] *Intel AES-NI*. Accessed: Jul. 26, 2020. [Online]. Available: https://www.intel.de/content/www/de/de/architecture-and-technology/adva%nced-encryption-standard-aes/data-protection-aes-general-technology.html

[65] *Architecture Reference Manual*, Arm Ltd, Cambridge, U.K., 2017.

[66] J. DiGiglio and D. Ricci, *High Performance, Open Standard, Virtualization with NFV and SDN*. Alameda, CA, USA: Wind River, 2013.

[67] *Algotronix AES IP-Cores*. Accessed: Jul. 26, 2020. [Online]. Available: http://www.algotronix-store.com/AES_IP_Cores_s/20.htm

[68] C.-S. Ha, J. Hyoung Lee, D. Soo Leem, M.-S. Park, and B.-Y. Choi, "ASIC design of IPSec hardware accelerator for network security," in *Proc. IEEE Asia–Pacific Conf. Adv. Syst. Integr. Circuits*, 2004, pp. 168–171.

[69] A. Hodjat, P. Schaumont, and I. Verbauwhede, "Architectural design features of a programmable high throughput AES coprocessor," in *Proc. Int. Conf. Inf. Technol.*, 2004, pp. 1–7.

[70] Y. Liu, D. Xu, W. Song, and Z. Mu, "Design and implementation of high performance IPSec applications with multi-core processors," in *Proc. Int. Seminar Future Inf. Technol. Manage. Eng.*, Nov. 2008, pp. 595–598.

[71] J. Meng, X. Chen, Z. Chen, C. Lin, B. Mu, and L. Ruan, "Towards High-Performance IPsec on Cavium OCTEON Platform," in *Int. Conf. Trusted Syst. (INTRUST)*, 2011, pp. 37–46.

[72] J. Park, W. Jung, G. Jo, I. Lee, and J. Lee, "PIPSEA: A practical IPsec gateway on embedded APUs," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 1255–1267.

[73] M. Rao, J. Coleman, and T. Newe, "An FPGA based reconfigurable IPSec ESP core suitable for IoT applications," in *Proc. 10th Int. Conf. Sens. Technol. (ICST)*, Nov. 2016, pp. 1–5.

[74] M. Vajaranta, V. Viitamaki, A. Oinonen, T. D. Hamalainen, A. Kulmala, and J. Markunmaki, "Feasibility of FPGA accelerated IPsec on cloud," in *Proc. 21st Euromicro Conf. Digit. Syst. Design (DSD)*, Aug. 2018, pp. 569–572.

[75] M. Korona, K. Skowron, M. Trzepinski, and M. Rawski, "FPGA implementation of IPsec protocol suite for multigigabit networks," in *Proc. Int. Conf. Syst., Signals Image Process. (IWSSIP)*, May 2017, pp. 1–5.

[76] B. Driessen, E. B. Kavun, O. Mischke, and C. Paar, "IPSecco: A lightweight and reconfigurable IPSec core," in *Proc. Int. Conf. Reconfigurable Comput. FPGAs (ReConFig)*, 2012, pp. 1–7.

[77] *[P4-dev] VPNs*. Accessed: Jul. 26, 2020. [Online]. Available: https://lists.org.pipermail.lists.org.html

[78] *Xilinx SDNet PX Programming Language User Guide*. Accessed: Jul. 26, 2020. [Online]. Available: https://www.xilinx.com.support.documentation.sw.manuals.xilinx.ug

[79] *Vagrant*. Accessed: Jul. 26, 2020. [Online]. Available: https://www.vagrantup.com/

[80] *GRPC*. Accessed: Jul. 26, 2020. [Online]. Available: https://grpc.io/

[81] *Netlink*. Accessed: Jul. 26, 2020. [Online]. Available: http://man7.org/linux/man-pages/man7/netlink.7.html

[82] *P4-NetFPGA Wiki*. Accessed: Jul. 26, 2020. [Online]. Available: https://github.com/NetFPGA/P4-NetFPGA-public/

[83] D. Scholz, A. Oeldemann, F. Geyer, H. Stubbe, T. Wild, A. Herkersdorf, and G. Carle, "Cryptographic hashing in P4 data planes," in *ACM/IEEE Symp. Archit. Netw. Commun. Syst. (ANCS)*, Sep. 2019, pp. 1–6.

[84] *EdgeCore Wedge 100BF-32X*. Accessed: Jul. 26, 2020. [Online]. Available: https://www.edge-core.com/productsInfo.php?cls=1&cls2=180&cls3=181&id=3%35

[85] X. Chen, "Implementing AES encryption on programmable switches via scrambled lookup tables," in *Proc. Workshop Secure Program. Netw. Infrastruct.*, Aug. 2020, pp. 1–7.

FREDERIK HAUSER (Graduate Student Member, IEEE) received the master's degree in computer science from the University of Tübingen, Germany, where he is currently pursuing the Ph.D. degree with the Chair of Communication Networks. He has been a Researcher at the Chair of Communication Networks, University of Tübingen. His main research interests include software-defined networking, network function virtualization, and network security.

MARCO HÄBERLE (Student Member, IEEE) received the master's degree in computer science from the University of Tübingen, Germany, where he is currently pursuing the Ph.D. degree with the Chair of Communication Networks. He has been a Researcher at the Chair of Communication Networks, University of Tübingen. His main research interests include software-defined networking, P4, network security, and automated network management.

MARK SCHMIDT (Member, IEEE) received the Diploma degree in computer science from the University of Tübingen, Germany, where he is currently pursuing the Ph.D. degree with the Chair of Communication Networks. He has been a Researcher at the Chair of Communication Networks, University of Tübingen. His main research interests include software-defined networking, OpenFlow, high-speed networks, and virtualization.

MICHAEL MENTH (Senior Member, IEEE) is a Professor at the Department of Computer Science, University of Tübingen, and the Chair Holder of Communication Networks. His special interests include performance analysis and optimization of communication networks, resilience and routing issues, resource and congestion management, software-defined networking and Internet protocols, industrial networking, and the Internet of Things.

1.2 P4-MACsec: Dynamic Topology Monitoring and Data Layer Protection With MACsec in P4-Based SDN

Received February 7, 2020, accepted February 25, 2020, date of publication March 23, 2020, date of current version April 7, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2982859

P4-MACsec: Dynamic Topology Monitoring and Data Layer Protection With MACsec in P4-Based SDN

FREDERIK HAUSER, (Student Member, IEEE), MARK SCHMIDT, (Member, IEEE), MARCO HÄBERLE, (Student Member, IEEE), AND MICHAEL MENTH, (Senior Member, IEEE)
Chair of Communication Networks, University of Tuebingen, 72076 Tübingen, Germany

Corresponding author: Frederik Hauser (frederik.hauser@uni-tuebingen.de)

This work was supported by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/1-2 and the bwNET100G+ project which is funded by the Ministry of Science, Research and the Arts Baden-Württemberg (MWK). The authors alone are responsible for the content of this paper.

ABSTRACT We propose P4-MACsec to protect network links between P4-based SDN switches through automated deployment of MACsec, a widespread IEEE standard for securing Layer 2 infrastructures. MACsec is supported by switches and routers from many manufacturers. On these devices, it has only little performance limitations compared to VPN technologies such as IPsec. P4-MACsec suggests a data plane implementation of MACsec including AES-GCM encryption and decryption directly on P4 targets. P4-MACsec features a two-tier control plane structure where local controllers running on the P4 targets interact with a central controller. We propose a novel secure link discovery mechanism that leverages protected LLDP frames and a two-tier control plane structure for secure and efficient management of a global link map. Automated deployment of MACsec creates secure channels, generates keying material, and configures the P4 targets for each detected link between two P4 targets. It detects link changes and performs rekeying to provide a secure, configuration-free operation of MACsec. In this paper, we review the technological background of P4-MACsec and explain its architecture. To demonstrate the feasibility of P4-MACsec, we implement it on the BMv2 P4 software target, validate the prototype through experiments, and evaluate its performance through experiments considering TCP goodput and round-trip time. We publish the prototype and experiment setup under the Apache v2 license on GitHub [7].

INDEX TERMS MACsec, P4, software-defined networking, VPN.

I. INTRODUCTION

MACsec [41] is a widespread IEEE standard that protects the Layer 2 with cryptographic integrity checks or symmetric encryption. MACsec prevents man-in-the-middle attackers from inspecting, inserting, or even modifying network packets that are transmitted between two network peers. In contrast to VPN technologies such as IPsec, MACsec processing is implemented on forwarding chips of many devices without notable overhead in line rate performance [30]. Packets are protected in a point-to-point manner between MACsec peers so that control plane functions targeting higher layers, e.g., access-control lists (ACLs), can be still applied. Although mechanisms for distributed key exchange exist, MACsec deployment still requires time-consuming and complex initial setup procedures on all devices. It requires knowledge about the network topology, large efforts in switch configuration, and typically maintenance of a key server. Currently, automated deployment using a network management system with legacy switches is not feasible. Legacy network switches only support the Link Layer Discovery Protocol (LLDP) [45] that lacks timely detection of topology changes. In addition, it is vulnerable to several attacks that may result in an incorrect view of the topology. Moreover, current legacy network switches do not support an automated configuration of MACsec through a southbound protocol. Although a MIB for manipulating MACsec configuration with SNMP exists [10], only basic MACsec parameters can be modified. Additional per-switch configuration and a key exchange server are still required.

Software-Defined Networking (SDN) splits the strong binding between data and control plane. OpenFlow (OF) [48]

The associate editor coordinating the review of this manuscript and approving it for publication was Chi-Tsun Cheng.

is the most widespread standard architecture and southbound protocol for SDN. It consists of SDN switches with a fixed-function data plane that are steered by a central SDN controller. To resolve inflexibilities of fixed-function data planes, P4 [37] emerged as novel domain-specific language that introduces programmability to the data plane of P4-capable packet forwarding devices such as ASICs, CPU-based targets, or field programmable gate arrays (FPGAs). Data plane behavior can be described in P4 programs that run on P4 targets so that network operators can continuously program the behavior of deployed packet processing devices. The P4Runtime [23] extends P4 targets by an API to an SDN controller. It serves roughly the same purpose as the OF southbound protocol.

In this paper, we consider MACsec to dynamically protect links between switches in SDN. We propose to use an SDN controller to continuously monitor the network topology and set up MACsec on all detected links between switches. As OF does not provide support for integrating the required MACsec functions on the data plane, we propose a concept based on P4 and call it P4-MACsec. We implement a P4 data plane with packet switching based on MAC addresses and MACsec, i.e., MACsec encryption, decryption, and integrity checks of packets. In P4-MACsec, P4 targets implement typical switching functionality. Therefore, we call them P4 switches instead of P4 targets in the following. P4 switches are managed by a two-tier control plane where each P4 switch runs a local controller that connects to a central controller. Functions of the control plane may be solely part of the local controller or part of both tiers. The control plane implements MAC address learning for packet switching, a novel mechanism for secure link discovery with encrypted LLDP packets, and automated deployment of MACsec. To demonstrate the feasibility of P4-MACsec, we provide a prototype based on the Behavioral Model version 2 (BMv2) P4 software target [6]. We perform a functional validation of P4-MACsec in a Mininet testbed through experiments and investigate on TCP goodput and round-trip time (RTT) by conducting a performance evaluation. We publish the source code of the prototype and testbed setup instructions under the Apache v2 license on GitHub [7]. In addition, we report on implementation experiences for the NetFPGA SUME [14] platform.

The rest of the paper is structured as follows. In Section II, we review technical background and related work for MACsec. Section III discusses technical background and related work on link discovery in SDN. In Section IV, we give an overview on the P4 programming model and language. Section V describes the architecture of P4-MACsec. In Section VI, we describe the prototypical implementation of P4-MACsec with Mininet that is validated in Section VII. In Section VIII, we present a performance evaluation of the Mininet prototype. In Section IX, we report on experiences in implementing P4-MACsec on the NetFPGA SUME platform. Section X concludes this work. The appendices include a list of acronyms that are used in the paper.

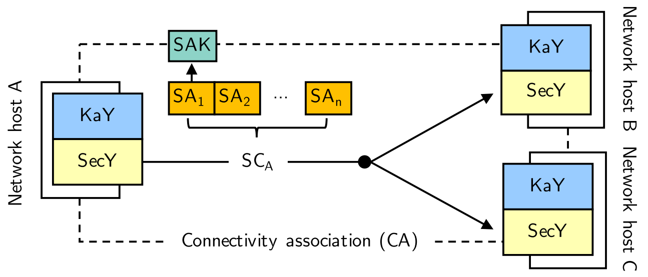


FIGURE 1. Secure communication between three stations in MACsec that are part of a CA. The unidirectional SC between the SecYs holds multiple SAs each with an SAK for encryption and decryption.

II. MACsec: FOUNDATIONS AND RELATED WORK

We give an overview of MACsec and explain how it protects the Ethernet layer. We describe mechanisms for configuration and key management and review related work on the application of MACsec in SDN.

A. OVERVIEW OF MACsec

IEEE 802.1AE [41] introduces the Media Access Control Security (MACsec) protocol. It provides point-to-point security between MACsec peers that are connected to the same LAN. Examples are links between two switches or routers, links between switches or routers and hosts, and links between hosts. MACsec ensures the integrity, confidentiality, and authenticity of Ethernet (IEEE 802) frames by applying symmetric encryption and cryptographic hash functions. In addition, it provides replay protection and a key exchange protocol to ensure perfect forward secrecy so that session keys are not affected by a compromised private key.

Figure 1 visualizes the principle and core components of MACsec. The network hosts A, B, and C are part of a LAN. Each network host has a MAC security entity (SecY) and a MAC security key agreement entity (KaY). The SecY provides secure MAC services over an insecure MAC service, i.e., it performs packet encryption, decryption, and authentication. The KaY discovers other KaYs in the LAN that participate in the same connectivity association (CA). It ensures that all network hosts are mutually authenticated and authorized. Afterwards, it creates and maintains secure channels (SCs) between the MACsec peers that are used by the SecYs to transmit and receive network packets. SCs are sender-specific, unidirectional, point-to-multi-point channels. Each SC holds multiple secure associations (SAs) that have a secure association key (SAK) used for encrypting, decrypting, and authenticating packets.

MACsec leverages cipher suites for packet encryption, decryption, and authentication. The standard defines the Advanced Encryption Standard in Galois/Counter mode (AES-GCM) with a block length of 128 bit (AES-GCM-128) as required cipher suite. If only packet authentication but no encryption is configured, MACsec applies the Galois Message Authentication Code (GMAC). Further specifications [43], [44] add GCM-AES-256, GCM-AES-XPN-128, and GCM-AES-XPN-256. Other cipher suites that meet requirements from the standard may also be applied.

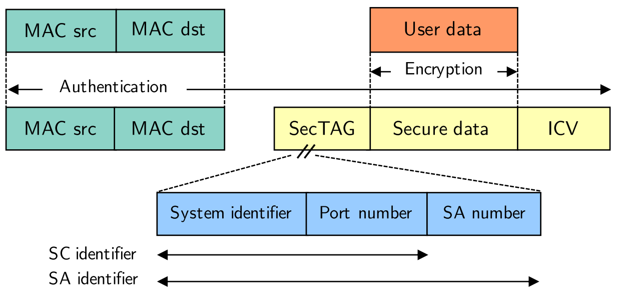


FIGURE 2. Packet structure of MACsec applied to an Ethernet packet. The MAC source and destination addresses of the MACsec packet are adopted from the original packet. The user data is transformed into a secure data block that is followed by the ICV calculated over the whole packet (1). The SecTAG includes among other things parameters to identify the SC and SA.

Figure 2 depicts the packet structure of MACsec. The Ethernet source and destination addresses of the MACsec packet are adopted from the original Ethernet packet. The secure data field either contains the encrypted user data of the original Ethernet packet or the user data in plain text if only MACsec packet authentication is configured. The integrity check value (ICV) field holds the result of a cryptographic hash function that is applied on the whole Ethernet packet including all header fields. The secure data and ICV are calculated by the chosen cipher suite. The security tag (SecTAG) contains MACsec information, e.g., the SC or SA identifier to indicate the corresponding SAK for packet encryption, decryption, and authentication. SecYs store multiple SAs with SAKs for each SC. When SAKs should be changed in rekeying, one SecY selects and uses a new SA for packet encryption and authentication. The SA field in the SecTAG reflects this transition so that the receiving SecY can switch to the new SA for decryption and authentication.

MACsec is supported by most access, distribution, and core switches from major manufacturers. Proprietary implementations such as the MACsec Toolkit [26] from Rambus provide control and data plane implementations that can be integrated in switches, routers, or network hosts. In addition, MACsec is part of the Linux kernel since version 4.6 [27] so that switch-to-host or host-to-host links can be protected with MACsec as well.

B. MACsec KEY EXCHANGE PROTOCOL

The 802.1AE standard does not define processes for key management or establishment of CAs and SAs between KaYs on MACsec peers. Therefore, network administrators are required to configure the CA affiliation and SAs with SAKs on every MACsec peer. IEEE 802.1X-2010 [42] introduces the MACsec Key Agreement (MKA) for automated peer discovery and exchange of SA data. With MKA, the initial CA affiliation and SA with SAK is derived from a connectivity association key (CAK). CAKs are either defined as pre-shared secret, derived from a master session key of an EAP process, or distributed by a MKA key server. Switches are required to implement additional functionalities to either exchange keying information via EAP with an AAA server or with an MKA key server. Independent of key exchange mechanisms, MACsec still needs to be initially set up on all peer devices via manual configuration.

C. COMPARISON TO VPNs

VPN technologies such as IPsec, OpenVPN, or WireGuard operate on Layer 3 or above. MACsec operates on Layer 2 and therefore provides link security for any higher-layer protocol. It applies point-to-point protection while VPNs aim at end-to-end protection. On every switch, router, or host in the network, MACsec packets are decrypted at the ingress port so that control plane functions targeting Layer 2 to 7 can be still applied. Access control lists that provide filtering based on IP addresses are an example. Then, packets are encrypted again at the egress port. MACsec is configured per Ethernet link so that administrators do not need to define additional policies for specific traffic to be encrypted. On routers and switches, MACsec is implemented on the packet forwarding chips, i.e., packet encryption and decryption is performed in line rate. In contrast, VPN technologies mostly encrypt and decrypt packets on ASICs that have limited bandwidth capacity. According to [30], IPsec traffic typically cannot exceed 40 Gb/s of bidirectional traffic while MACsec encryption and decryption scales with line rate.

D. APPLICATION OF MACsec IN SDN

Choi *et al.* [38] adopt MACsec to secure communication in vehicular networks between Linux-based electronic control units (ECUs). An SDN controller is responsible for automated setup of MACsec between ECUs to provide an end-to-end protection for network traffic. However, MACsec deployment is limited to the ECUs, i.e., MACsec deployment on SDN switches that connect the ECUs is not considered. Szyrkowiec *et al.* [55] develop an intent-based multi-layer orchestrator as application that interfaces an SDN controller. It automatically deploys protection technologies such as IPsec or MACsec on legacy switches through different southbound protocols, e.g., OpenFlow, NETCONF, or RESTCONF. However, MACsec deployment on SDN switches is not considered. Bentstuen and Flathagen [36] propose to implement MACsec for SDN but do not formulate any concrete approach. Vajaranta *et al.* [56] discuss implementation experiences and design challenges for WAN overlays using SDN and propose MACsec as viable option to implement link layer encryption. However, the presented implementation is limited to OpenVPN. Automated configuration of MACsec on SDN switches is proposed, but not part of the presented implementation. Mohamed *et al.* [49] describe a mechanism for MACsec key distribution of particular MACsec flows to switches. MACsec flows are end-to-end SCs that break up the point-to-point concept of the original standard. They are realized by configuring MACsec keys only on both end peers, but not on the peers in between. As prerequisite, all MACsec peers are expected to forward MACsec packets if no key for the received packet is found. OpenCORD [17] describes another example of MACsec deployment in SDN.

The SecY remains on the SDN switch while the KaY is transposed into a northbound application running on the ONOS SDN controller. SecYs on SDN switches are then configured via a NETCONF interface by the SDN controller. However, the authors state that MACsec support on SDN switches is still under developed so that only key agreement and configuration were implemented and tested in a Mininet simulation. Layer 2 packet encryption has not been examined.

III. LINK DISCOVERY IN SDN: FOUNDATIONS AND RELATED WORK

Automated deployment of MACsec by a SDN controller requires a topology view that is maintained through topology monitoring with link discovery between SDN switches. We give an overview of link discovery in SDN and describe the OpenFlow Discovery Protocol (OFDP). We review related work on variants of the OFDP that are optimized regarding security, efficiency, and applicability in hybrid SDN networks.

A. TOPOLOGY MONITORING AND LINK DISCOVERY IN SDN

Topology monitoring in SDN maintains a network map on the SDN controller that consists of SDN switches and links in between. In contrast to legacy networks, topology monitoring in SDN can be limited to link discovery. This is because of the mandatory connection setup routine between SDN switches and the SDN control plane whenever a SDN switch is started. Thereby, the control plane knows about the presence of all SDN switches in the network so that only links need to be detected. In OpenFlow, SDN switches establish a connection to a pre-configured SDN controller right after start. The SDN controller receives information about the SDN switch, e.g., a list of all physical ports, within the OF handshake at connection setup. With the P4Runtime API, the SDN controller may connect to P4 switches right after start.

B. OpenFlow DISCOVERY PROTOCOL (OFDP)

The OpenFlow Discovery Protocol (OFDP) was the first de-facto standard for link discovery in SDN. It is based on the Link Layer Discovery Protocol (LLDP) [45], the most widely used protocol for link detection in legacy networks. The Cisco Discovery Protocol (CDP) [2] is a proprietary alternative but less widely used. LLDP advertisements include information about the identity of a host, its capabilities, and its current status. LLDP protocol data units (PDUs) are periodically sent as payload of Ethernet frames with a multicast receiver address and the EtherType $0 \times 88cc$. Figure 3 depicts their structure. The PDUs may contain various type-length value (TLV) blocks, the standard defines three required TLV blocks. First, the Chassis ID TLV identifies the sending host, e.g., by its MAC address. Second, the Port ID TLV identifies the sender's port, e.g., its physical port number. Last, the Time-to-Live TLV defines the time validity of the information. Optional TLV blocks, e.g., the system's name defined by the administrator, and custom TLVs may be used as well. Network hosts

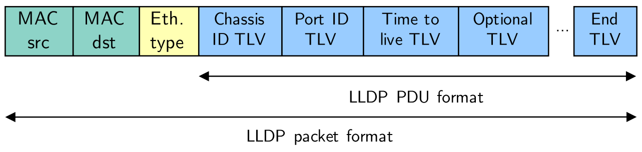


FIGURE 3. Format of LLDP packet and LLDP PDU.

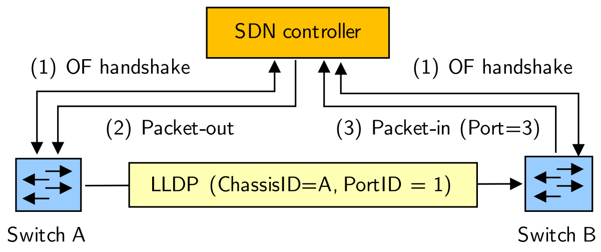


FIGURE 4. Link discovery in SDN with OFDP. The SDN controller learns about the SDN switch within the OF handshake (1). Afterwards, it sends out an LLDP packet on a particular port of an SDN switch via a packet-out message (2). Another SDN switch that receives the LLDP packet forwards it back to the SDN controller using a packet-in message (3).

that implement LLDP can receive, but not request LLDP information. Legacy switches periodically send out LLDP packets on each active port as described before. The packets are received, processed, and dropped by neighbouring LLDP agents on switches. They store the received information in the management information bases (MIBs) that can be queried via SNMP.

The OFDP leverages LLDP as introduced before but delegates all functionalities to the SDN controller. It uses packet-out messages to send out created network packets over particular ports of an SDN switch and packet-in messages to receive packets from the SDN switch that match specific criteria, e.g., an LLDP EtherType. Figure 4 depicts the process of link discovery with OFDP. First, the SDN controller learns about the switch identity and its ports within the OpenFlow handshake (1). Afterwards, it creates dedicated LLDP packets for all ports of a switch that are sent out via packet-out messages (2). For incoming LLDP packets, the OpenFlow switches are configured to forward any LLDP packet as packet-in message to the SDN controller (3). The packet-in message includes the LLDP packet with the Chassis and Port ID of the sender along the identity and the ingress port of the receiving SDN switch. By repeating this process for each port on each switch, the SDN controller performs link discovery.

C. OPTIMIZED VARIANTS OF OFDP

We review related work on optimized variants of OFDP that can be subdivided into publications investigating the security of OFDP, efficiency of OFDP, and applicability of OFDP in hybrid networks.

1) SECURITY OF OFDP

Alharbi *et al.* [31], [32], Azzouni *et al.* [33], and Nguyen and Yoo [50] show that OFDP is vulnerable to spoofing attacks. Injected LLDP control messages may create fake links that redirect traffic to the host of an attacker. Azzouni *et al.* [33] demonstrate that OFDP is additionally vulnerable to controller fingerprinting, switch fingerprinting, and LLDP flooding attacks. Nguyen and Yoo [50] show that OFDP is vulnerable to replay attacks. Faked LLDP packets result in incorrect link information of the topology. As improvement, Alharbi *et al.* [31], [32] propose to add a message authentication code (MAC) and a message identifier to each packet to provide authentication, packet integrity, and that prevent replay attacks. sOFTDP [34] encrypts LLDP packets to further prevent fingerprinting attacks. Marin *et al.* [47] investigate the security of SDN topology discovery mechanisms for OpenFlow. The authors propose mandatory integrity checks of LLDP and countermeasures against out-of-band channel and host location hijacking attacks. However, the latter two are not relevant in switch-to-switch link detection as required by MACsec.

2) EFFICIENCY OF OFDP

Azzouni *et al.* [33] and Rojas *et al.* [53] show that OFDP results in too many packet-out messages as the SDN controller has to create and send out one message for every active port on each SDN switch. As an improvement, ONOS [16] applies LLDP with Port IDs set to zero. The SDN controller creates one LLDP packet for every SDN switch that is configured to output the LLDP packet on all ports. This process is repeated for all SDN switches. Adjacent SDN switches are configured to forward received LLDP packets to the SDN controller so that it learns about the unidirectional link. Pakzad *et al.* [52] propose to reduce the number of packet-out messages by rewriting LLDP packets on the SDN switch. sOFTDP [34] introduces several mechanisms to shift large parts of link discovery back to the SDN switch. It adds liveliness detection mechanisms for switch ports and memorizes topology information locally on the SDN switch that asynchronously notifies the SDN controller about specific events. Rojas *et al.* [53] propose the Tree Exploration Discovery Protocol. SDN controllers create and send out specific frames flooded in the network that explore its topology. However, all concepts that shift functionality back to the SDN switches require extensive functional changes on the fixed-function data plane of typical SDN switches.

3) APPLICABILITY OF OFDP IN HYBRID NETWORKS

Hybrid networks consist of SDN and non-SDN switches. OFDP can be only applied to detect links in networks that consist of SDN switches. Legacy switches that may connect SDN switches process and discard received LLDP packets. The Broadcast Domain Discovery Protocol (BDDP) is a non-standardized approach that is implemented by several SDN controllers [3], [15], [19]. BDDP messages adapt the LLDP packet structure but use a broadcast Ethernet address instead of a multicast Ethernet address and the custom EtherType 0×8999. SDN switches are programmed to forward received BDDP packets to the SDN controller, just as with LLDP. Legacy switches flood the packet via all ports because of the broadcast address. They relay BDDP packets so that links will appear as single hops no matter how many legacy devices are on the path between two SDN switches. Ochoa-Aday *et al.* [51] show that the usage of broadcast packets leads to inefficient and excessive use of network resources. The authors propose a two-phase process for topology detection. First, the SDN controller performs link discovery using OFDP as described before. Afterwards, it outputs BDDP packets on any active port for which it does not detect a direct link to another SDN switch via LLDP. This way, the SDN controller detects direct links via LLDP and indirect links via BDDP.

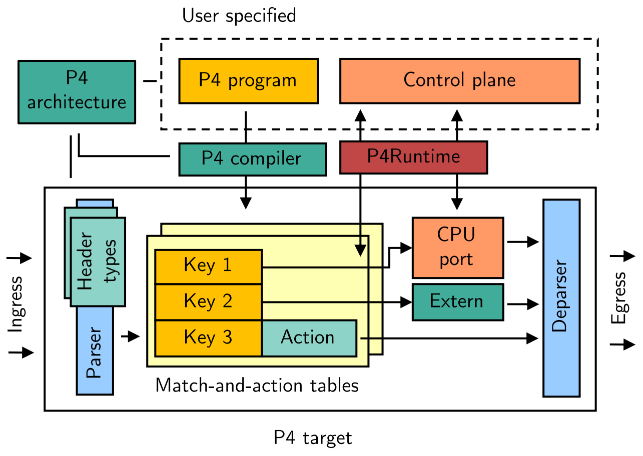


FIGURE 5. Concept of P4$_{16}$: P4 pipline and interaction with the control plane.

IV. P4: FOUNDATIONS

We give a brief overview of P4 with its core components for programming a P4 switch, describe the P4Runtime, and present examples for P4 software and hardware targets.

A. OVERVIEW

P4 is a domain-specific language for programmable data planes of packet forwarding devices. It offers high-level constructs that are optimized for specifying their forwarding behavior. P4 was first published in 2014 [37]. Today, the specification and development takes place in the non-profit P4 Language Consortium [21] with over 110 members from industry and academia. P4$_{16}$ [24] is the latest version of the language specification, the source code of all related components is available under an Apache license.

Figure 5 depicts P4's core concept and components. P4 programs describe the entire behavior of packet forwarding devices. They are formulated for a particular P4 architecture. A P4 architecture defines the P4 programming model of a packet forwarding device. P4 targets are software or hardware packet forwarding devices that implement a specific P4 architecture. Target-specific P4 compilers generate binary code from P4 programs that can be loaded on the P4 targets.

B. CORE COMPONENTS FOR PROGRAMMING A P4 SWITCH

Figure 5 shows all core components of the P4 pipeline following the specification of $P4_{16}$. We explain them in the following.

- **P4 header types** describe the format of packet headers through an ordered collection of base types. For example, an Ethernet header is described by bit vectors for the MAC source address, the MAC destination address, and the EtherType.
- **P4 parsers** are state machines that extract packet data through applying predefined sequences where data is identified and extracted based on P4 header types. For instance, the value of a parsed EtherType field of a packet determines the following extraction state which could be LLDP, MACsec, or IP.
- **P4 tables** are match-and-action structures mapping user-defined keys to particular P4 actions that may manipulate packet data.
- **P4 externs** are functions provided by a P4 target that can be used within P4 programs. P4 externs have a defined interface with a set of methods that can be used in the P4 program. An example would be a function that calculates a checksum for a given chunk of data.
- The **P4 deparser** assembles the headers back into a well-formed network packet that can be sent out via an egress port of the packet forwarding device.

C. P4Runtime

The P4Runtime framework provides an API for controlling P4 targets. Its operation is visualized in Figure 5. The P4Runtime features the manipulation of match-and-action tables through the control plane. In addition, it provides a CPU port for sending out and receiving packets similar to the packet-in and packet-out mechanism known from OpenFlow. P4Runtime leverages gRPC [9] that is based on HTTP/2 and protocol buffer [8] data structures. The connection between P4 targets and the control plane can be secured through TLS with optional client and server certificates for mutual authentication.

D. P4 SOFTWARE & HARDWARE TARGETS

The BMv2 [6] is the most widely-used P4 software target platform that features multiple P4 targets. Examples are a P4 target with the P4Runtime API (simple_switch_grpc) or a P4 target implementing the Protocol Independent Switch Architecture (PISA). The NetFPGA-SUME [57] and the Netcope NFB-200G2QL [13] are P4 targets that are based on FPGA platforms. Ethernet switches featuring the Barefoot Tofino ASIC [1] are the most widely-used P4 targets nowadays. The Barefoot Tofino ASIC implements the PISA and offers 12.8 Tb/s of packet throughput in its latest version. It is part of Ethernet and whitebox switches that also feature a general-purpose computing unit with a x86 CPU, RAM, and a SSD that runs a Linux-based operating system.

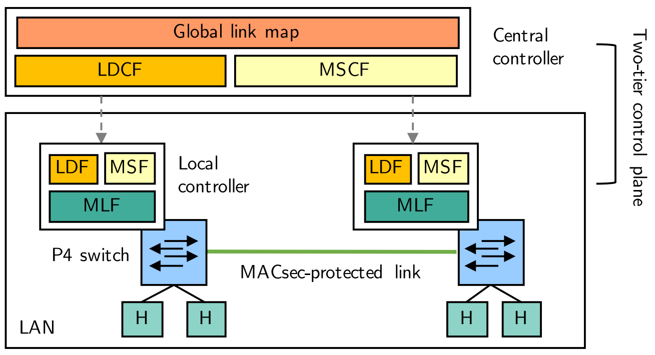


FIGURE 6. P4-MACsec in LAN with P4 switches. The local controllers consist of the MLF, LDF, and MSF. They communicate with the central controller that runs the LDCF and MSCF.

V. P4-MACsec

In this section, we describe P4-MACsec. We review its architecture and outline the three functional parts in detail: packet switching with MAC address learning, secure link discovery, and automated deployment of MACsec.

A. OVERVIEW

Figure 6 depicts the concept of P4-MACsec. It consists of a LAN with P4 switches that are steered by an SDN control plane. The switches provide Ethernet connectivity for the connected hosts and include the data plane functionality of the three functional parts. P4-MACsec features a *two-tier control plane structure*. Each P4 switch runs a local controller that connects to a central controller. The two-tier control plane offers the possibility to implement functions either on the global controller or on local controllers, or to split them and implement their parts on both the glocal controller and the local controllers. This reduces traffic in the management network, load on the central controller, and latency from forwarding packets between the SDN switches and the SDN controller. The concept of hierarchical SDN control is not novel but part of several SDN control plane architectures (e.g., [39], [40], [46]). With P4-based SDN, a two-tier control plane can be easily supported as most P4 targets feature general purpose computing capacities that can host a local controller. This is different with most OF switches which usually do not offer local computing capacities so that additional devices are needed to implement local controllers. Having the local controller on the switch eliminates dependencies on external devices and reduces the risk of failure. Bannour *et al.* [35] provide an overview of hierarchical and distributed SDN control planes and discuss their specifics. Although we see many advantages in the two-tier control plane architecture, P4-MACsec can also be implemented with only a central controller that carries out the functions of the local controllers. However, this is less elegant, causes more communication overhead, and is more prone to failure.

We design the three functional parts as follows. First, *MAC address learning for packet switching* ensures that

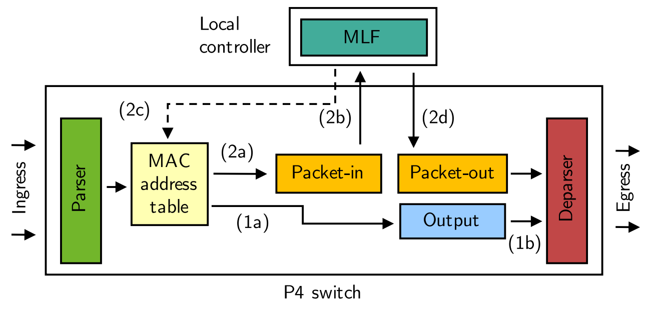


FIGURE 7. Process of forwarding Ethernet packets on the P4 processing pipeline. If the MAC table misses an entry, it triggers MAC address learning that is performed by the MLF running on the local controller.

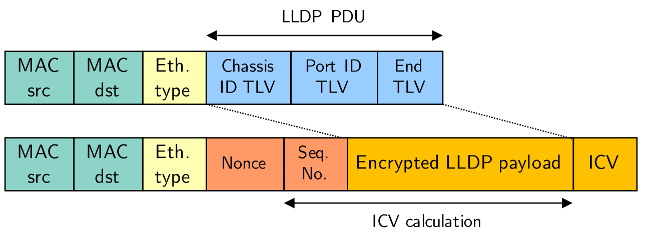


FIGURE 8. Proposed format for encrypting LLDP PDUs with AES-GCM. The EtherType from the original packet is followed by a Nonce that is used in AES-GCM encryption and decryption. A sequence number protects against replay attacks, a ICV holds a cryptographic checksum for authentication.

P4 switches forward Ethernet packets to provide Layer 2 connectivity for all network hosts. We implement it as MLF on the local controller and describe its details in Section V-B. Second, *secure link discovery* detects and monitors the link topology of the network using LLDP PDUs that are protected with AES-GCM. We implement it as two-tier control plane function consisting of LDFs and the LDCF. The LDFs run on the local controllers and inform the LDCF on the central controller about local links. The LDCF composes the global link map which is the basis for automated deployment of MACsec. We describe the details of this functional part in Section V-C. Third, *automated deployment of MACsec* dynamically creates, sets up, and maintains SCs on all switch-to-switch links from the global link map. We implement it as two-tier control plane function with decentral MSFs that receive configuration from a MSCF running on the central controller. We describe its details in Section V-D.

B. ETHERNET PACKET SWITCHING WITH MAC ADDRESS LEARNING

Although MAC address learning is a typical example for a local switch function, it cannot be solely implemented on the data plane of a P4 switch. Our proposed architecture for that functional part consists of two components. First, a MLF that runs on the local controller. Second, the data plane implementation for MAC address learning with the MLF and packet switching.

Figure 7 visualizes the process with all interactions between both components. When the P4 switch receives an Ethernet packet, it first checks if the source and destination MAC address of the Ethernet packet are already part of the MAC address table. If the MAC address table has entries for both (1a), the switch forwards the packet on the port specified for the destination MAC address (1b). If the MAC table yields no match for both addresses (2a), the switch forwards the Ethernet packet to the MLF running on the local controller as packet-in message (2b). The MLF first checks if the MAC source address and the ingress port is already part of the MAC address table. If not, the MLF updates the MAC address table (2c). Afterwards, the MLF floods the Ethernet packet on all ports except the ingress port from where it received the packet through the packet-out function (2d).

C. SECURE LINK DISCOVERY

The functional part of secure link discovery consists of three components. First, LDFs running on local controllers that do link detection and monitoring. Second, the LDCF running on the central controller that composes the global link map from local link information received from the LDFs. Third, the data plane implementation for receiving and sending out LLDP packets via packet-in and packet-out messages.

As novelty, we propose to create, encrypt, and decrypt LLDP PDUs on the LDF using AES-GCM with a common encryption key. We additionally introduce sequence numbers for LLDP PDUs to defend them against replay attacks. Figure 8 visualizes our proposed format of LLDP PDUs in comparison to the original format of LLDP packets. As in legacy LLDP, the MAC source address is set to the MAC address of the P4 switch. The MAC destination address and the EtherType are set to the LLDP defaults as introduced in Section III-B. LLDP PDUs consists of three TLVs. The Chassis ID TLV contains the identity of the switch as defined by the network administrator, the Port ID TLV contains the number of the physical port, the End TLV marks the end of the LLDP PDU. The common encryption key is installed and frequently updated by the LDCF on all LDFs. In addition, AES-GCM uses a 12 byte random number as nonce that is re-generated for each LLDP PDU leaving a particular port. It is part of the packet header following the EtherType. The four byte sequence number as protection against replay attacks is initialized with the LDF bootup timestamp and incremented with each packet sent out. The receiving LDF holds a sequence number counter for every physical port that is incremented with any received packet. The sequence number is part of the packet authentication of AES-GCM that is applied to the sequence number and on the LLDP PDU. Its result, the authentication tag, is stored within the ICV field following the encrypted LLDP PDU. Our approach resembles the one presented by Azzouni *et al.* [34] and protects against all attacks that were discussed in Section III-C.

Figure 9 visualizes the process of secure link discovery with all interactions among the three components. At startup, the LDF initiates a connection to the LDCF through a preconfigured IP address or FQDN (1). The LDCF installs the common key that is used for encrypting and decrypting LLDP packets with AES-GCM and instructs the LDF to start secure link discovery (2). The LDF generates and

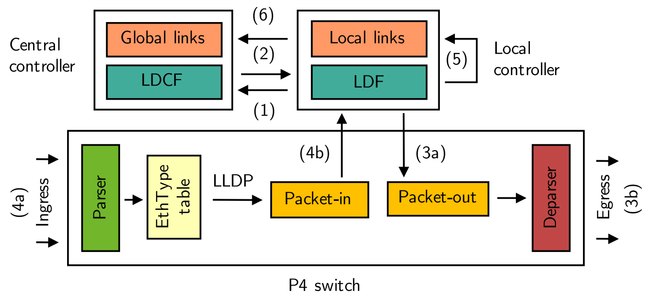


FIGURE 9. Process of secure link discovery using the local LDF, the LDCF, and the processing pipeline on the P4 switch.

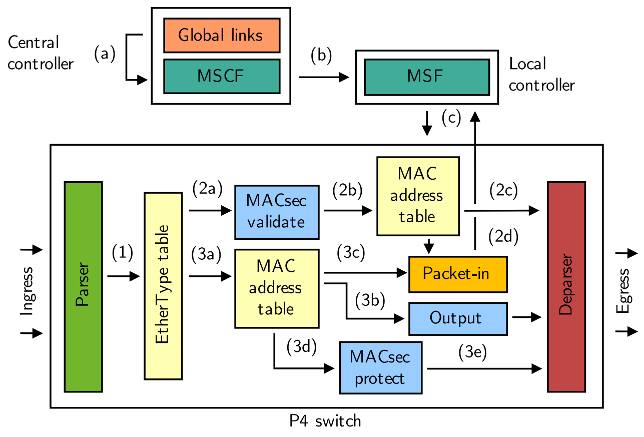


FIGURE 10. Process of automated MACsec deployment with the MSF, MSCF, and the processing pipeline of the P4 switch.

transmits encrypted LLDP packets for all active ports via packet-out messages to the P4 switch (3a) which then outputs the received packets on the specified ports (3b). As all other P4 switches received the same instruction to start link discovery, the P4 switch now receives encrypted LLDP packets from other P4 switches (4a) that it sends as packet-in messages to the LDF (4b). It performs decryption and extracts the Chassis and Port ID of the distant switch from the LLDP PDU along the physical port number of the packet-in message to update the map of local links (5). Finally, all changes of the local link map are sent to the global link map on the central controller (6). The global link map consists of bidirectional links in the form of two tuples that indicate the identity of the P4 switch and the physical port of the link as $(\text{Switch}_A, \text{Port}_A) \rightarrow (\text{Switch}_B, \text{Port}_B)$. The link topology can change at any time, e.g., when links between switches are added or when cables break. Therefore, link discovery is executed whenever the LDF running on the local controller receives status messages from its assigned switches, e.g., due to a port-down notification when a cable breaks. Link discovery is additionally performed after a fixed time interval of 30 seconds so that security can be sustained even if status messages from the P4 switches get lost.

In comparison to other approaches presented in Section III-C, our proposal does not require modifications of the SDN switches. All required mechanisms are implemented as control plane functions that rely on packet-in and packet-out mechanisms as offered by the CPU port in P4 or the packet-in and packet-out messages of OpenFlow. A two-tier control plane as used by P4-MACsec is not mandatory, the LDCF and LDF could be also combined into one application to be run on a central controller.

D. AUTOMATED DEPLOYMENT OF MACsec

Automated deployment of MACsec consists of three parts. First, a MSCF that creates two unidirectional SCs for each link of the global link map. Second, a MSF that runs on the local controller. It receives configuration data from the MSCF and sets up the SCs on the P4 switch. Third, the data plane implementation of MACsec that consists of the P4 processing pipeline and implementations of the MACsec validate and protect functions that can be used as P4 externs.

Figure 10 visualizes the automated deployment of MACsec with all interactions between the three components. We later describe the details of the MACsec validate and protect function. The P4 processing pipeline is an extension of the Ethernet packet forwarding pipeline that was introduced in Section V-B. At start, the MSCF creates and maintains MACsec SCs based on the global link map (a). It passes SC configuration data to the MSF (b) which updates various P4 tables in the processing pipeline (c). For the P4 processing pipeline, it sets a MACsec flag to entries of the MAC address table if a SC for that particular link exists.

Then, the data plane processing for packets works as follows. First, ingress packets are matched in an EtherType table (1). Ethernet packets matching the MACsec EtherType are forwarded to the MACsec validate function (2a). It validates its authenticity, optionally decrypts the secure data, and returns an Ethernet packet. Afterwards, the processing pipeline continues with Ethernet packet forwarding, i.e., it consults the MAC address table (2b), outputs the packet in case of a match for both source and destination MAC address (2c), or sends the packet as packet-in message to the MLF on the local controller otherwise (2d). Ethernet packets matching other EtherTypes are forwarded to the MAC address table (3a). If the MAC address table holds no flag for an SC for the particular destination MAC address, the packets are either sent out (3b) or passed to the MLF (3c) as explained in Section V-B. If the MAC table yields a match for source and destination MAC addresses of the packet and an SC flag, it forwards the packet to the MACsec protect function (3d). The MACsec protect function responds with a MACsec packet that can be sent out (3e) via the egress of the processing pipeline.

Figure 11 visualizes the MACsec protect and validate function. As SCs are unidirectional, the P4 switch has an ingress MACsec SC (IG-SC) table and an egress MACsec SC (EG-SC) table that maps secure channel identifiers (SCIs) to SAs. SAKs are part of the SA table which holds SAs for both ingress and egress SCs. The MACsec protect function (1) either encrypts or authenticates Ethernet payloads. It is applied to Ethernet packets if the MAC address table in the Ethernet packet forwarding pipeline has a flag set

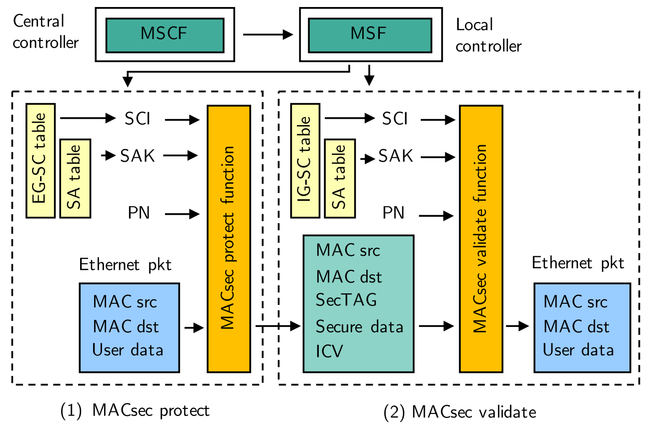


FIGURE 11. MACsec protect (1) and MACsec validate (2) functions implemented as P4 externs within the P4 processing pipeline.

for the particular physical port. Within the protect function, an EG-SC table maps the egress port number as SCI to security association identifiers (SAIs). The SA table holds the SAKs to be used for the protect function. The MACsec protect function receives the SAK and SCI from the SA table, the packet number from a packet counter as part of the switch, and the Ethernet packet. The AES-GCM cipher is initialized with a concatenation of the SCI and packet number as initialization vector. Afterwards, the EtherType and the payload are concatenated and encrypted. The MACsec protect function creates a new Ethernet packet with the MAC source and MAC destination address of the Ethernet packet, the MACsec EtherType, the SecTAG, the secure data, and the ICV. The MACsec validate function (2) works in a similar manner. Again, an IG-SC table maps SCIs to SAIs. The MACsec validate function receives the SAK and SCI from the IG-SC and SA table, the packet number from a packet counter, and the MACsec packet. It then checks the integrity and optionally decrypts the packet. It returns an Ethernet packet with the original Ethernet header and payload to the processing pipeline where forwarding and MAC address learning are performed as described before.

The MSCF creates and maintains MACsec SCs whenever the global link map changes. SCs exist until the corresponding link is deleted, e.g., in case of a link failure, the corresponding SC with all its SAs is deleted. If a new link is detected, the MSCF creates a new MACsec SC with SAs. In addition, MACsec SCs and SAs are renewed on a regular basis. Administrators define timeouts for encryption keys, the MSCF generates and installs new SAs on the P4 switches after the defined time interval through the MSF. Whenever SCs are created, changed, or deleted, configuration data and SAs are passed to the MSF that programs the P4 switch through writing in the EG-SC, IG-SC, and SA table.

VI. PROTOTYPICAL IMPLEMENTATION WITH MININET

In the following, we describe a prototypical implementation of P4-MACsec. We review the Mininet testbed environment and describe the three components of P4-MACsec in detail.

A. TESTBED ENVIRONMENT

We use the Mininet [12] network emulator to build the testbed environment for the prototypical implementation. We use the BMv2 P4 software target [6] for implementing the P4 switch and run the local controllers and the central controller as Python applications. For testing purposes, we additionally run Mininet network hosts that are connected to the P4 switches. All testbed components are executed within a KVM/QEMU VM that runs Ubuntu 16.04 with 4 CPU cores and 4 GB RAM. The hypervisor host features an Intel Core i5 8250U CPU, 16 GB RAM, and an SSD. It runs Manjaro Linux in Version 18.1.5.

We publish the source code with instructions for the testbed under the Apache v2 license on GitHub [7]. As of 2020-02, the maintainers of BMv2 discuss about integrating our implementation as an example for a crypto extern in the BMv2 source code repository [5].

B. P4 SWITCH

We extend the simple_switch_grpc [29] P4 target of the BMv2 P4 software target [6] to later run our P4 program that describes the data plane functions. Figure 12 depicts its parts. First, we implement the MACsec protect and validate functions as P4 externs within the simple_switch_grpc P4 target. We program the extensions in C++ and use the EVP interface of OpenSSL [20] to apply AES-GCM for encryption, decryption, and packet authentication. Both functions can be used as P4 externs within the P4 processing pipeline. When accessing the functions from the P4 processing pipeline, packet header data are exchanged using P4 attributes where packet payload data can be accessed directly. Second, we implement an interface to the local controller. It leverages the P4Runtime API via gRPC and allows the local controller to modify entries of the P4 tables in the processing pipeline. In addition, it holds the CPU port interface that provides packet-in and packet-out messages as known from OpenFlow. Packets sent from the P4 pipeline to the CPU port are forwarded to the local controller, packets received from the CPU port are injected into the P4 processing pipeline. The data plane functions of P4-MACsec described as P4 processing pipeline in Section V are implemented as $P4_{16}$ program using P4 constructs as introduced in Section IV. The P4 program then is executed on the modified simple_switch_grpc P4 target.

C. LOCAL CONTROLLER

We implement the local controller as Python 2.7 application. Figure 12 depicts its parts. We use the gRPC library [4] to program the interfaces to the associated P4 switch and to the central controller. We use the Scapy library [28] to create and parse LLDP packets and the cryptography library [25] for applying AES-GCM to encrypt and decrypt LLDP packets. For development and testing purposes, we include a simple CLI in the local controller. It provides functions to write/read table entries and to display status changes on the P4 switch.

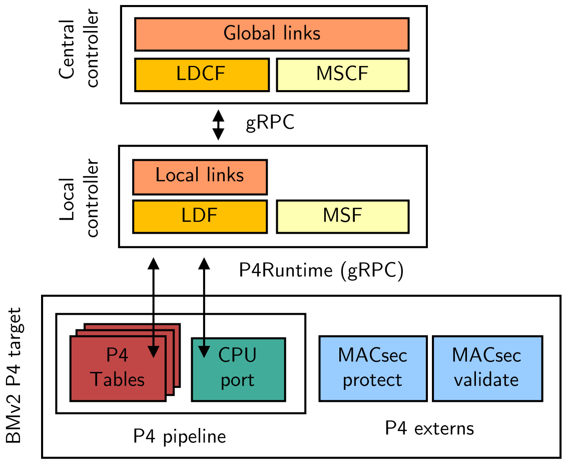


FIGURE 12. Structure of the prototypical implementation of MACsec. It consists of a BMv2 P4 software switch that implements the data plane functions of P4-MACsec and the two-tier control plane with the functions as introduced in Section V.

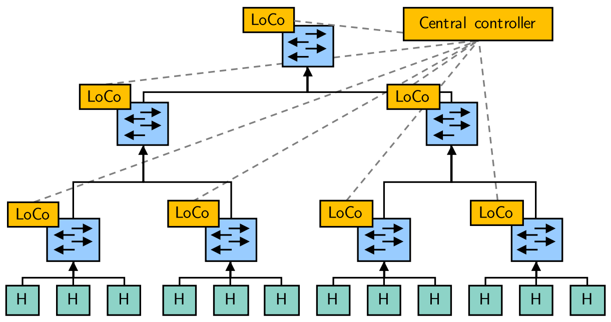


FIGURE 13. Virtual testbed environment that consists of network hosts, access switches, distribution switches, and a core switch. Each P4 switch is steered by a local controller (LoCo) that connects to the central controller.

D. CENTRAL CONTROLLER

We implement the central controller similar to the local controller as Python 2.7 application. Figure 12 depicts its parts. It also leverages the gRPC library [4] to build a gRPC interface to the local controller. It also features a simple CLI for development and testing purposes which displays information about the current topology and active MACsec SCs. The control plane functions of P4-MACsec can also be integrated in ONOS [15] or other controller frameworks. However, our objectives was a lightweight prototype using a slim and easy-to-understand controller implementation which directly leverages the P4Runtime API and gRPC library for communication. Thereby, we avoided dependencies on other controller frameworks which increase error space and implementation complexity.

VII. FUNCTIONAL VALIDATION

We describe experiments for functional validation, execute them on the testbed from Section VI, and report their results.

A. EXPERIMENT I: COMPLIANCE TO THE MACsec STANDARD

We first perform an experiment to examine the compliance of P4-MACsec to the IEEE 802.1AE standard. Therefore, we create a virtualized testbed that consists of a KVM virtual machine running Ubuntu Server in Version 18.04.1 LTS and our implementation of P4-MACsec on the BMv2 P4 software target as described in Section VI. The P4 switch connects another Ubuntu Server 18.04.1 LTS KVM/QEMU virtual machine that represents a network host behind a MACsec-enabled switch. We configure a static MACsec connection between the P4 switch and the Linux host to check whether the MACsec implementation for BMv2 is compatible with the Linux implementation of MACsec. On the Linux host, we configure the static MACsec connection using the iproute2 tools. For MACsec setup on the P4 switch, we use a simple Python script that adds the corresponding entries in the EG-SC, IG-SC, and SA tables of the P4 processing pipeline. We successfully validate that the Linux host communicates with the P4 switch via MACsec in different communication scenarios, e.g., ICMP or streaming random data via TCP connections with netcat. This does not validate a full compliance to all parts of the MACsec standard but demonstrates that the P4 switch can communicate via MACsec with legacy devices.

B. EXPERIMENT II: FULL P4-MACsec SCENARIO

We now investigate the complete set of functionality of P4-MACsec. Therefore, we create the topology depicted in Figure 13. It follows the model of hierarchical network switches that consists of core, aggregation, and access switches. A set of 12 network hosts is split into four groups, each attached to an access switch. The four access switches are connected to two aggregation switches that are connected by a single core switch. The testbed network is a single Layer 2 domain, i.e., network packets are forwarded based on their MAC address. After starting the Mininet testbed, we verify the following aspects. First, we examine that topology monitoring works correctly. In initial link discovery, we verify that the detected topology matches the actual network topology. Afterwards, we sporadically remove and re-add links between switches and supervise the process of link monitoring on the central controller via a CLI. Second, we examine that automated deployment of MACsec and rekeying works correctly. We investigate MACsec setup after changes in link monitoring through supervising the EG-SC, IG-SC, and SA tables on all P4-MACsec switches. Last, we examine packet switching and correct setup of MACsec protection. Therefore, we use ICMP and netcat to create network traffic between various pairs of network hosts in the experiment scenario. We investigate packet traces on links between switches and verify that all packets are protected by MACsec.

VIII. PERFORMANCE EVALUATION

We describe experiments for performance evaluation, execute them on the testbed from Section VI, and analyze their results.

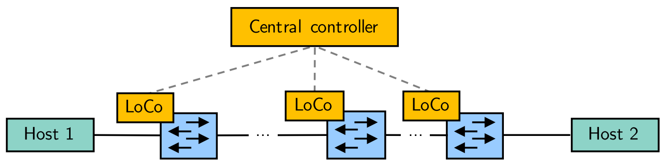


FIGURE 14. Evaluation testbed that consists of two network hosts that are attached to two P4 switches. In between, we vary from 0 to 6 additional P4 switches to form variable-length P4 switch chains for the evaluation experiments.

A. EVALUATION SETUP

Figure 14 depicts the evaluation setup. It consists of two network hosts that are attached to two P4 switches with 0 to 6 P4 switches in between. We perform performance evaluation experiments to investigate the goodput and RTT. We vary the number of P4 switches between the two network hosts and measure goodput and RTT for 1 to 8 hops. For each evaluation experiment, we consider three scenarios. In the first scenario, MACsec is disabled, i.e., the P4 switches between Host 1 and Host 2 only perform MAC address learning and Layer 2 forwarding. In the second scenario, we enable MACsec so that all packets between Host 1 and Host 2 are protected with AES-GCM implemented as P4 extern. In the third scenario, we enable MACsec but skip AES-GCM encryption and decryption in the P4 extern so that only plaintext payloads are sent within the MACsec packets. Although encapsulating plaintext payloads is not part of the MACsec standard, we still use this scenario to measure the effect of AES-GCM encryption/decryption by comparing it to network packet exchange with a P4 extern.

B. TCP GOODPUT

We first investigate the TCP goodput in P4-MACsec in our setting with software switches. To that end, we measure TCP transmissions between Host 1 and Host 2 with iperf3 [11]. Host 1 runs an iperf server, Host 2 runs an iperf client. We perform three runs, each with a duration of 30 seconds. Figure 15 depicts the results calculated as average over the three runs. The observed TCP goodput decreases with the number of hops, which is a result of increased round-trip time due and in particular of added queuing delay. However, we witness that forwarding without MACsec achieves significantly larger throughput than forwarding with MACsec. Obviously, there is additional packet processing delay with MACsec. However, this is only to a minor degree due to encryption, which can be seen by the fact that MACsec without encryption decreases the TCP goodput in a similar manner. Therefore, we conclude that the mere usage of externs on BMv2 increase the packet processing delay and reduces the TCP goodput.

C. ROUND-TRIP TIME (RTT)

In the second experiment, we investigate the RTT between the two network hosts that are connected by 1 to 8 P4 switches in between. We use the ping tool on Host 1 to send 1000 consecutive ICMP echo requests to Host 2. We set an idle period of

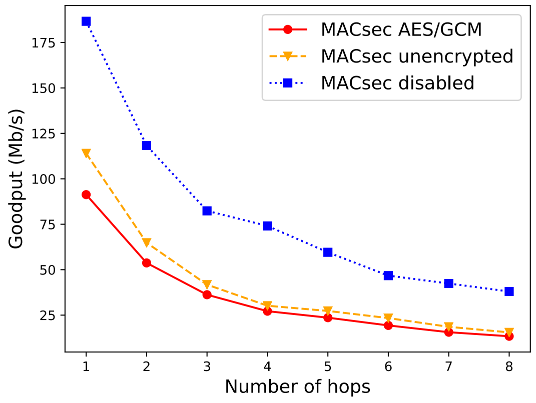


FIGURE 15. TCP goodput evaluation with 1 to 8 hops represented by P4 switches between two network hosts with iperf3. We consider three scenarios: disabled MACsec, enabled MACsec with encryption and decryption using AES-GCM, and enabled MACsec without encryption and decryption.

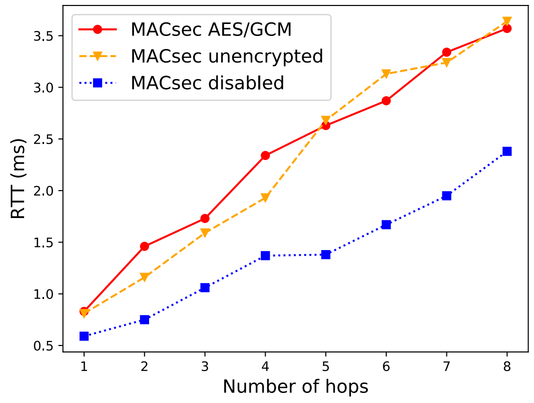


FIGURE 16. RTT evaluation with 1 to 8 hops represented by the P4 switches between two network hosts with pings. We consider three scenarios: disabled MACsec, enabled MACsec with encryption and decryption using AES-GCM, and enabled MACsec without encryption and decryption.

0.01 seconds between two ICMP packets and perform three runs of the experiment. Figure 16 depicts the RTTs calculated as average over the three runs. The evaluation results are similar to those of the experiment for TCP goodput. Enabled MACsec causes an increase of the RTT. Again, applying or omitting AES-GCM in the P4 extern does not cause large differences in the RTT. As in the experiment for TCP goodput, the interaction between the P4 pipeline and the MACsec protect and MACsec validate P4 externs in BMv2 seems to cause the negative effects on the RTT.

D. MACsec SETUP TIME

As described in Section V, the two-tier control plane automatically configures and enables MACsec on all assigned P4 switches. As described in Section II-B, MACsec deployment requires manual configuration effort regardless of static key setup on all switches or MKA. With P4-MACsec, that configuration effort completely disappears. The process of link discovery, MACsec setup, and MACsec rekeying are performed within nearly not measurable time. The two-tier control plane processes all events sequentially, i.e., delays might visibly increase only with a very large number of controlled P4 switches.

IX. IMPLEMENTATION ON NetFPGA SUME

In the following, we briefly describe the NetFPGA SUME [14] platform and outline our experiences in implementing P4-MACsec for that platform.

A. NetFPGA SUME PLATFORM

The NetFPGA SUME board is an open-source platform for rapid prototyping of network applications with support for bandwidths up to 100 Gb/s. It features a Virtex-7 690T FPGA, four SFP+ network transceivers, and a PCI Express interface to the host system [57]. The P4-NetFPGA project [22] transforms the NetFPGA SUME board into a hardware P4 target. P4 programs are transformed into SDNet descriptions by the P4-SDNet compiler that creates HDL modules which run as part of the reference architecture of the NetFPGA SUME board.

B. IMPLEMENTATION OF P4-MACsec

We modify the P4 processing pipeline of our software prototype to cope with limitations of the NetFPGA SUME architecture, e.g., a missing look-ahead function in packet parsing or the limitation to a single control block instead of multiple control blocks in the P4 processing pipeline. We implemented AES-GCM based on a publicly available Verilog module from OpenCores [18]. However, we were not able to create a fully working P4-MACsec switch due to two severe limitations. First, the NetFPGA SUME platform does not provide functions to parse or access variable-length payloads of network packets. Therefore, payloads of network packets need to be parsed as headers, which limits the implementation to fixed-length packets. Last, exchange of packet data between the P4 processing pipeline and the P4 external function is limited. Currently, data that is transferred from the P4 processing pipeline to a P4 external function needs to be transmitted within one clock cycle of the FPGA. Due to timing limitations, it is only possible to transmit very small amounts of data. The developers from the P4-NetFPGA project confirmed that its current version does not provide support for processing complete network packets within P4 externs. We were able to increase data to be exchangeable to 128 bytes by reducing the base clock frequency of the NetFPGA. However, this is still far away from applicability to real-world problems. A packet streaming function was announced, but is not available so far. Summing up, both limitations did not allow us to build a prototype that is suitable for real-world scenarios with variable-length packets exceeding a total length of 128 bytes.

Scholz *et al.* [54] pursued implementation of cryptographic hash functions on P4 data planes. NetFPGA SUME is part of their examined platforms. The authors confirm our experiences and propose a workaround where cryptographic hash functions are relocated to the platform's egress path behind the synthesized P4 program. However, if packet processing within the P4 programm relies on the output of the cryptographic hash functions, other workarounds need to be considered. All proposed workaround modifications require in-depth knowledge about FPGA programming. From our point of view, this conflicts with P4's original idea of platform-independent and abstract network programming.

X. CONCLUSION

In this work we proposed P4-MACsec, a concept to automatically protect links between switches with MACsec in P4-based SDN. Our concept features a P4 data plane implementation for MACsec including encryption and decryption using AES-GCM. P4 switches are steered by a novel two-tier control plane that consists of local controllers running on all P4 switches that connect to a central controller. We presented a novel mechanism for link discovery using encrypted LLDP packets and automated deployment of MACsec link protection. P4-MACsec completely eliminates previous configuration efforts for MACsec. We presented the architecture of P4-MACsec and demonstrated its feasibility in a prototypical implementation for the BMv2 P4 software target. We used that prototype to experimentally validate P4-MACsec in a virtualized testbed built with Mininet and performed evaluation experiments. We also reported on unsuccessful efforts to implement P4-MACsec on the NetFPGA SUME platform. Our work is an example for P4 switches supporting security features like authentication, encryption, and identity checks. Other applications may be traffic protection on different layers, e.g., Layer 3 VPNs. Therefore, P4 switches should offer native functional blocks for encryption and decryption and overhead-free interfaces to P4 externs.

LIST OF ACRONYMS

MACsec	Media Access Control Security
MKA	MACsec Key Agreement
SDN	Software-Defined Networking
OF	OpenFlow
BMv2	Behavioral Model version 2
FPGA	field programmable gate array
AES-GCM	Advanced Encryption Standard in Galois/Counter mode
LLDP	Link Layer Discovery Protocol
SecY	MAC security entity
KaY	MAC security key agreement entity
CA	connectivity association
SC	secure channel
SA	secure association
SAK	secure association key
GMAC	Galois Message Authentication Code
ICV	integrity check value
SecTAG	security tag
CAK	connectivity association key
ACL	access-control list
ECU	electronic control unit
OFDP	OpenFlow Discovery Protocol
CDP	Cisco Discovery Protocol

PDU	protocol data unit
TLV	type-length value
MIB	management information base
MAC	message authentication code
BDDP	Broadcast Domain Discovery Protocol
SCI	secure channel identifier
SAI	security association identifier
RTT	round-trip time
MLF	MAC address learning function
LDF	link discovery function
LDCF	link discovery controller function
MSF	MACsec function
MSCF	MACsec controller function
IG-SC	ingress MACsec SC
EG-SC	egress MACsec SC
PISA	Protocol Independent Switch Architecture

ACKNOWLEDGMENT

The authors would like to thank Joshua Hartmann for fruitful discussions and programming contributions.

REFERENCES

[1] *Barefoot Networks: Tofino 2*. Accessed: Feb. 2, 2020. [Online]. Available: https://www.barefootnetworks.com/products/brief-tofino-2/

[2] *Cisco Discovery Protocol Configuration Guide*. Accessed: Feb. 2, 2020. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/cdp/configuration/15-mt/cdp-15-mt-book/nm-cdp-discover.html

[3] *Floodlight: OpenSource SDN Controller*, Accessed: Feb. 2, 2020. [Online]. Available: http://www.projectfloodlight.org/floodlight/

[4] *GitHub: GRPC Python*. Accessed: Feb. 2, 2020. [Online]. Available: https://github.com/grpc/grpc/tree/master/src/python/grpcio

[5] *GitHub: Open Pull Requests of p4lang/Behavioral-Model: Add Crypto Extern to Behavioral-Model*. Accessed: Feb. 2, 2020. [Online]. Available: https://github.com/p4lang/behavioral-model/pull/834

[6] *GitHub: P4lang/Behavioral-Model (BMv2)*. Accessed: Feb. 2, 2020. [Online]. Available: https://github.com/p4lang/behavioral-model

[7] *GitHub: Uni-Tue-kn/p4-Macsec*. Accessed: Mar. 29, 2020. [Online]. Available: https://github.com/uni-tue-kn/p4-macsec

[8] *Google Protocol Buffers*. Accessed: Feb. 2, 2020. [Online]. Available: https://developers.google.com/protocol-buffers/

[9] *Grpc*. Accessed: Feb. 2, 2020. [Online]. Available: https://grpc.io/

[10] *IEEE8021-SECY-MIB: Definitions of Managed Objects Supporting IEEE 802.1AE MACsec*. Accessed: Apr. 14, 2019. [Online]. Available: http://www.ieee802.org/1/files/public/MIBs/IEEE8021-SECY-MIB-200601100000Z.txt.

[11] *Iperf*. Accessed: Feb. 2, 2020. [Online]. Available: https://iperf.fr/

[12] *Mininet*. Accessed: Feb. 2, 2020. [Online]. Available: http://mininet.org/

[13] *Netcope P4*. Accessed: Feb. 2, 2020. [Online]. Available: https://www.netcope.com/en/products/netcopep4

[14] *NetFPGA*. Accessed: Feb. 2, 2020. [Online]. Available: https://netfpga.org

[15] *ONOS*. Accessed: Feb. 2, 2020. [Online]. Available: https://onosproject.org/

[16] *ONOS Wiki: Network Discovery*. Accessed: Feb. 2, 2020. [Online]. Available: https://wiki.onosproject.org/display/ONOS/Network+Discovery

[17] *OpenCORD Wiki: MAC Security*. Accessed: Feb. 2, 2020. [Online]. Available: https://wiki.opencord.org/display/CORD/MAC+Security

[18] *Open Cores: GCM-AES*. Accessed: Feb. 2, 2020. [Online]. Available: https://opencores.org/projects/gcm-aes

[19] *Open Daylight*. Accessed: Feb. 2, 2020. [Online]. Available: https://www.opendaylight.org/

[20] *OpenSSL*. Accessed: Feb. 2, 2020. [Online]. Available: https://www.openssl.org/

[21] *P4 Language Consortium*. Accessed: Feb. 2, 2020. [Online]. Available: https://p4.org/

[22] *P4->NetFPGA Wiki*. Accessed: Feb. 2, 2020. [Online]. Available: https://github.com/NetFPGA/P4-NetFPGA-public

[23] *P4 Runtime Spec. 1.0.0*. Accessed: Feb. 2, 2020. [Online]. Available: https://p4.org/p4runtime/spec/v1.0.0/P4Runtime-Spec.html

[24] *P4_16 Language Specification, Version 1.2.0*. Accessed: Feb. 2, 2020. [Online]. Available: https://p4.org/p4-spec/docs/P4-16-v1.2.0.html

[25] *Pyca/Cryptography Documenta*. Accessed: Feb. 2, 2020. [Online]. Available: https://cryptography.io/en/latest/

[26] *Rambus: MACsec Toolkit for Ethernet Security*. Accessed: Feb. 2, 2020. [Online]. Available: https://www.rambus.com/security/software-protocols/secure-communication-toolkits/macsec-tk/

[27] *RHD Blog: MACsec: A Different Solution to Encrypt Network Traffic*. Accessed: Feb. 2, 2020. [Online]. Available: https://developers.redhat.com/blog/2016/10/14/macsec-a-different-solution-to-encrypt-network-traffic/,

[28] *Scapy*. Accessed: Feb. 2, 2020. [Online]. Available: https://scapy.net/

[29] *Simple Switch Grpc*. Accessed: Feb. 2, 2020. [Online]. Available: https://github.com/p4lang/behavioral-model/tree/master/targets/simple_switch_grpc

[30] *Cisco: Innovations in Ethernet Encryption (802.1AE—MACsec)*. Accessed: Feb. 2, 2016. [Online]. Available: https://www.cisco.com/c/dam/en/us/td/docs/solutions/Enterprise/Security/MACsec/WP-High-Speed-WAN-Encrypt-MACsec.pdf

[31] T. Alharbi, M. Portmann, and F. Pakzad, "The (in)security of topology discovery in software defined networks," in *Proc. IEEE 40th Conf. Local Comput. Netw. (LCN)*, Oct. 2015, pp. 502–505.

[32] T. Alharbi, M. Portmann, and F. Pakzad, "The (In)Security of topology discovery in openflow-based software defined network," *Int. J. Netw. Secur. Its Appl.*, vol. 10, no. 3, pp. 01–16, May 2018.

[33] A. Azzouni, N. T. Mai Trang, R. Boutaba, and G. Pujolle, "Limitations of openflow topology discovery protocol," in *Proc. 16th Annu. Medit. Ad Hoc Netw. Workshop (Med-Hoc-Net)*, Jun. 2017, pp. 1–3.

[34] A. Azzouni, R. Boutaba, N. Thi Mai Trang, and G. Pujolle, "SOFTDP: Secure and efficient topology discovery protocol for SDN," 2017, *arXiv:1705.04527*. [Online]. Available: http://arxiv.org/abs/1705.04527

[35] F. Bannour, S. Souihi, and A. Mellouk, "Distributed SDN control: Survey, taxonomy, and challenges," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 333–354, 1st Quart., 2018.

[36] O. I. Bentstuen and J. Flathagen, "On bootstrapping in-band control channels in software defined networks," in *Proc. IEEE Int. Conf. Commun. Workshops (ICC Workshops)*, May 2018, pp. 1–6.

[37] P. Bosshart, "P4: Programming Protocol-independent Packet Processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95. Jul. 2014.

[38] J.-H. Choi, S.-G. Min, and Y.-H. Han, "MACsec extension over software-defined networks for in-vehicle secure communication," in *Proc. 10th Int. Conf. Ubiquitous Future Netw. (ICUFN)*, Jul. 2018, pp. 180–185.

[39] Y. Fu, "Orion: A Hybrid Hierarchical Control Plane of Software-Defined Networking for Large-Scale Networks," in *IEEE Int. Conf. Netw. Protocols (ICNP)*, 2014, pp. 569–576.

[40] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proc. 1st workshop Hot topics Softw. defined Netw.*, 2012, pp. 19–24.

[41] *IEEE Standard for Local and Metropolitan Area Networks—Media Access Control (MAC) Security*, Standard 802.1AE-2006, 2006.

[42] *IEEE Standard for Local and Metropolitan Area Networks—Port-Based Network Access Control*, Standard 802.1X-2010, 2010.

[43] *IEEE Standard for Local and Metropolitan Area Networks—Media Access Control (MAC) Security Amendment 1: Galois Counter Mode—Advanced Encryption Standard—256 (GCM-AES-256) Cipher Suite*, Standard 802.1AEbn-2011, 2011.

[44] *IEEE Standard for Local and Metropolitan Area Networks—Media Access Control (MAC) Security Amendment 2: Extended Packet Numbering*, Standard 802.1AEbw-2013, 2013.

[45] *IEEE Standard for Local and Metropolitan Area Networks—Station and Media Access Control Connectivity Discovery, Corrigendum 2: Technical and Editorial Corrections*, Standard 802.1AB-2009, 2015.

[46] T. Kohler, F. Durr, and K. Rothermel, "ZeroSDN: A highly flexible and modular architecture for full-range distribution of event-based network control," *IEEE Trans. Netw. Service Manage.*, vol. 15, no. 4, pp. 1207–1221, Dec. 2018.

[47] E. Marin, N. Bucciol, and M. Conti, "An in-depth look into SDN topology discovery mechanisms: Novel attacks and practical countermeasures," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 1101–1114.

[48] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, p. 69, Mar. 2008.
[49] P. S. Mohamed, "Network controller provisioned MAC sec keys," U.S. Patent 14 763 484, Aug. 27, 2019.
[50] T.-H. Nguyen and M. Yoo, "Analysis of link discovery service attacks in SDN controller," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, 2017, pp. 259–261.
[51] L. O. Aday. (2015). *Current Trends of Topology Discovery in OpenFlow-based Software Defined Networks*. [Online]. Available: https://upcommons.upc.edu/handle/2117/77672.
[52] F. Pakzad, M. Portmann, W. L. Tan, and J. Indulska, "Efficient topology discovery in software defined networks," in *Proc. 8th Int. Conf. Signal Process. Commun. Syst. (ICSPCS)*, Dec. 2014, pp. 1–8.
[53] E. Rojas, J. Alvarez-Horcajo, I. Martinez-Yelmo, J. A. Carral, and J. M. Arco, "TEDP: An enhanced topology discovery service for software-defined networking," *IEEE Commun. Lett.*, vol. 22, no. 8, pp. 1540–1543, Aug. 2018.
[54] D. Scholz, A. Oeldemann, F. Geyer, S. Gallenmuller, H. Stubbe, T. Wild, A. Herkersdorf, and G. Carle, "Cryptographic hashing in p4 data planes," in *Proc. ACM/IEEE Symp. Archit. Netw. Commun. Syst. (ANCS)*, Sep. 2019, pp. 1–6.
[55] T. Szyrkowiec, M. Santuari, M. Chamania, D. Siracusa, A. Autenrieth, V. Lopez, J. Cho, and W. Kellerer, "Automatic intent-based secure service creation through a multilayer SDN network orchestration," *J. Opt. Commun. Netw.*, vol. 10, no. 4, p. 289, Apr. 2018.
[56] M. Vajaranta, J. Kannisto, and J. Harju, "Implementation experiences and design challenges for resilient SDN based secure WAN overlays," in *Proc. Asia Joint Conf. Inf. Secur. (AsiaJCIS)*, Aug. 2016, pp. 17–23.
[57] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore, "NetFPGA SUME: Toward 100 Gbps as research commodity," *IEEE Micro*, vol. 34, no. 5, pp. 32–41, Sep. 2014.

FREDERIK HAUSER (Student Member, IEEE) received the master's degree in computer science from the University of Tuebingen, Germany, in 2016, where he is currently pursuing the Ph.D. degree with the Chair of Communication Networks. Since then, he has been a Researcher with the Chair of Communication Networks, University of Tuebingen. His main research interests include software-defined networking, network function virtualization, and network security.

MARK SCHMIDT (Member, IEEE) received the Diploma degree in computer science from the University of Tuebingen, Germany, in 2010, where he is currently pursuing the Ph.D. degree with the Chair of Communication Networks. Since then, he has been a Researcher with the Chair of Communication Networks, University of Tuebingen. His main research interests include software-defined networking, OpenFlow, high-speed networks, and virtualization.

MARCO HÄBERLE (Student Member, IEEE) received the master's degree in computer science from the University of Tuebingen, Germany, in 2018, where he is currently pursuing the Ph.D. degree with the Chair of Communication Networks. Since then, he has been a Researcher with the Chair of Communication Networks, University of Tuebingen. His main research interests include software-defined networking, P4, network security, and automated network management.

MICHAEL MENTH (Senior Member, IEEE) is currently a Professor with the Department of Computer Science, University of Tuebingen, and the Chair holder of Communication Networks. He has published more than 150 articles in the field of computer networking. His special interests are performance analysis and optimization of communication networks, resilience and routing issues, resource and congestion management, software-defined networking and the Internet protocols, industrial networking, and the Internet of Things.

1.3 xRAC: Execution and Access Control for Restricted Application Containers on Managed Hosts

xRAC: Execution and Access Control for Restricted Application Containers on Managed Hosts

Frederik Hauser, Mark Schmidt, and Michael Menth
Chair of Communication Networks, University of Tuebingen, Tuebingen, Germany
Email: {frederik.hauser,mark-thomas.schmidt,menth}@uni-tuebingen.de

***Abstract*—We propose xRAC to permit users to run special applications on managed hosts and to grant them access to protected network resources. We use restricted application containers (RACs) for that purpose. A RAC is a virtualization container with only a selected set of applications. Authentication verifies the RAC user's identity and the integrity of the RAC image. If the user is permitted to use the RAC on a managed host, launching the RAC is authorized and access to protected network resources may be given, e.g., to internal networks, servers, or the Internet. xRAC simplifies traffic control as the traffic of a RAC has a unique IPv6 address so that it can be easily identified in the network. The architecture of xRAC reuses standard technologies, protocols, and infrastructure. Those are the Docker virtualization platform and 802.1X including EAP-over-UDP and RADIUS. Thus, xRAC improves network security without modifying core parts of applications, hosts, and infrastructure. In this paper, we review the technological background of xRAC, explain its architecture, discuss selected use cases, and investigate on the performance. To demonstrate the feasibility of xRAC, we implement it based on standard components with only a few modifications. Finally, we validate xRAC through experiments. We publish the testbed setup guide and prototypical implementation on GitHub [1].**

I. Introduction

In this paper we consider the problem of permitting users to run special applications on managed hosts and to grant them access to protected network resources. This is an important challenge in practice as applications communicate with multiple peers and have multiple, possibly a priori unknown flows characterized by 5-tuples. Moreover, common packet filters such as firewalls or deep packet inspectors lack knowledge of allowed flows, and identifying traffic from specific applications becomes more difficult with traffic encryption using TLS.

We address this challenge by running applications in containers as so-called restricted application containers (RACs) on managed hosts. RACs provide selected sets of applications including their dependencies and configuration. The managed host gives users only limited freedom, e.g., they can download and launch RACs. We propose authentication and authorization (AA) for RACs so that their execution is restricted to authorized users. That means, the user identity, the integrity of the RAC's image, and the permission of the user to execute the RAC are verified before a RAC is launched. We suggest to extend this authorization also to protected network resources required by the RAC, e.g., to internal networks, servers, or Internet access. That means, appropriate network control elements, e.g., firewalls or SDN controllers, may be informed about authorized RACs and their needs. The authorized traffic can be identified by the RAC's IPv6 address. We call this concept xRAC as it controls execution and access for RACs.

We mention a few use cases that may benefit from xRAC. RACs may allow users to execute only up-to-date RAC images. Execution of a RAC may be allowed only to special users or user groups, e.g., to enforce license restrictions. Only selected users in a high-security area may get access to the Internet through a RAC-based browser which is isolated from the remaining infrastructure. Only selected users may be able to execute administration software with access to servers providing confidential material. RACs may be used for applications with increased quality of service (QoS) requirements, e.g., voice-over-IP, video conferences, or games. Their traffic may be preferentially treated by network elements.

To facilitate deployment of xRAC, we reuse and adapt standard technologies, protocols, and infrastructure. We leverage the Docker platform to create and deploy containerized applications. When the container management daemon (CMD) is requested to launch a RAC, it first issues an AA request and launches the RAC only after successful AA. We adopt 802.1X for AA purposes and adapt its components 802.1X supplicant (802.1X S), 802.1X authenticator (802.1X A), and 802.1X authentication server (802.1X AS). The CMD interfaces with an 802.1X container supplicant (802.1X CS) on the host, the 802.1X CS with a 802.1X container authenticator (802.1X CA), and the 802.1X CA with an 802.1X authentication server (802.1X AS). The 802.1X AS holds RAC-specific AA data, performs authentication, and returns authorization data to the 802.1X CA. The 802.1X CA interfaces with network control elements and configures access to network resources depending on authorization data. It also forwards the authorization data to the 802.1X CS. EAP-over-UDP and RADIUS are utilized for communication. To demonstrate the feasibility of xRAC, we provide a prototype based on existing 802.1X components, implement the 802.1X CS as plugin for the Docker authorization framework and the 802.1X CA as part of an SDN controller. We use a RADIUS server for 802.1X AS and extend its data structures to store AA data for users and RAC. We use this prototype to experimentally validate xRAC in a testbed.

This work was supported by the bwNET100G+ project which is funded by the Ministry of Science, Research and the Arts Baden-Württemberg (MWK). The authors alone are responsible for the content of this paper.

The rest of the paper is structured as follows. In Section II, we review technical background and related work on container virtualization. In Section III, we review technical background and related work on 802.1X and AA for applications. In Section IV, we present the architecture of xRAC in detail. Section V discusses use cases along benefits and limitations of xRAC. Section VI describes the prototypical implementation of xRAC which is used for its experimental validation in Section VII. In Section VIII, we briefly investigate on performance considerations of xRAC. Section IX concludes this work.

II. Container Virtualization: Technical Background and Related Work

We first introduce the concept of container virtualization, describe advantages and give an overview of Docker as a widespread implementation. Then, we review container security platforms and containers for GUI applications.

A. *Overview*

Containers implement virtualization on the operating system (OS) level. They provide virtualized OS environments that are isolated with regard to hardware resources and security. They share the OS kernel and may include binaries and libraries that are required to run one or several enclosed applications. In xRAC, we enclose only one special application with its dependencies and configuration in a container. Containers run on top of a container runtime and are managed by a container management daemon (CMD) that creates, starts and suspends containers. Examples for container platforms are Docker [2], Kubernetes [3], BSD jails [4], and Solaris containers [5].

B. *Advantages*

Virtualization facilitates efficient and flexible usage of hardware resources, improves security through isolation, and provides fault-safety and scalability through simple migration processes. Containers in particular have the following additional advantages. Due to the shared OS, containers require less CPU, memory, and hard disk resources. Container images are much smaller than virtual machines, which simplifies distribution among many recipients. Containers simplify application deployment. Instead of providing support for complex combinations of applications, dependencies, and user configurations, administrators just deploy containers that are tested prior to release. Containers have no bootup times, which makes them especially suitable for short-lived applications.

C. *Docker Container Virtualization Platform*

Docker [2] is one of the most popular container platforms today. Figure 1 provides a simplified overview of the Docker platform including its operations. A host with a common OS runs the Docker CMD that generates and manages containers and container images. Container images are write-protected templates that include applications with their dependencies. Containers are runtime instances that extend the write-protected container images by a writeable layer. Therefore, multiple container instances may share a common image. This introduces scalability with low hard disk requirements. The Docker CMD is controlled by the Docker client via a REST interface. The Docker client can be located on the host running the Docker CMD or on a remote host. The Docker command line interface (CLI) is an example for user control through CLI calls. The Docker CMD may connect to Docker registries that allow users to upload (push) or download (pull) container images. Those registries are either private or publicly available. Docker Hub [6] with more than 100.000 container images is an example for the latter. Common operations are build (1), pull (2), and run (3). With build, users may create individual container images. With pull, users may download container images from a repository to become part of the set of local images. With run, container images from the set of local images can be executed on the host system.

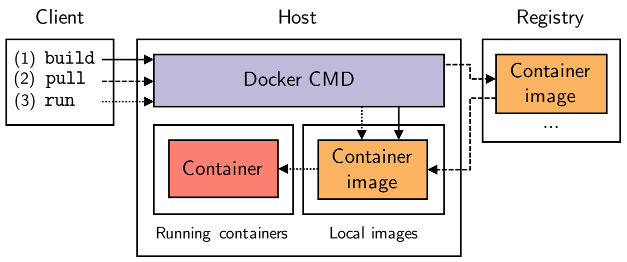


Fig. 1: The Docker architecture consists of the Docker client, the Docker CMD, and the Docker registry [7].

Docker uses several functions of the OS kernel [8]–[10] to provide isolation and resource emulation. It supports storage drivers, e.g., AuFS [11], OverlayFS [12], and ZFS [13], to enable file system stacking. The container format and runtime environment of Docker were adopted by the Open Container Initiative [14] as open industry standard.

D. *Container Security Platforms*

Container security platforms extend the CMD by security functions. Twistlock [15], [16] and the Aqua Container Security Platform [17] provide a runtime engine based on machine learning mechanisms to permanently monitor containers for detecting fraudulent behavior and special network firewalls to filter container traffic. The Sysdig Secure [18] platform allows the formulation of service-aware policies, i.e., policies that are based on applications, containers, hosts, or network activities. The platform provides alerts and actions based on policy violations, an event history, and incident captures. The Atomicorp Secure Docker Kernel [19] is a hardened Linux kernel that provides security-related features such as breakout protection, memory corruption protection, or prevention of direct userland access by the kernel. All platforms focus on monitoring and controlling potentially untrustworthy containers that are executed on a shared container runtime. Features for AA for users, containers, and their network flows are not part of those platforms.

The Docker Authorization Framework [20] is part of Docker since Version 1.10 [21]. It extends the Docker CMD by a REST interface to external authorization plugins. Requests from the Docker CMD, e.g., to start a container, are forwarded to an authorization plugin that implements mechanisms to decide whether to allow or deny the request. The Docker Authorization Framework does not implement security functions but provides a base for implementing such security concepts. xRAC leverages this framework.

E. *Containers for GUI Applications*

Containers typically deploy applications or services without graphical user interfaces (GUIs) that run in the cloud or on data center infrastructures. Examples are containers that enclose web applications with their requirements, e.g., an nginx web server with a PHP runtime and a MySQL database server. The idea of leveraging Docker containers to deploy desktop applications with a GUI was first presented in [22]. The author proposes to mount X11 sockets for GUI presentation and hardware devices of the host system, e.g., an audio card or a web camera, to the Docker container. Thereby, even more complex GUI applications such as the Chrome web browser, the Spotify music player, or the Skype video chat application can be run in containers and be deployed as container images. Today, many Docker container images with GUI applications can be downloaded from Docker Hub.

III. 802.1X: Technical Background and Related Work

We give an overview of 802.1X and explain how it supports AA. We present EAPoUDP which is an alternative protocol to carry AA data in 802.1X. We summarize how AA for applications is currently performed in practice and review another research work that provides AA for applications based on 802.1X.

A. *Overview*

IEEE 802.1X [23]–[25] introduces port-based network access control in wired Ethernet networks. However, it is mainly known from wireless 802.11 networks today. An example is Eduroam [26], a federation of wireless university campus networks. Participants can connect to the Internet no matter if located at their home institution or at a foreign university campus, e.g., while attending a conference.

Figure 2 depicts the three components of 802.1X and the principle of port-based network access control. The supplicant system is a network host that contains the 802.1X supplicant (802.1X S). The authenticator system contains the 802.1X authenticator (802.1X A) and controls network access of network hosts. Examples are access switches that connect network hosts to the main network. Prior to authorization, the supplicant system can reach the 802.1X A but not the network. The 802.1X AS is an authentication, authorization, and accounting server. It stores authentication data to verify user identities and authorization data to grant permission to the network. It authenticates the 802.1X S and delivers authorization information to the 802.1X A.

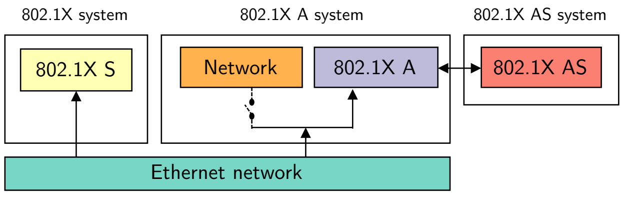


Fig. 2: Port-based authorization model of 802.1X [24].

B. *AA with 802.1X*

802.1X leverages the Extensible Authentication Protocol (EAP) [27] and the Remote Authentication Dial In User Service (RADIUS) [28] to exchange AA data. Both provide a fixed request and response scheme to exchange AA data. The Diameter protocol [29] is a less widespread alternative. Authentication data is transmitted in Ethernet frames as EAP-over-LAN (EAPoL) encapsulation between the 802.1X S and 802.1X A and as EAP-over-RADIUS (EAPoRADIUS) between the 802.1X A and 802.1X AS. Figure 4 depicts the packet structure of EAPoL. Authorization data is transmitted in RADIUS frames between the 802.1X AS and 802.1X A.

We explain the details of 802.1X with the four-step process of AA as depicted in Figure 3. In the first step (1), the 802.1X S initializes authentication by sending an EAPOL-Start message to the 802.1X A. In the second step, the 802.1X A requests the identity from the 802.1X S (2a) and forwards it to the 802.1X AS (2b). RADIUS supports large domains that consist of many hierarchically organized RADIUS servers. Each identity is associated with a domain and known by the RADIUS server of that domain so that AA attempts can be forwarded in RADIUS infrastructures. In the third step (3), authentication is performed between the 802.1X S and 802.1X AS. The authenticator decapsulates EAP packets from EAPoL frames and reencapsulates them as EAPoRADIUS frames and vice versa. The flexible message structure of EAP allows the use of different authentication procedures. Simple approaches carry plain-text identity information or simple MD5-hashed passwords, but more secure authentication procedures like EAP Tunneled TLS [30] and EAP-TLS [31] are also supported. The authenticator only relays EAP messages in pass-through manner. Therefore, new EAP types only need to be implemented on the 802.1X S and 802.1X AS but not on the 802.1X A. In the fourth step, the RADIUS server may return authorization data after successful authentication to the 802.1X A (4a). It can be coarse-granular, e.g., a binary access decision whether the supplicant system gets access or no access, or fine-granular, e.g., VLAN tags [32] to be set for prospective user traffic or filter rules [33] that are applied by the authenticator. The authenticator applies the authorization data on the particular physical port of the switch, e.g., it sets a VLAN tag. Afterwards, the authenticator confirms successful AA to the supplicant with an EAP-Success message (4b).

C. *EAP-over-UDP (EAPoUDP)*

EAPoUDP is a variation of EAP that allows transmission of EAP data over UDP and IP. Figure 4 depicts its packet

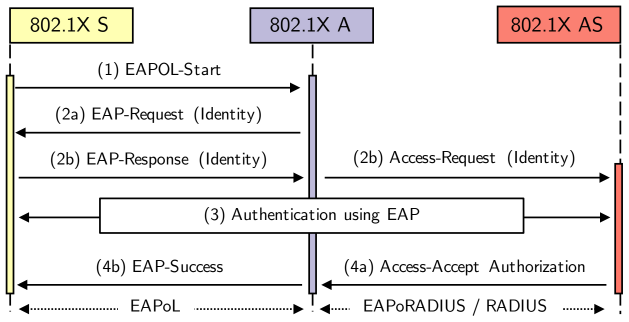

802.1X S
802.1X A
802.1X AS
(1) EAPOL-Start
(2a) EAP-Request (Identity)
(2b) EAP-Response (Identity)
(2b) Access-Request (Identity)
(3) Authentication using EAP
(4b) EAP-Success
(4a) Access-Accept Authorization
EAPoL
EAPoRADIUS / RADIUS

Fig. 3: Communication example of 802.1X based AA.

structure in comparison to EAPoL. In contrast to EAPoL, EAPoUDP can be used to authenticate multiple applications that run on a network host. Also, UDP packets can be transmitted over any link layer technology or even routed within multi-domain networks. EAPoUDP was introduced as Internet draft [34] that expired without standardization in the PANA working group of the IETF in 2002. Cisco leveraged EAPoUDP in its Trust Agent [35] tool that runs on network hosts and interacts with Cisco NAD, a prorietary network control system. The Trust Agent collects host system information, interfaces host software, and delivers notifications to network hosts within EAPoUDP frames. xRAC leverages EAPoUDP for frontend authentication.

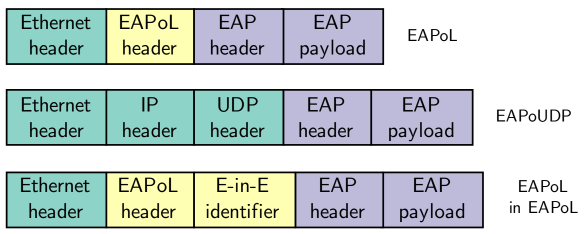


Fig. 4: EAPoUDP (a) in comparison to EAPoL (b). EAPoUDP uses UDP as transport protocol, EAPoL leverage Ethernet frames along an EAPoL header.

D. AA for Applications in Practice

802.1X focuses on port-based access control for network hosts. In practice, AA for applications is implemented as part of the application or with the help of Kerberos.

Most network applications implement some form of AA mechanism. Examples are login forms in the launch window where users are required to enter valid credentials to start using the application. Other examples are client certificates that are used in conjunction with TLS and a public key infrastructure. However, AA functionalities that are part of the application have an impact that is limited to the client and server side of the network application. Neither the start of the application nor the network infrastructure in between can be controlled.

Kerberos [36], [37] is a network authentication protocol that provides mutual authentication for clients and servers over an insecure network. Clients are entire hosts, users, or applications; servers represent hosts that offer particular network applications. Kerberos adapts user tickets for authentication for various network applications. Kerberos needs to be implemented by applications on client and server side, which prevents its application for legacy applications that cannot be modified. Again, neither the start of kerberized applications nor the network infrastructure in between can be controlled.

E. AA for Applications with FlowNAC

FlowNAC [38] introduces a fine-granular SDN network access control system that adapts 802.1X for AA of applications on network hosts. To enable multiple AA for different applications on a network host, the authors introduce EAPoL-over-EAPoLAN encapsulations. As depicted in Figure 4, FlowNAC introduces another variation of EAPoL. An EAPoL-in-EAPoL packet field identifies up to 64K different EAP processes that are transmitted as encapsulated EAP payloads. However, this deviation from legacy 802.1X requires major changes of the 802.1X S and 802.1X A. The 802.1X S is part of an OS's kernel, the 802.1X A is part of network switches so only open source OSs and firmwares allow modifications. Nevertheless, it is difficult to carry on the modifications in new versions of the OS's kernel or firmware image. The authors rely on EAPoL, i.e., AA data transfer is limited to the Ethernet link. EAPoUDP would solve those shortcomings but was not considered in the work. Unlike xRAC, FlowNAC neither introduces IP addresses for applications nor restricts the start of applications by AA.

IV. xRAC Architecture

In this section, we first explain RACs and give an overview of xRAC. Then, we explain the operation of its three control components in details.

A. Restricted Application Containers (RACs)

RACs are executable container images that enclose a single application, its dependencies, and configuration data such as program settings or software licensing information. As depicted in Figure 5, RACs are executed on a container runtime in parallel to OS-native applications. The CMD controls the execution of RACs and provides an interface for users to create, delete, start, or stop RACs. Each RAC has a unique IPv6 address so that its traffic can be easily identified in the network. RAC images are created by network administrators and deployed via RAC registries. They are either downloaded from those registries by users or automatically synchronized to managed hosts.

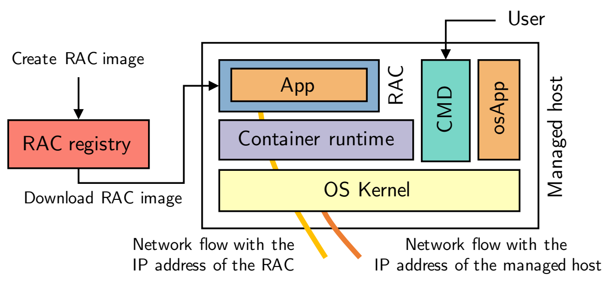


Fig. 5: The managed host executes a RAC and a OS application (osApp).

B. Overview of xRAC

xRAC provides execution and access control for RACs on managed hosts. A RAC needs to be authenticated and authorized before launching. Figure 6 depicts the AA process for RACs with 802.1X. First, a user attempts to start a RAC via the CMD (1), and the CMD requests the 802.1X CS for AA (2). After successful authentication (3), the 802.1X AS responds with authorization data via the 802.1X CA (4) to the 802.1X CS (4a). The 802.1X CS notifies the CMD to launch the RAC (4b). In addition, the 802.1X CA informs network control elements about the authorized RAC. In our example, it configures the firewall to permit access to the proteced server (4c). Other examples are SDN controllers that program SDN switches. Now, the authorized RAC but not the managed host or other RACs can communicate with the protected server (5).

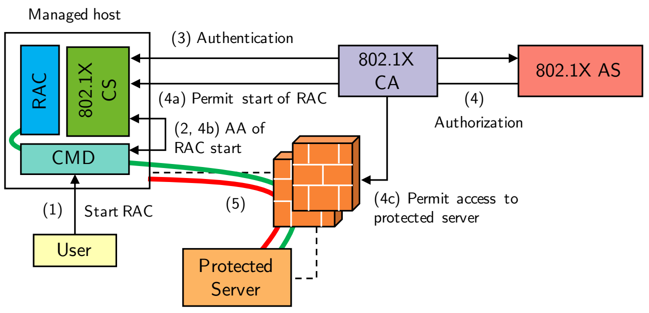


Fig. 6: Adaption of 802.1X for AA of RACs. The 802.1X CS authenticates the RAC using the 802.1X AS. The 802.1X CA receives authorization data and forwards them to the 802.1X CS and to a firewall that secures a protected server from unauthorized traffic.

AA for RACs introduces two additional advantages to common application deployment and network security. First, AA of RACs restricts RAC launches on managed hosts to predefined RAC images and permitted users. This allows network operators to ensure that only up-to-date and unmodified RAC images can be launched. This improves computer and network security as only valid RAC images can be executed on the managed hosts. In addition, network operators may deploy RAC images to managed hosts in advance, e.g., by synchronizing their set of RAC images with an internal RAC repository in the background. Users have all available RAC images on managed hosts but are only able to start them if they become authorized after AA. Last, each RAC has a globally unique IPv6 address that can be used to identify and steer all traffic originating from a particular RAC. RAC authorization data on the 802.1X AS includes information on how the RAC's traffic should be steered by network elements that can be applied by network control elements.

C. 802.1X Authentication Server (802.1X AS)

The authentication request from the 802.1X CS to the 802.1X AS contains user and container authentication data (UAND, CAND). The 802.1X AS authenticates the user and verifies the integrity of the RAC image. If the RAC image is valid, and if the user is authenticated, and if the user has permissions to run the RAC, the 802.1X AS responds to the 802.1X CA with authorization data. To perform that decision, the 802.1X AS requires a new data model which is depicted in Figure 7. It consists of user profiles, RAC profiles, and groups that define whether a particular user is permitted to run a particular RAC. User profiles contain user authentication data (UAND) that is used to authenticate the user. Examples are user credentials, e.g., user names and passwords. RAC profiles contain container authentication data (CAND) and container authorization data (CAZD). The first is used to verify the integrity of the RAC through calculating the cryptographic hashing function over the RAC image. CAZD include all permissions of a RAC, i.e., to be started by the requesting user and to utilize network resources. In the depicted example, the RAC is allowed to access a specified network resource. The AA data for the described model is stored on the 802.1X AS. The data model is an example that can be easily extended to support other requirements.

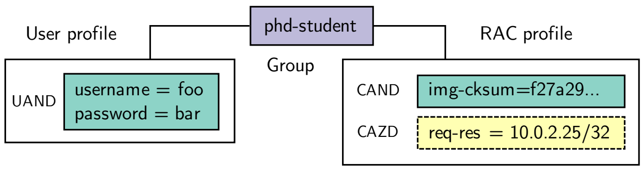


Fig. 7: The AA data model for RACs consists of user profiles, RAC profiles, and group mappings. User profiles include UAND, RAC profiles contain CAND for authentication and CAZD for authorization.

D. 802.1X Container Supplicant (802.1X CS)

The 802.1X CS authenticates RACs with the 802.1X AS via the 802.1X CA. It transmits UAND and CAND to the 802.1X AS and receives CAZD from the 802.1X CA.

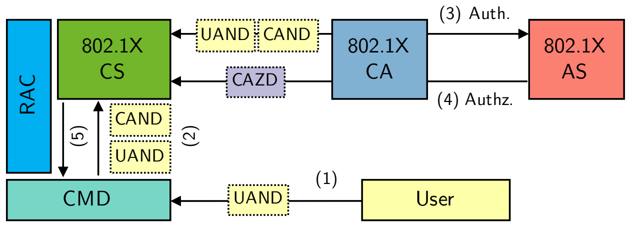
Figure 8a illustrates the process of AA from the perspective of the 802.1X CS. It runs on the managed host, interfaces the CMD, and is configured with the IP address or URL of the 802.1X CA so that it can initiate AA. In (1), the user requests the CMD to start a particular RAC on the managed host. The request includes UAND. The CMD requests the 802.1X CS to permit the user's demand (2). The request includes UAND and CAND collected by the CMD. The 802.1X CS initiates AA by establishing an EAPoUDP session with the 802.1X CA. Afterwards, it performs authentication with the 802.1X AS via the 802.1X CA (3). Backend authentication is performed via EAPoRADIUS while frontend authenticated is performed via EAPoUDP as in legacy 802.1X. In case of successful authentication, the 802.1X CS receives CAZD from the 802.1X CA (4). Then, the 802.1X CS permits the CMD to start the RAC (5).

E. 802.1X Container Authenticator (802.1X CA)

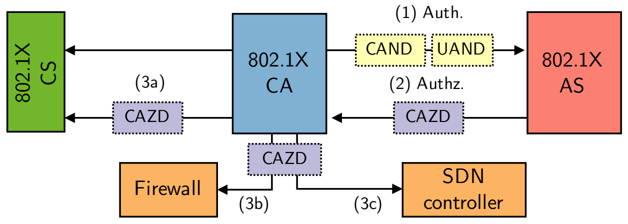
The 802.1X CA relays AA data between the 802.1X CS and the 802.1X AS. Moreover, it informs network control elements about authorized RACs.

In step (1) of Figure 8b, authentication data are transported over EAP between 802.1X CS and 802.1X AS. Between

802.1X CS and 802.1X CA, the EAP data are carried over UDP (EAPoUDP) and between 802.1X CA and 802.1X AS, they are carried over RADIUS. Thus, one task of the 802.1X CA is to modify the tunnel for EAP data. Moreover, after successful authentication the 802.1X AS returns CAZD over RADIUS to the 802.1X CA (2) which then informs the 802.1X CS about successful authorization (3a). While the conventional 802.1X A just opens ports on a switch for authorized devices, the 802.1X CA may also inform other network control elements about authorized RACs. Those may be ports on a switch, firewalls (3b), or SDN controllers (3c). The firewall is then programmed to pass through all outbound traffic with the RAC's IP address and the SDN controller instructs SDN switches to forward all traffic with the RAC's IP address appropriately. More specific flow descriptors are not needed.



(a) 802.1X CS receives UAND and CAND from the CMD. It then performs frontend authentication via EAPoUDP and backend authentication via EAPoRADIUS with the 802.1X AS. In case of successful authentication, the 802.1X CA and the 802.1X CS receive CAZD.



(b) The 802.1X CA performs frontend authentication with the 802.1X CS and backend authentication with the 802.1X AS. In case of successful authentication, it receives CAZD from the 802.1X AS that is forwarded to the 802.1X CS and other network components.

Fig. 8: AA from the perspective of the 802.1X CS and the 802.1X CA.

V. Use Cases and Discussion

We discuss two exemplary use cases of xRAC and discuss its benefits and limitations.

A. *Use Case I: Web Browsers in High-Security Areas*

Research departments, state departments, or clinics dealing with highly sensitive data isolate their internal networks from the Internet. However, web browsers are still required for online research activities. We propose to deploy web browsers as RACs on managed hosts. The isolation of RACs prevents that malicious users may misuse the Internet access to leak internals or contaminate the internal network through infectious downloads, e.g., PDF documents that include a trojan or virus. The network flow control of RACs ensures, that only the web browser can reach the Internet. If the RAC's traffic is encrypted, e.g., DNS queries and web site data, network control elements can still perform packet filtering based on the IP addresses of the RAC.

B. *Use Case II: Confidential Data Access*

Applications dealing with confidential data, e.g., research activities, medical documentation, or law enforcement, often access servers with confidential data. If such applications are deployed as RACs on managed hosts, only legitimate users have access to those servers. The isolation feature of RACs prevents remote hackers from attacking the server. Normally, they get system access through gateways provided by viruses or trojans received as browser downloads or e-mail attachments which is not possible with RACs. Furthermore, malicious users of legitimate applications may use hacker tools to gain access to the server and to leak information from it, which is not possible in the digital domain with an isolated application. The isolation of RACs prevent malicious users or applications from attacking the server. The network flow control ensures that the server can be reached only by legitimate RACs and users but not by other RACs or the managed host itself.

C. *Benefits of xRAC*

xRAC inherits the advantages generally known from virtualization and container virtualization that we have discussed in Section II-B. In addition, xRAC can guarantee that only valid containers are launched on managed hosts and that they can be used only by legitimate users. Thus, xRAC performs AA for applications without the need to modify them, which is a particular benefit for legacy applications. Moreover, the 802.1X CA can configure network control elements such that authorized RACs have access to protected network resources. RACs facilitate this control as all traffic of a RAC is identified by a single IPv6 address. This is a particular benefit as in today's networks there is no information about legitimate flows, many application flows may have the same IP address, and applications may even be invisible due to encryption using TLS. Thus, steering traffic from legitimate or trusted applications is a tough problem for which xRAC provides a solution. xRAC is flexible as it implements software-defined network control by interacting with other network control elements. In particular, it does not depend on and is not limited to specific technologies.

D. *Limitations of xRAC*

xRAC requires a managed infrastructure where the managed host, its CMD, and the 802.1X components, i.e., 802.1X CS, 802.1X CA, and 802.1X AS, are trusted. Encapsulating applications in RACs complicates access to shared resources so that xRAC may be cumbersome or infeasible for some use cases.

VI. Prototypical Implementation

We describe a prototypical implementation of xRAC and publish its source code with a testbed setup guide on GitHub [1]. In the following, we give an overview on the testbed environment and describe all components in detail.

A. Testbed Environment

Figure 9 depicts the testbed environment. The managed host executes RACs. The SDN switch connects the managed host, the protected server, and the public server and is controlled by an SDN controller. The SDN controller runs the 802.1X CA as SDN application that communicates with an 802.1X AS.

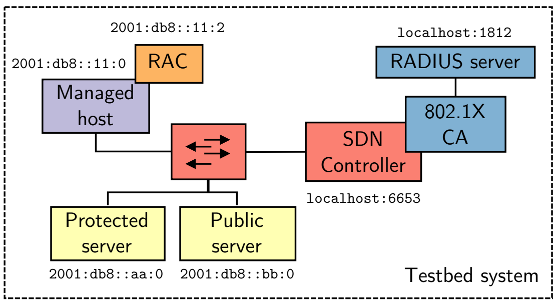


Fig. 9: Testbed environment.

We execute the testbed on a ThinkPad T460s with an i5-6200U CPU, 20GB RAM, SSD, and running Ubuntu 18.04.3 LTS. The managed host and both servers are Virtual Box virtual machines (VMs), each running Ubuntu 18.04.3 LTS. Open vSwitch [39] serves as SDN switch that is controlled by the Ryu SDN controller [40] (v4.34). The FreeRADIUS 802.1X AS (v3.0.16) is executed directly on the testbed host.

B. Docker as Container Virtualization Platform for RACs

We use Docker [2] (v19.03.5) as container virtualization platform to implement RACs. We configure the Docker CMD so that each RAC gets a dedicated IPv6 global unicast address that is reachable by other network hosts. Figure 10 depicts the applied networking configuration that follows the approach presented in [41]. By default, RACs only receive a link-local IPv6 address. Therefore, we set up a fixed IPv6 subnet with routable addresses for RACs. The managed host is configured with the IPv6 subnet `2001:db8::11:0/116` and the RACs receive an IPv6 address from that range. The first IPv6 address is reserved for the `docker0` interface. Therefore, the first RAC receives `2001:db8::11:2` and the second RAC `2001:db8::11:3`, respectively. The Docker daemon automatically adds routes to the routing table of the system and enables IPv6 forwarding so that all traffic to the IPv6 subnet will be routed via the `docker0` interface. To make the RACs reachable from other network hosts, we leverage the NDP proxy daemon [42]. It forwards L2 address resolution for IPv6 addresses of the RACs, i.e., it listens to neighbor solicitation requests for the RACs addresses and answers with the MAC address of the managed host. Afterwards, packets that address a RAC are received and forwarded through the Docker host via the `docker0` device to the particular RAC.

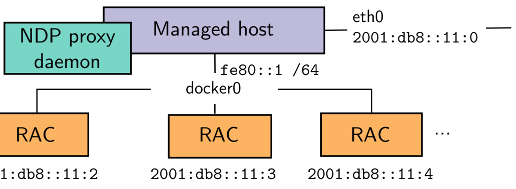


Fig. 10: Network configuration of Docker in the testbed environment. Each RAC gets an IPv6 address of the IPv6 subnet that is assigned to the managed host. The NDP proxy daemon resolves L2 addresses for the RACs.

C. 802.1X Container Supplicant (802.1X CS)

We implement the 802.1X CS as plugin for the Docker Authorization Framework introduced in Section II-D. We program the plugin in Python and leverage the Flask [43] library to implement its REST interface. Figure 11 depicts the authorization process. In (1), the user requests the CMD to start a container. The request includes UAND, e.g., a user name and a password. The Docker Authorization Framework predefines a two-step authorization process, but we only require the second step. The first authorization request (2) includes only minimal data, e.g., the name of the RAC image. As we solely rely on the second authorization step, the 802.1X CS responds with a permit by default. The second authorization request (3) includes UAND and CAND. The 802.1X CS performs authentication with the 802.1X AS through the 802.1X CA (3) as discussed before. In (4), 802.1X AS returns CAZD that is forwarded to the 802.1X CS in case of successful AA.

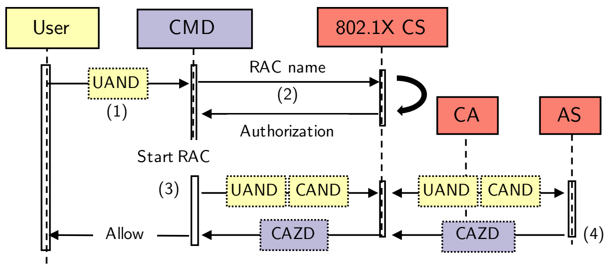


Fig. 11: Two-step authorization process in the Docker Authorization Framework [20]. The CMD requests the 802.1X CS to perform AA.

D. 802.1X Container Authenticator (802.1X CA)

We implement the 802.1X CA as SDN application for the Ryu SDN controller framework [40]. We extend the 802.1X A of [44] by adding support for authentication with the 802.1X CS using EAPoUDP. The 802.1X CA opens a UDP socket on port 5995 and waits for connections from the 802.1X CS. The 802.1X CA still may act as legacy 802.1X A. As example for network control with xRAC, we implemented a restricted MAC-learning switch. It learns MAC addresses from connected hosts but only forwards packets if the IP addresses of both sender and receiver are in a whitelist. The whitelist contains static entries, e.g., for public servers, and dynamic entries that can be modified by the 802.1X CA after receiving CAZD from the 802.1X AS. We implement the restricted MAC-learning switch by extending the L2 switch [45] from the Ryu SDN controller framework.

E. 802.1X Authentication Server (802.1X AS)

We leverage the widely-used 802.1X AS software FreeRADIUS and extend its AA data model to implement CAND and CAZD. In FreeRADIUS, additional attributes for AA can be implemented using vendor-specific attributes (VSAs) [28], [46]–[48]. Simple policies are defined with the unlang [49] processing language. The defined AA data model can be easily extended and modified by adding more VSAs.

F. Protected and Public Server

We run a public server with the static IPv6 address `2001:db8::aa:0` that is accessible without authorization. As example for a protected network host, we run a protected server with the static IPv6 address `2001:db8::bb:0`.

VII. Experimental Validation

We describe the experiment setup and validation experiments for the testbed from Section VI to validate xRAC.

A. Experiment Setup

The experiments investigate the communication between the managed host, a particular RAC, the protected server, and the public server. We use the latest Busybox [50] image as RAC. In our experiments, we use the RAC to send ICMP echo request packets to both, the protected and the public server. We add UAND, CAND and CAZD on the 802.1X AS that allows a particular user to run the RAC and access the protected server.

B. Validation Experiments

We perform the following experiments as depicted in Figure 12. Before launching the RAC, we validate with ICMP echo requests from the managed host that the public server (1a) but not the protected server (1b) is accessible without authorization. Now, we demonstrate that the integrity of RACs is verified during authentication, i.e., that a RAC with a divergent image checksum cannot be started. We start an Alpine Linux [51] container image as RAC and try to start it using the user credentials as set up on the RADIUS server. Authentication fails, i.e., the RAC cannot be started on the managed host. Now, we demonstrate that the correct RAC can be started and that it can access the protected server after successful AA. After issuing the command to start the RAC, it is authenticated and authorized as described before (2a). The SDN controller receives CAZD and programs the SDN switch to permit packet forwarding between the RAC and the protected server (2c). Now, the RAC is able to exchange ICMP packets with the protected server (2d). Trying to exchange ICMP packets directly from the managed host fails (2e), i.e., the protected server can be reached by the RAC but not by the managed host.

VIII. Performance Considerations

Container virtualization has no remarkable performance overhead compared to native application execution [52]. xRAC adds delay only to the startup time through the CMD. The CMD requires time to calculate the container's integrity checksum and to perform AA using network-based 802.1X. Without xRAC, starting the wget container from Section VII-A takes approximately 0.6 s. With xRAC, the startup time is increased to 1.51 s. The Busybox container image is 6 MB large and the calculation of its SHA256 hash takes 0.12 s on our platform. However, the computation time scales with the image size. For example, a Chrome browser container image with 880 MB takes 3.2 s. The duration of the AA operation depends on the performance of the three 802.1X components and the network in between. The container supplicant is part of the managed host and, therefore, covers only little load. The container authenticator is part of the controller and responsible for many hosts. However, its performance can be scaled up by running multiple instances. The 802.1X AS may be based on RADIUS. This technology is proven to scale well with large deployments and high load by replicating server instances.

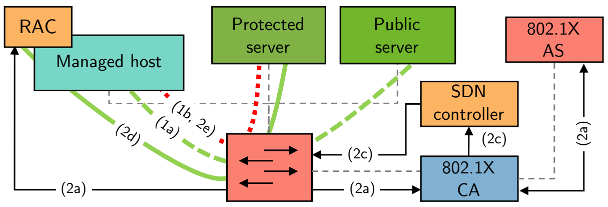


Fig. 12: Experiments to investigate the communication between the managed host, a RAC, and both servers.

IX. Conclusion

In this work we proposed xRAC, a concept for execution and access control for restricted application containers (RACs) on managed clients. It includes authentication and authorization (AA) for RACs such that only up-to-date RAC images can be executed by permitted users. Moreover, authorization is extended to protected network resources such that authorized RACs can access them. Traffic control is simplified through the fact that all traffic of a RAC is identified by its IPv6 address. We presented the architecture of xRAC and showed by a prototype implementation that xRAC can be built from standard technologies, protocols, and infrastructure. Our prototype of xRAC leverages Docker as container virtualization platform, signalling is based on 802.1X components. Modifications were needed to the supplicant, the authenticator, and the authentication server so that both user and container AA data can be exchanged. Moreover, the container authenticator is extended to inform required network control elements about authorized RACs. We used the prototype to experimentally validate xRAC and investigate on the performance. After all, we discussed use cases and showed that xRAC supports software-defined network control and improves network security without modifying core parts of applications, hosts, and infrastructure.

Acknowledgment

The authors thank Julian Rilli for fruitful discussions and programming contributions.

References

[1] "xRAC Repository on GitHub," https://github.com/uni-tue-kn/xrac.

[2] "Docker," https://www.docker.com/, accessed 01-21-2020.

[3] "Kubernetes," https://kubernetes.io/, accessed 01-21-2020.

[4] "FreeBSD: jails," https://www.freebsd.org/doc/handbook/jails.html, accessed 01-21-2020.

[5] "Oracle Solaris Containers," https://www.oracle.com/technetwork/server-storage/solaris/containers-169727.html, accessed 01-21-2020.

[6] "Docker Hub," https://hub.docker.com/, accessed 01-21-2020.

[7] "Docker Docs," https://docs.docker.com, accessed 01-21-2020.

[8] "cgroups," http://man7.org/linux/man-pages/man7/cgroups.7.html, accessed 01-21-2020.

[9] "LWN.net: PID namespaces in the 2.6.24 kernel," http://lwn.net/Articles/259217/, accessed 01-21-2020.

[10] "LWN.net: Network namespaces," http://lwn.net/Articles/219794/, accessed 01-21-2020.

[11] "AuFS4," http://aufs.sourceforge.net/, accessed 01-21-2020.

[12] "OverlayFS," https://www.kernel.org/doc/Documentation/filesystems/overlayfs.txt, accessed 01-21-2020.

[13] "OpenZFS," http://www.open-zfs.org, accessed 01-21-2020.

[14] "Open Container Initiative," https://www.opencontainers.org/, accessed 01-21-2020.

[15] "Twistlock," https://www.twistlock.com/, accessed 01-21-2020.

[16] "Twistlock 18.11 Data Sheet," https://www.twistlock.com/twistlock-data-sheet/, accessed 01-21-20.

[17] "Aqua Container Security Platform," https://www.aquasec.com/products/aqua-container-security-platform/, accessed 06-03-2018.

[18] "Sysdig Secure," https://www.sysdig.com, accessed 06-03-2018.

[19] "Atomicorp Atomic Secured Docker," https://atomicorp.com/features-atomic-secured-docker-kernel/, accessed 06-03-2018.

[20] "Docker Access Authorization Plugin," https://docs.docker.com/engine/extend/plugins_authorization/, accessed 01-21-2020.

[21] "Docker AuthZ Plugins: Twistlock's Contribution to Docker Security," https://www.twistlock.com/2016/02/18/docker-authz-plugins/, accessed 01-21-2020.

[22] "Jessie Frazelle: Docker Containers on the Desktop," https://blog.jessfraz.com/post/docker-containers-on-the-desktop/, accessed 01-21-2020.

[23] IEEE, "802.1X-2001 IEEE Standard for Local and Metropolitan Area Networks - Port-Based Network Access Control," 2001.

[24] ——, "802.1X-2004 IEEE Standard for Local and Metropolitan Area Networks - Port-Based Network Access Control," 2004.

[25] ——, "802.1X-2010 IEEE Standard for Local and Metropolitan Area Networks - Port-Based Network Access Control," 2010.

[26] "Eduroam - About," https://www.eduroam.org/index.php?p=about, accessed 07-19-2018.

[27] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz, "Extensible Authentication Protocol (EAP)," RFC 3748 (Proposed Standard), 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3748.txt

[28] C. Rigney, S. Willens, A. Rubens, and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)," RFC 2865 (Draft Standard), jun 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2865.txt

[29] V. Fajardo (Ed.), J. Arkko, J. Loughney, and G. Zorn (Ed.), "Diameter Base Protocol," RFC 6733 (Proposed Standard), RFC Editor, Fremont, CA, USA, pp. 1–152, Oct. 2012, updated by RFC 7075. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6733.txt

[30] P. Funk and S. Blake-Wilson, "Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0)," RFC 5281 (Informational), RFC Editor, Fremont, CA, USA, pp. 1–51, Aug. 2008. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5281.txt

[31] D. Simon, B. Aboba, and R. Hurst, "The EAP-TLS Authentication Protocol," RFC 5216 (Proposed Standard), RFC Editor, Fremont, CA, USA, pp. 1–34, Mar. 2008. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5216.txt

[32] P. Congdon, M. Sanchez, and B. Aboba, "RADIUS Attributes for Virtual LAN and Priority Support," RFC 4675 (Proposed Standard), RFC Editor, Fremont, CA, USA, pp. 1–15, Sep. 2006. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4675.txt

[33] ——, "RADIUS Filter Rule Attribute," RFC 4849 (Proposed Standard), RFC Editor, Fremont, CA, USA, pp. 1–9, Apr. 2007. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4849.txt

[34] P. Engelstad, "EAP over UDP (EAPoUDP)," Internet Engineering Task Force, Internet-Draft draft-engelstad-pana-eap-over-udp-00, Feb. 2002, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-engelstad-pana-eap-over-udp-00

[35] "Cisco Trust Agent," https://community.cisco.com/t5/security-documents/cisco-trust-agent/ta-p/3113750, accessed 01-21-2020.

[36] "Kerberos: The Network Authentication Protocol," http://web.mit.edu/kerberos/, accessed 01-21-2020.

[37] C. Neuman, T. Yu, S. Hartman, and K. Raeburn, "The Kerberos Network Authentication Service (V5)," RFC 4120 (Proposed Standard), Internet Engineering Task Force, Jul. 2005.

[38] J. Matias, J. Garay, A. Mendiola, N. Toledo, and E. Jacob, "FlowNAC: Flow-based network access control," in *Third European Workshop on Software Defined Networks*, 2014, p. 6.

[39] "Open vSwitch," https://www.openvswitch.org/, accessed 01-21-2020.

[40] "Ryu SDN Framework," http://osrg.github.io/ryu/, accessed 01-21-2020.

[41] "Docker Docs: IPv6 with Docker," https://docs.docker.com/v17.09/engine/userguide/networking/default_network/ipv6/, accessed 01-21-2020.

[42] "NDP Proxy Daemon," https://github.com/DanielAdolfsson/ndppd, accessed 01-21-2020.

[43] "Flask," https://palletsprojects.com/p/flask/, accessed 01-21-2020.

[44] F. Hauser, M. Schmidt, and M. Menth, "Establishing a Session Database for SDN using 802.1X and Multiple Authentication Resources," in *2017 IEEE International Conference on Communications (ICC)*, May 2017, pp. 1–7.

[45] "Examplary MAC-learning Switch for OpenFlow 1.3," https://github.com/osrg/ryu/blob/master/ryu/app/example_switch_13.py, accessed 07-19-2018.

[46] A. DeKok (Ed.) and G. Weber, "RADIUS Design Guidelines," RFC 6158 (Best Current Practice), RFC Editor, Fremont, CA, USA, pp. 1–38, Mar. 2011, updated by RFCs 6929, 8044. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6158.txt

[47] A. DeKok and A. Lior, "Remote Authentication Dial In User Service (RADIUS) Protocol Extensions," RFC 6929 (Proposed Standard), RFC Editor, Fremont, CA, USA, pp. 1–68, Apr. 2013. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6929.txt

[48] A. DeKok, "Data Types in RADIUS," RFC 8044 (Proposed Standard), RFC Editor, Fremont, CA, USA, pp. 1–35, Jan. 2017. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8044.txt

[49] "FreeRADIUS man pages: unlang," https://freeradius.org/radiusd/man/unlang.html, accessed 01-21-2020.

[50] "busybox Docker Image," https://hub.docker.com/_/busybox, accessed 01-21-2020.

[51] "Alpine Linux Docker Image," https://hub.docker.com/_/alpine, accessed 01-21-2020.

[52] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and Linux containers," in *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, March 2015, pp. 171–172.

1.4 Demo: Execution and Access Control for Restricted Application Containers on Managed Hosts (xRAC)

Demo: Execution and Access Control for Restricted Application Containers on Managed Hosts (xRAC)

Frederik Hauser and Michael Menth
Chair of Communication Networks, University of Tuebingen, Tuebingen, Germany
Email: {frederik.hauser,menth}@uni-tuebingen.de

***Abstract*—Restricted application containers (RACs) encapsulate applications with their dependencies and configuration for execution on a hypervisor host. xRAC [1] is a novel approach for execution control and network access control (NAC). That is, a RAC can be executed only after successful authentication and authorization (AA) and obtain limited access to network resources. A RAC has a unique IPv6 address so that its traffic is identifiable and controllable by network components. For AA, xRAC adopts and extends components and procedures of 802.1X. We publish its source code and a testbed setup guide on GitHub [2]. In this paper, we give a brief overview on the architecture and functionality of xRAC, describe the prototypical implementation, a testbed, and four demo scenarios.**

***Index Terms*—Network Access Control, Application Execution Control, 802.1X, SDN**

I. Overview on xRAC

In today's networks, traffic is increasingly encrypted so that traffic control becomes hard, e.g., for Quality of Service (QoS) or security purposes, since the content of the traffic is unclear. xRAC [1] tackles this problem. Applications are run in restricted application containers (RACs) on managed hosts and are executed only after successful AA. As each RAC has its own IPv6 address, the traffic of such applications can be easily identified and appropriately treated.

With xRAC, users are assigned permissions to run special RACs on managed hosts. When a user starts a RAC on a managed host, an AA server is contacted to authenticate the user and grant authorization to launch the RAC. Only in case of success, the RAC is executed. Managed hosts may run multiple RACs in parallel to host-native applications.

For authentication, the managed host sends user authentication data (UAND) and container authentication data (CAND) to the AA server. UAND may be a user identity with a password and CAND may be the integrity checksum of the RAC. If the user is permitted to run the RAC, the AA server returns container authorization data (CAZD) to the managed host and network control elements. The former permits execution of the RAC, the latter grants access to protected network resources.

For applying this AA procedure with 802.1X on RACs, we adopt the three 802.1X components with the following modifications to the original standard. First, we introduce a softwarized 802.1X container supplicant (802.1X CS) that runs on the managed host with an interface to the container management daemon (CMD). Second, we introduce a softwarized 802.1X container authenticator (802.1X CA) that is deployed as network application connecting the 802.1X CS and 802.1X authentication server (802.1X AS). Third, we substitute EAP-over-LAN (EAPoL) by EAP-over-UDP (EAPoUDP) for the communication between the 802.1X CS and 802.1X CA. Last, we configure the 802.1X AS to support CAND validation and CAZD. More technical details can be found in the paper [1].

Figure 1 depicts AA of RACs with 802.1X. A user attempts to start a RAC via the CMD (1). The start request includes the RAC name and UAND. The CMD calculates the image checksum as CAND and requests the 802.1X CS for permission to run the RAC (2). The 802.1X CS initiates and performs authentication with the 802.1X AS via the 802.1X CA (3). In case of successful authentication and execution permission, the 802.1X AS responds with CAZD (4) to the 802.1X CA. The 802.1X CA forwards CAZD to the 802.1X CS (4a), the 802.1X CS permits the RAC launch permit of the CMD (4b). In parallel, the 802.1X CA forwards CAZD to network control elements (4c). Here, it instructs an SDN controller to install rules so that the RAC can communicate with the protected server. Now, the RAC but not the managed host or other RACs can communicate with the protected server (5).

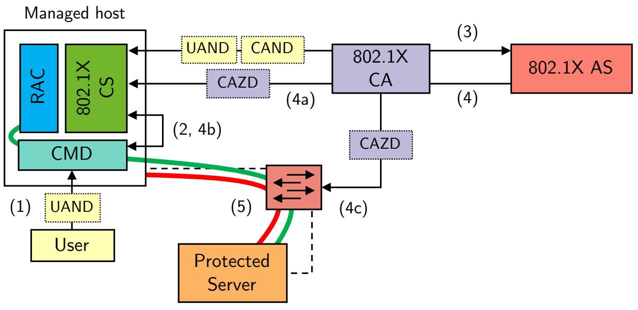


Fig. 1. Components and process of AA in xRAC (similar to [1]).

II. Prototypical Implementation

The prototypical implementation of xRAC comprises three parts. First, the managed host runs RACs with the help of the CMD and 802.1X CS. We use Docker (19.03.5) as virtualization platform, enable IPv6 networking, and create a fixed subnet with globally routeable IPv6 addresses in the CMD configuration. Each RAC receives a dedicated IPv6

This work was supported by the bwNET100G+ project which is funded by the Ministry of Science, Research and the Arts Baden-Württemberg (MWK). The authors alone are responsible for the content of this paper.

global unicast address, the CMD manages the routing table of the host system and enables IPv6 forwarding so that other hosts can reach the RACs. We run the NDP Proxy Daemon (NDPPD) [3] that responds to neighbor solicitation request for RAC addresses with the MAC address of the managed host. We implement the 802.1X CS as plugin for the Docker Authorization (AuthZ) framework [4]. The AuthZ framework provides a REST API that allows individual authorization plugins to approve or deny requests to the CMD. We implement the 802.1X CS as Flask [5] web application and run it with with the uWSGI [6] application server. Second, the 802.1X CA is implemented as application for the Ryu SDN controller framework [7] (v4.34). Therefore, we extend the 802.1X Authenticator from our previous work [8] by AA for RACs using EAPoUDP. As example for network access control (NAC), we extend a L2 switch by access control lists. Static white-list entries that allow communication with public hosts are defined by the administrator. Dynamic white-list entries are added by the 802.1X CA when receiving CAZD from the 802.1X AS. Last, we use FreeRADIUS (v3.0.16) as 802.1X AS. We extend its data model with vendor-specific attributes (VSAs) for CAND/CAZD and add unlang [9] control sequences to validate CAND within AA.

III. Testbed Environment

Figure 2 depicts the testbed environment. We execute Virtual Box virtual machines (VMs), an Open vSwitch (OVS) instance, the Ryu SDN controller, and a FreeRADIUS instance on a ThinkPad T460s with an i5-6200U CPU, 20 GB RAM, SSD, and running Ubuntu 18.04.3 LTS. The managed host is deployed as VM running Ubuntu Server 18.04.3 LTS. The managed host VM is configured with the IPv6 subnet `2001:db8::11:0/116`. The `docker0` interfaces receives `2001:db8::11:1`, xRAC containers receive IPv6 addresses starting from `2001:db8::11:2`. Both public and protected servers are VMs running Ubuntu Server 18.04.3 LTS. The public server is configured with the IPv6 address `2001:db8::aa:0`, the protected server with `201:db8::bb:0`. All VMs are interconnected via an OVS bridge with a tap device. The Ryu SDN controller with the 802.1X CA application manages the OVS bridge via OpenFlow. OVS, the Ryu SDN controller, and FreeRADIUS are directly executed on the local testbed system. A GitHub repository [2] includes a complete setup guide and the source code of all software components.

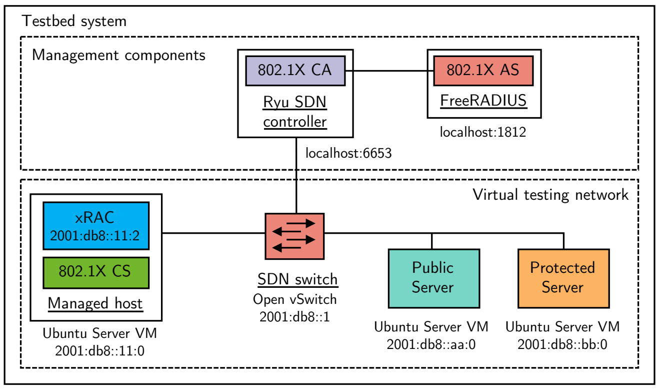


Fig. 2. Testbed environment with xRAC components.

IV. Demo Scenario

We describe the experiments shown in the demonstration. For visualizing xRAC's operations, we open four console windows on the testbed computer that show the CLI of the managed host, the console output of the 802.1X CS application, the console output of the 802.1X CA application, and the console output of the 802.1X AS.

We perform the following four experiments from the managed host that should demonstrate xRAC's functionality. First, we show that only the public, but not the private server are reachable before AA. We send ICMP echo request packets to both servers but only receive ICMP echo response packets from the public server. Second, we show that the latest Busybox [10] container image as example for a RAC can successfully access the protected server after AA. We define UAND (username and password), CAND (checksum of the Busybox image), and CAZD (IPv6 address of the protected server) on the 802.1X AS. We start a new Busybox RAC with the user credentials as UAND. With successful AA, the 802.1X CA adds a dynamic white-list entry so that the RAC can communicate with the protected server. RAC start is granted, and we show successful AA by sending ICMP echo request packets from the RAC to the protected server that are answered with ICMP echo response packets. Third, we show that only the RAC, but not the managed host can access the protected server after AA. While sending ICMP echo requests from the RAC, we start to send ICMP echo requests from the managed host to the protected server. The RAC still receives ICMP echo response packets, but no answers arrive at the managed host. Last, we show that only permitted RACs can be started on the managed host. Instead of Busybox, we try to start an Alpine Linux [11] container image. As expected, execution on the managed host is denied by the 802.1X CS.

Acknowledgment

The authors thank Mark Schmidt and Julian Rilli for fruitful discussions and programming contributions.

References

[1] F. Hauser, M. Schmidt, and M. Menth, "xRAC: Execution and Access Control for Restricted Application Containers on Managed Hosts," in *IEEE/IFIP Network Operations and Management Symposium 2020*.
[2] "xRAC Repository on GitHub," https://github.com/uni-tue-kn/xrac.
[3] "ndppd," https://github.com/DanielAdolfsson/ndppd.
[4] "Docker AuthZ," https://docs.docker.com/engine/extend/plugins_authorization/.
[5] "Flask," https://palletsprojects.com/p/flask/.
[6] "uWSGI," https://uwsgi-docs.readthedocs.io/en/latest/.
[7] "Ryu SDN Controller Framework," https://osrg.github.io/ryu/.
[8] F. Hauser, M. Schmidt, and M. Menth, "Establishing a Session Database for SDN using 802.1X and Multiple Authentication Resources," in *IEEE International Conference on Communications 2017*.
[9] "unlang," https://freeradius.org/radiusd/man/unlang.html.
[10] "busybox Docker Image," https://hub.docker.com/_/busybox.
[11] "Alpine Linux Docker Image," https://hub.docker.com/_/alpine.
All web resources were last accessed on 01-12-2020.

1.5 Establishing a session database for SDN using 802.1X and multiple authentication resources

Establishing a Session Database for SDN Using 802.1X and Multiple Authentication Resources

Frederik Hauser, Mark Schmidt, Michael Menth
Chair of Communication Networks, University of Tuebingen, Tuebingen, Germany

Abstract—**Network control systems based on identities allow fine-grained access control for users. They require a network-wide session database containing information about active authenticated and authorized users. We propose an authentication and authorization (AA) module (AAM) as a controller application for software-defined networking to establish a network-wide session database and provide a prototypical implementation with OpenFlow. End systems issue authentication requests and the switch redirects them to the AAM. The AAM either relays them to a RADIUS server as in legacy 802.1X (pass-through mode) or processes them based on directly attached AA resources (authentication server mode). After successful authentication, the AAM authorizes the requesting user and maintains a network-wide session database of authenticated and authorized identities. As the AAM interfaces to end systems and AA resources through existing protocols, i.e., EAP and RADIUS, its use is compatible with current infrastructures. Through implementation as distributed network functions, the AAM can be scaled so that high rates of authentication requests can be supported.**

I. Introduction

Securing networks by introducing authentication and authorization is a major goal in network security. Especially large-scale networks with thousands of users require sophisticated network access control. Infrastructures for deploying authentication, authorization, and accounting (AAA) provide the technical foundations that have been used for many years. So far, authorization mostly supports coarse-granular access permissions or identity-based VLAN tagging. However, today's demands for secure network admission control go beyond that.

The rise of software-defined networking (SDN) led to extensive work on fine-granular network control systems that are based on user identities. Approaches like Ethane [1], Resonance [2] or Kinetics [3] allow network control with abstract identity-centric rules and stateful network control actions. This concept is called identity-based security [4] and has the potential to improve and simplify security solutions for large-scale networks with different security levels and user-specific access rights.

Network control systems based on user identities require network-wide session databases and reliable authentication mechanisms. The latter should be compatible to existing AAA infrastructures and end systems. Today, the majority of approaches for network access control applications in OpenFlow-based SDN suggest authentication using web frontends or MAC address mappings. Both approaches reveal shortcomings regarding compatibility, usability, and security that are pointed out in Section III. 802.1X is the most widely applied method for authentication and authorization (AA) in networks today. Some research works have adopted 802.1X for SDN but without leveraging the increased flexibility offered by SDN. In particular, there is no common best practice to perform AA using 802.1X for SDN.

We suggest an authentication and authorization module (AAM) as controller application to maintain a network-wide session database which interfaces with 802.1X to end systems and with other protocols, e.g., RADIUS, to authentication resources. The AAM can be easily used with existing end systems implementing 802.1X but does not suffer from the limitations of current 802.1X deployments. In particular, the AAM maintains a session database as needed for fine-granular network access control and can also leverage other authentication resources than RADIUS, e.g., user information in local databases.

The remainder of this paper is structured as follows. Section II shortly reviews key aspects of 802.1X. Section III discusses related work. In Section IV, we describe the AAM in detail. Section V describes its prototypical implementation and experimental evaluation giving a proof-of-concept. Section VI discusses how network function virtualization may be used to scale the AAM to large request rates. Finally, Section VII summarizes this work and draws conclusions.

II. Authentication and Authorization using 802.1X

In the following, we give an overview of the 802.1X architecture, describe the supporting protocols EAP and RADIUS, illustrate the operation of 802.1X, and discuss limitations of current deployments.

A. 802.1X Architecture

IEEE 802.1X [5]–[7] describes port-based network admission control in Ethernet networks. Although originally introduced for wireline networks, 802.1X is mainly known from wireless 802.11 networks today. A prominent application of 802.1X is eduroam [8], a federation of wireless university campus networks worldwide which allows participants to connect to the Internet in foreign institutions.

Figure 1 shows the architecture of 802.1X which adopts the components of the abstract AAA architecture in [9] with a different nomenclature. A network host is called supplicant system and contains a supplicant module. A LAN edge switch controlling the access of network hosts to the network is called

This work was supported by the bwNET100G+ project which is funded by the Ministry of Science, Research and the Arts Baden-Wuerttemberg (MWK). The authors alone are responsible for the content of this paper.

authenticator system and contains an authenticator module. The AAA server is called authentication server system and contains the authentication server. It is responsible for executing the actual authentication and provision of authorization information, and is triggered by the authenticator.

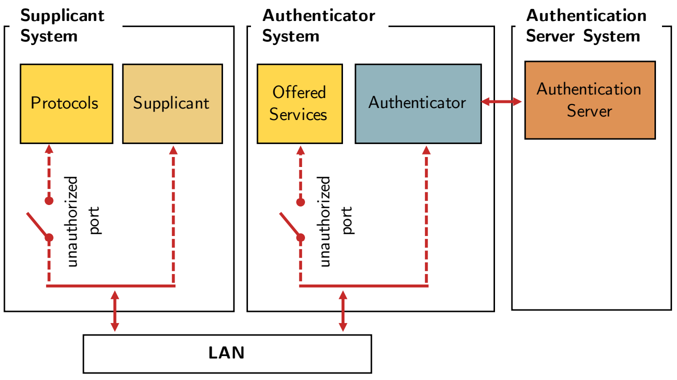


Fig. 1: Port-based authorization model of 802.1X according to [6].

The ports within the supplicant and authenticator system can be considered as abstract port entities. Without successful authentication and authorization, the supplicant system can reach only the authenticator module on an authenticator system. A 802.1X AA procedure is always initiated by the supplicant sending a start message to the authenticator. After successful authentication and authorization the unauthorized ports become authorized. Authorization in this context can be coarse- or fine-granular. The authentication server may inform the authenticator with a binary information whether access should be granted, or it may also provide a specific VLAN tag [10] for prospective user traffic.

Figure 2 shows that 802.1X encompasses both frontend and backend AA. Frontend authentication between the supplicant and authenticator modules is defined by the Extensible Authentication Protocol (EAP) [11]. Backend AA between the authenticator module and the authentication server can be performed by the Remote Authentication Dial In User Service (RADIUS) [12] or the Diameter protocol [13].

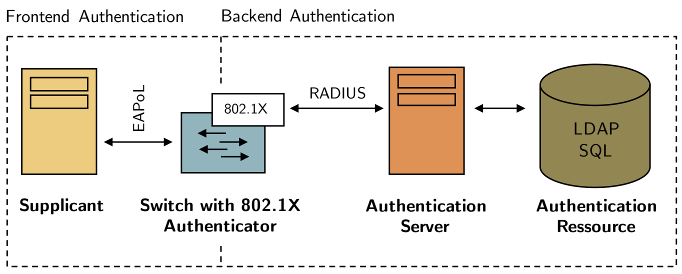


Fig. 2: Interaction of components in the classic 802.1X architecture.

B. *EAP and RADIUS in 802.1X*

In the following, we introduce EAP and RADIUS in the context of 802.1X by looking at the examplary AA process depicted in Figure 3. We solely focus on RADIUS because it is the most-widely used protocol for backend AA.

1) Initialization of AA in 802.1X: The supplicant module on the network client initiates AA by sending an EAPOL-Start message. EAPoL is an EAP-over-LAN encapsulation to transport EAP messages within Ethernet frames that was introduced with 802.1X. EAP facilitates communication between supplicant and authenticator, and it provides a fixed request and response scheme to exchange authentication data between supplicant and authentication server.

2) Identity-based AA in 802.1X: The authenticator requests the client's identity and forwards it to the RADIUS authentication server. RADIUS supports large domains that consist of a large number of hierarchically organized RADIUS servers. Each identity (e.g. an user) is associated with a domain and known by the RADIUS server of that domain. Therefore, the identity is the most relevant information for routing AA attempts within RADIUS infrastructures. This principle is used, e.g., in eduroam which allows users to leverage the wireless university campus network infrastructure on foreign campuses or special venues like IETF meetings and conferences.

3) Authentication in 802.1X: Authentication is performed between supplicant and authentication server. The authenticator decapsulates EAP packets from EAPoL frames and reencapsulates them in RADIUS frames and vice versa. The flexible message structure of EAP allows the use of different authentication procedures. Simple approaches carry plain-text identity information or simple MD5-hashed passwords, but more secure authentication procedures like IKEv2 for EAP [14], EAP Tunneled TLS [15], and EAP-TLS [16] are also supported. As the authenticator only relays EAP messages in pass-through manner, it does not need to implement any EAP type specifics.

4) Authorization in 802.1X: After successful authentication, the RADIUS server may return authorization data, e.g., a VLAN tag. The authenticator applies the authorization data on the particular physical port on the switch, e.g., it sets a VLAN tag. Afterwards, the authenticator confirms successful AA to the supplicant with an EAP-Success message.

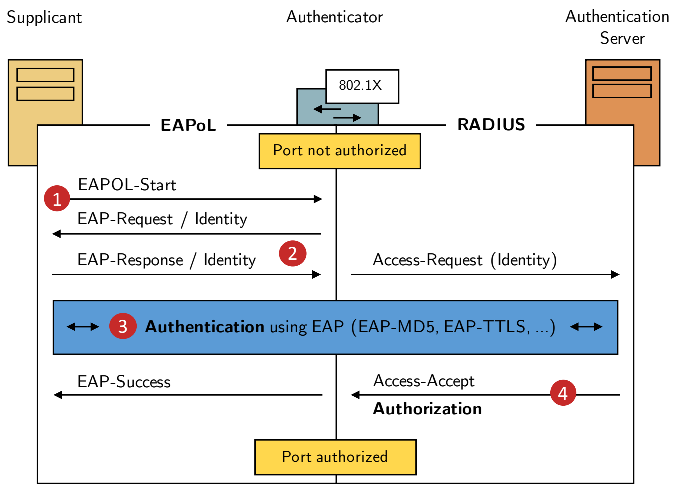


Fig. 3: Communication example of 802.1X based authentication and authorization (AA).

C. *Limitations of 802.1X*

In [17], client-side security concerns of 802.1X are pointed out. Here, we discuss limitations regarding flexibility with respect to the infrastructure side.

1) Dependence on RADIUS or Diameter: 802.1X requires a RADIUS or Diameter server for backend AA. In most cases, RADIUS servers use AA data stored on external resources like an Lightweight Directory Access Protocol (LDAP) server or an SQL database. The use of RADIUS is an advantage if user data needs to be accessed from a foreign domain, but for exclusively local applications this is unnecessary overhead because RADIUS is an additional service which requires configuration and administration effort. Direct interaction between an authenticator and an AA resource is more lightweight and may be used in parallel to other RADIUS resources. Another aspect is formal administration overhead. Official RADIUS infrastructures are managed by central computation centers and adding users is a major administration process which may be desirable to avoid for separate experimental infrastructure or student labs.

2) Change of Authorization: Backend authentication in 802.1X does not support unsolicited messages from an authentication server to an authenticator. As a consequence, changes in AA data, e.g., revocation of a user's permission to access the network, cannot be applied to existing sessions. Due to this limitation, dynamic authorization extensions [18] to RADIUS have been defined but they are are supported by only a few authenticator implementations.

3) Stateless Property of RADIUS: Today, an unlimited number of concurrent authorized network accesses may be initiated with 802.1X using the credentials of the same identity. This is due to the stateless property of the system, i.e., authenticator and authentication servers do not keep records of currently authorized user sessions. Due to this shortcoming, the *Simultaneous-Use* extension [19] was defined to limit the number of concurrent sessions and to introduce some session context on the RADIUS server. The *radutmp* module [20] for FreeRADIUS implements this extension but is hardly deployed. Active sessions are tracked by little standardized RADIUS accounting messages or through the use of SNMP, Finger, and telnet which is not a reliable solution to the problem.

III. Related Work

We give an overview of network control systems based on user identities and report state of the art for authentication in SDN.

A. *Network Control Systems Based on User Identities*

Ethane [1] introduced abstract rules based on user identities, host classes, and protocols for the definition of abstract rules for network access. Resonance [2] extends this rather stateless view by introducing state graphs for network hosts. Their states change in the network control system, e.g., when a network security scanner detects a malware infection on an authorized host. Kinetics [3] introduces a domain-specific language for the definition of fine-granular network control rules. In addition, it allows formal verification of these rules. Ravel [21] focuses on plain network rule representation in an SQL database and heavily uses view abstractions. All these approaches require a session database, but none of them addresses how to build and maintain it using common AA mechanisms.

B. *Authentication and Authorization (AA) in SDN*

MAC address mapping and web frontends are most widely used for AA in SDN. Moreover, some research prototypes adopt 802.1X in various forms to provide the same service as today, but they do not maintain a session database.

1) Identification and Authorization Using MAC Addresses: In this approach, MAC addresses are either used as identities or they are mapped to identities. The identities are used to fetch authorization data from an AA resource, e.g., a RADIUS or an LDAP server. In contrast to authentication, this process of identification does not verify the identity claimed by the network client. MAC addresses are unique but not confidential, especially network devices like printers often have a printed label revealing their MAC addresses. Moreover, MAC addresses are easy to eavesdrop. Even a novice attacker can spoof MAC addresses and, therefore, easily impersonate network hosts to obtain access to the network.

2) AA Using Web Frontends: When web frontends are used for AA and an unauthorized user sends traffic, the user is redirected to a web frontend to provide credentials, e.g., a user name and password, or a client certificate. This is problematic, because it requires a valid IP configuration prior to authorization. Moreover, an HTTPS-capable web browser is not available on all platforms like printers, document scanners, phones, or surveillance cameras. Besides, re-directions require web browser usage. If browsers are not used, like on pure administration workplaces, users have to recognize that network connectivity is currently limited due to missing authorization and manually visit the web frontend. Providing credentials in a web frontend is cumbersome, especially if users are required to complete frequent re-authentications.

The web frontend directly interfaces an AA resource, e.g., an LDAP server or an SQL database. Therefore, the web application needs to implement the actual AA procedure. First, it may compare the input from a user with a hashed password from the AA resource. Then, it may report the authorization data, possibly including a VLAN tag, to the SDN controller. The authors in [22] and [23] focus on compatibility with existing backend authentication infrastructures that are based on RADIUS. The web frontend acts as a RADIUS client performing the AA procedure with the help of a RADIUS server.

3) AA in OpenFlow-Based SDN Using 802.1X: Some research prototypes for OpenFlow-based SDN adopt 802.1X for AA. Most of them make use of hostapd [24], an open-source user-space implementation for an 802.1X authenticator.

The SDN controller *FAUCET* [25] forwards EAPOL frames to a user-space instance of hostapd [24] that authenticates and authorizes the network client. The frontend hostapd_cli interfaces hostapd and outputs information about all AA attempts in a log file. A script monitors the log file and reports successful authentication attempts to the SDN controller. As an alternative approach, *AuthFlow* [26] extends hostapd by introducing an SSL-based communication channel to directly signal successful AA attempts to the SDN controller. Finally, *FlowIdentity* [27] builds a wrapper for running an instance of hostapd within the SDN controller Trema.

Instead of authenticating and authorizing users or end systems, *FlowNAC* [28] uses 802.1X to build a fine-granular network access control system to authenticate different applications on a network host. To enable multiple authentication and authorization processes per host, FlowNAC introduces EAPOL-in-EAPOL encapsulation as an extension to legacy 802.1X. This deviation from the original standard requires changes on all 802.1X components (supplicant modules, authenticator modules, and authentication servers) to support the additions. In particular, the implementations of the 802.1X supplicant (wpa_supplicant) and authenticator (hostapd) were extended to support EAPoL-in-EAPoL encapsulations.

IV. The AAM Architecture

The AAM is an AA module which serves as application for an SDN controller. We describe how the AAM implements the 802.1X concept, how it leverages multiple authentication resources, and how it maintains a session database.

A. Implementation of 802.1X in an SDN Context

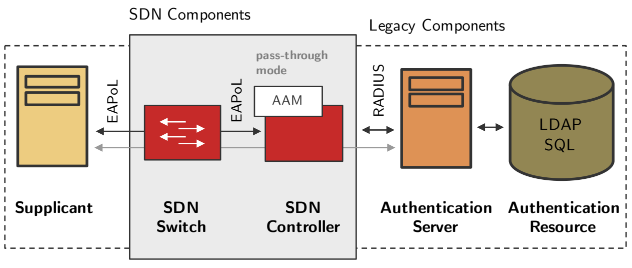
Figure 4(a) illustrates how 802.1X can be adopted for SDN. In legacy 802.1X infrastructures, the authenticator resides on the edge switch as shown in Figure 2. In SDN, we propose to implement the authenticator as a module, the AAM, on the SDN controller.

A network host initiates AA by sending an EAPOL-Start message as depicted in Figure 3. The SDN edge switch is instructed to forward that message to the controller which redirects it to the AAM. The AAM adopts the functionality of a legacy 802.1X authenticator, i.e., it relays communication between the supplicant and the authentication server as depicted in Figure 3. Therefore, it does not need to implement specific EAP types, e.g., EAP-TLS, EAP-TTLS, or EAP-PEAP. We designate this first mode of the AAM as *pass-through mode*.

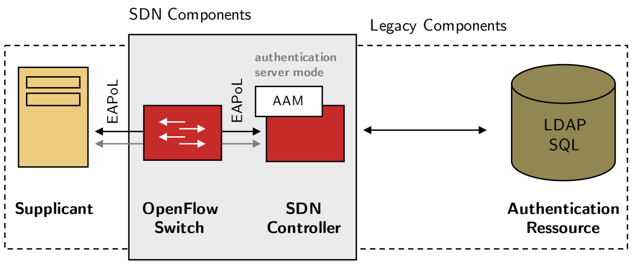
After successful authentication, the RADIUS server may return authorization data, e.g., a binary permission to access a particular network or a VLAN tag. This authorization data is transmitted in the RADIUS Access-Accept message as shown in Figure 3. The AAM implements mechanisms to translate authorization data from RADIUS Access-Accept messages into corresponding SDN rules to be applied on the SDN edge switch.

B. Integration of Alternative AA Resources

As mentioned before, legacy 802.1X authenticators interoperate only with RADIUS or Diameter and so does the above described concept for SDN. To allow for more flexibility, alternative AA resources should be supported. Figure 4(b) shows our proposal for that integration. The AAM essentially acts both as authenticator and authentication server. That means, the AAM must implement all EAP type specifics and perform the desired authentication procedure using authentication data



(a) Pass-through mode.



(b) Authentication server mode.

Fig. 4: Interaction of components in the SDN-enabled 802.1X architecture

from the alternative resources. We designate this second mode of the AAM as *authentication server mode*.

A simple example for an alternative AA is a CSV file containing user names, hashed passwords, and authorization data stored on the SDN controller accessible by the AAM. A more complex example is an LDAP or an SQL database with appropriate information that is remotely accessibly by the AAM but without a RADIUS server in between. The AA resources also provide authorization data that the AAM should apply after successful authentication in the form of appropriate flow rules on the edge switch.

If multiple AA resources exist, the AAM must choose the appropriate one. We propose two selection options for that problem: port-based resource selection and identity-based resource selection.

We first explain port-based resource selection. When an edge switch redirects the initial EAP request of the supplicant to the AAM, it includes context information, in particular the physical port and identifier of the SDN edge switch over which the packet was received. The AAM may use this port to determine the authentication resource to be used. A potential use case is a student lab where particular Ethernet ports may be authenticated and authorized using a simple AA resource instead of the RADIUS architecture for the overall campus network.

The alternative identity-based AA resource selection is limited to specific EAP methods. All EAP methods have an identical initialization routine where the authenticator requests the supplicant's identity to be transmitted in plain text. As an improvement for providing confidentiality about the identity on intermediate nodes, most EAP types support the concept of outer and inner identities. The outer identity, e.g., anonymous@foo.bar, is transmitted in the initialization in plain text and only serves as forwarding hint within a distributed

RADIUS infrastructure to find the RADIUS server in the home organization of the user. The actual identity of the user, e.g., john.smith@foo.bar is transmitted within an encrypted tunnel between supplicant and authentication server. The authors in [29] leverage multiple RADIUS infrastructures and use the outer identity to select the one to be used. We follow a similar approach and use the outer identity to determine the appropriate AA resource.

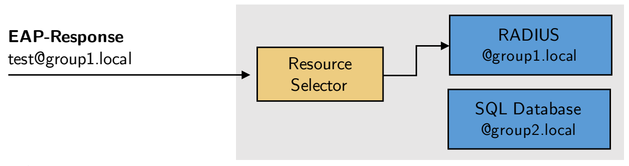


Fig. 5: The AAM selects the appropriate AA resource using the outer identity in the EAP response from the supplicant.

C. Network-Wide Session Database

We propose that the AAM maintains a network-wide session database which may be used by an identity-based network control system. This approach allows maintenance of user state while leaving AA servers and resources simple and stateless as originally desired.

The session database contains information about all authenticated and authorized identities, in particular all of their (simultaneously) active sessions. After a network host has successfully passed AA, the AAM adds a corresponding session entry to the session database. Conversely, the AAM uses port-down events from the SDN edge switch to remove sessions. As this feature is not available on all switches, sessions may be closed without notification. Therefore, we propose that the AAM may periodically issue EAP re-authentication requests to keep sessions and authorization alive.

Figure 6 illustrates exemplary contents of a session database for two identities. Each of their session entries contains information regarding the time of the last successful AA, the AA method used, the physical port of the network host and authorization information received by the AA resource.

```
{test@group1.local : { max_sessions : 2,
    sessions : (
        { aaa_time : Mo 13 Jun 2016 14:16:26 CEST,
          aaa_method : Radius(ip=10.0.20.100, meth=EAP-MD5),
          phys_port : OF-Switch(ip=10.0.20.222, port=1),
          assigned_vlans : (10)},
        { aaa_time : Mo 13 Jun 2016 14:18:31 CEST,
          aaa_method : Radius(ip=10.0.20.100, meth=EAP-MD5),
          phys_port : OF-Switch(ip=10.0.20.222, port=2),
          assigned_vlans : (10)},
         )},
 test@group2.local : { max_sessions : 1,
    sessions : (
        { aaa_time : Mo 13 Jun 2016 12:18:31 CEST,
          aaa_method : SqlDb(ip=10.0.20.101, meth=EAP-MD5),
          phys_port : OF-Switch(ip=10.0.20.222, port=3),
          assigned_vlans : (20)},
         )}
}
```

Fig. 6: Exemplary contents of a session database.

External applications can interact with the AAM by using communication techniques like REST interfaces. For example, after detection of uncommon behavior, a network security scanner may issue a de-authorization of a network host. The AAM triggers the necessary actions on the session database and SDN edge switches. Vice versa, the AAM could request the network security scanner to perform a scan on a network host that passed AA in a network for the first time.

V. Prototypical Implementation & Functional Validation

We explain the implementation of the AAM, describe our semi-virtualized testbed, and validate the AAM approach through experimentation.

A. Prototypical Implementation of the AAM

We implemented the AAM as a proof-of-concept for the Ryu SDN controller framework [30]. Its source code package contains several bootstrap applications that we used as starting point. We chose the *SimpleSwitch* application which controls an OpenFlow-based SDN switch in such a way that it acts like a simple Ethernet switch.

In contrast to most approaches presented in section III, we did not reuse the open-source 802.1X authenticator hostapd but provided a native Python-based implementation. We chose *dpkt* for network packet generation and parsing. We provided own implementations for EAP and RADIUS as they are not contained in *dpkt*.

For AAM's authentication server mode, the AAM requires implementations for EAP types (e.g., EAP-TTLS or EAP-PEAP) and interfaces for AA resources (e.g. LDAP database or SQL server). We designed an object-oriented class hierarchy and heavily used the concept of mixins to minimize intersections. For simplicity reasons, we implemented only EAP-MD5 as a single EAP type.

The initialization routine of the SDN controller installs on all connected SDN switches two proactive rules for each physical port that is marked as controlled by the AAM. The first rule forwards all EAPoL frames from network clients to the controller which relays them to the AAM. EAPoL frames are identified by the Ethernet frame's EtherType value 0x888E. The second rule drops all other packets at the switch port. This implements the behavior of the unauthorized state prior to successful AA.

The AAM also translates authorization data into flow rules and applies them to the corresponding port on the SDN edge switch. For example, simple permission of an identity just removes the flow rule for packet drop. More complex authorization data may result in flow rules setting a desired VLAN tag or allowing only a set of predefined destinations.

B. Testbed

We used the logical experimentation setup depicted in Figure 7 for the validation of the AAM. We experimented with a hardware- and a software-based OpenFlow-capable switch. The testbed is entirely virtualized except for the hardware switch. In the following, we describe the virtualization methodology and the OpenFlow switches in more detail.

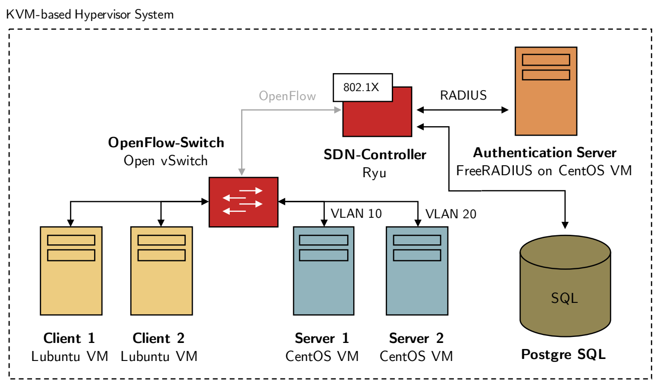


Fig. 7: Logical experimentation setup.

We used QEMU-based virtual machines with KVM acceleration managed by libvirt. As shown in Figure 7, we used Lubuntu 14.04 for user network hosts with a wpa_supplicant. As server network host, we used CentOS 7, and FreeRADIUS as backend authentication server.

We first experimented with HP Enterprise 2920 OpenFlow hardware switches. They failed handling EAPoL packets with firmware version 16.01.0006, i.e., flow rules installed to forward EAPoL frames to the controller were not effective, the frames were simply dropped. We consider this an incompatibility issue caused by the hybrid mode of the switch. Besides pure OpenFlow operation, the switch can be used as legacy L3/L4 switch with CLI and web frontend configuration. In legacy mode, the switch includes an 802.1X authenticator. As a hardware-based alternative, we used the SDN prototyping platform Zodiac FX with firmware version 0.66 that showed the expected behavior. As a software-based replacement, we used Open vSwitch [31] in version 2.4.0.

C. Validation Scenario

We used FreeRADIUS server and Postgre SQL as AA resources. The RADIUS server contained authentication data for user test@group1.local along with the VLAN tag 10 as authorization data. Authentication data for the user test@group2.local along with the VLAN tag 20 were stored in the SQL database. The AA resource was chosen on the outer identity provided in the EAP-Response message as depicted in Figure 3.

We performed various tests to validate the AAM's behavior. We tested the ability to use multiple AA resources. The 802.1X supplicant of Client 1 was alternately configured with the identity managed by the RADIUS server and the identity managed by the SQL database as AA resource. Depending on the identity, Client 1 got access to different VLANs, i.e., only Server 1 or Server 2 was reachable by an ICMP ping. We tested the session database, in particular its ability to limit the number of concurrent sessions. To that end, Client 1 and Client 2 were both authenticated with the same identity. We first set the number of concurrent sessions to be unlimited – both clients could be authorized. We then limited the number of concurrent sessions to 1 and verified that only a single session could be established.

VI. Scalability of the AAM

The AAM handles authentication requests, possibly initiates re-authentication requests, and performs authentication processes and cryptographic operations in authentication server mode. This may lead to high load in large networks and overload a central SDN controller.

We suggest to deploy the AAM as network function. It can be easily scaled to many instances because AA processes between multiple network hosts and authenticators are independent of each other and can be parallelized. Multiple AAM instances operate on the same session database that may also be distributed. The SDN controller installs flow rules for EAPoL frames on the edge switches to directly forward such frames to a particular AAM instance. This offloads the SDN controller and protects it from unauthorized network traffic.

VII. Conclusion

In this work we proposed to adopt 802.1X for authentication and authorization (AA) in SDN in such a way that multiple AA resources can be used and that a session database can be maintained. The former is useful for the integration of temporary and experimental local accounts, the latter for fine-grained network control systems which were a driver for early SDN research.

We implemented a controller application for OpenFlow-based SDN, the AA module (AAM). It uses EAP for frontend authentication so that it is compatible with existing end systems. In pass-through mode, the AAM leverages RADIUS or Diameter for backend authentication, which is easy to implement. In authentication server mode, the AAM acts as authentication server, i.e., it implements complex EAP types and checks AA data in LDAP servers, SQL databases, or local files. Independently of the specific mode, the AAM updates a session database with admitted sessions whenever a new session is authorized or whenever the controller recognizes a session teardown.

We implemented the AAM for the Ryu controller and conducted experiments in a semi-virtualized OpenFlow-controlled network including hardware and software switches. We validated the functionality of the proposed concept and observed that some switches refuse to forward EAPoL frames to their controller which may be problematic for the deployment of the AAM without appropriate patches. As the AAM performs time-consuming operations in authentication server mode, we suggested an organization of the AAM as a distributed network function to relieve the controller from potentially heavy load and to scale the AAM and the session database to very large networks.

VIII. Acknoledgements

The authors would like to thank Stefan Winter for fruitful discussions.

References

[1] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking Control of the Enterprise," in *ACM SIGCOMM*, Kyoto, Japan, Aug. 2007.

[2] A. Nayak, A. Reimers, N. Feamster, and R. Clark, "Resonance: Dynamic Access Control for Enterprise Networks," in *Workshop: Research on Enterprise Networking (WREN)*, Barcelona, Spain, Aug. 2009.

[3] H. Kim, J. Reich, A. Gupta, M. Shahbaz, N. Feamster, and R. Clark, "Kinetic: Verifiable Dynamic Network Control," in *USENIX Syposium on Networked Systems Design & Implementation (NSDI)*, 2015.

[4] Cyberoam. Cyberoam's Layer 8 Technology. [Online]. Available: https://www.cyberoam.com/layer8.html

[5] IEEE, "802.1X-2001 IEEE Standard for Local and Metropolitan Area Networks - Port-Based Network Access Control," 2001.

[6] ——, "802.1X-2004 IEEE Standard for Local and Metropolitan Area Networks - Port-Based Network Access Control," 2004.

[7] ——, "802.1X-2010 IEEE Standard for Local and Metropolitan Area Networks - Port-Based Network Access Control," 2010.

[8] Eduroam, "Eduroam - About." [Online]. Available: https://www.eduroam.org/index.php?p=about

[9] C. de Laat, G. Gross, L. Gommans, J. Vollbrecht, and D. Spence, "Generic AAA Architecture," RFC 2903 (Experimental), aug 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2903.txt

[10] P. Congdon, M. Sanchez, and B. Aboba, "RADIUS Attributes for Virtual LAN and Priority Support," RFC 4675 (Proposed Standard), sep 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4675.txt

[11] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz, "Extensible Authentication Protocol (EAP)," RFC 3748 (Proposed Standard), 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3748.txt

[12] C. Rigney, S. Willens, A. Rubens, and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)," RFC 2865 (Draft Standard), jun 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2865.txt

[13] V. Fajardo, J. Arkko, J. Loughney, and G. Zorn, "Diameter Base Protocol," RFC 6733 (Proposed Standard), 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6733.txt

[14] H. Tschofenig, D. Kroeselberg, A. Pashalidis, Y. Ohba, and F. Bersani, "The Extensible Authentication Protocol-Internet Key Exchange Protocol version 2 (EAP-IKEv2) Method," RFC 5106 (Experimental), feb 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5106.txt

[15] P. Funk and S. Blake-Wilson, "Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0)," RFC 5281 (Informational), aug 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5281.txt

[16] D. Simon, B. Aboba, and R. Hurst, "The EAP-TLS Authentication Protocol," RFC 5216 (Proposed Standard), 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5216.txt

[17] S. Brenza, A. Pawlowski, and C. Pöpper, "A practical investigation of identity theft vulnerabilities in eduroam," in *ACM Conference on Security and Privacy in Wireless and Mobile Networks (ACM WiSec)*, New York, NY, USA, Jun. 2015.

[18] M. Chiba, G. Dommety, M. Eklund, D. Mitton, and B. Aboba, "Dynamic Authorization Extensions to Remote Authentication Dial In User Service (RADIUS)," RFC 5176 (Informational), jan 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5176.txt

[19] FreeRADIUS, "FreeRADIUS Manual - Simultaneous-Use Attribute." [Online]. Available: ftp://ftp.gnu.org/old-gnu/Manuals/radius/html_node/radius_177.html#SEC180

[20] Raddb, "raddb 3.0.10 Documentation - rlm_radutmp." [Online]. Available: http://networkradius.com/doc/3.0.10/raddb/mods-available/radutmp.html

[21] A. Wang, X. Mei, J. Croft, M. Caesar, and B. Godfrey, "Ravel: A database-defined network," in *ACM Symposium on SDN Research (SOSR)*, Santa Clara, CA, USA, Mar. 2016.

[22] V. Dangovas and F. Kuliesius, "Sdn-driven authentication and access control system," in *The International Conference on Digital Information, Networking, and Wireless Communications (DINWC)*. Society of Digital Information and Wireless Communication, 2014, p. 20.

[23] F. Kuliesius and V. Dangovas, "Sdn enhanced campus network authentication and access control system," in *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*, July 2016, pp. 894–899.

[24] J. Malinen, "hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator." [Online]. Available: https://w1.fi/hostapd/

[25] "FAUCET SDN: 802.1x authentication on FAUCET (NFV offload of authentication)." [Online]. Available: https://faucet-sdn.blogspot.de/2016/07/8021x-authentication-on-faucet-nfv.html

[26] D. M. Ferrazani Mattos and O. C. M. B. Duarte, "Authflow: authentication and access control mechanism for software defined networking," *Annals of Telecommunications*, pp. 1–9, 2016. [Online]. Available: http://dx.doi.org/10.1007/s12243-016-0505-z

[27] S. T. Yakasai and C. G. Guy, "Flowidentity: Software-defined network access control," in *Network Function Virtualization and Software Defined Network (NFV-SDN), 2015 IEEE Conference on*, Nov 2015, pp. 115–120.

[28] J. Matias, J. Garay, A. Mendiola, N. Toledo, and E. Jacob, "Flownac: Flow-based network access control," in *European Workshop on Software Defined Networks (EWSDN)*, Budapest, Hungary, Sep. 2014.

[29] Z. Cao, J. Fitschen, and P. Papadimitriou, "Freesurf: Application-centric wireless access with sdn," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 357–358. [Online]. Available: http://doi.acm.org/10.1145/2785956.2790000

[30] Ryu, "Ryu SDN Framework." [Online]. Available: https://osrg.github.io/ryu/

[31] O. vSwitch, "Open vSwitch - Production Quality, Multilayer Open Virtual Switch." [Online]. Available: http://openvswitch.org/

2 Submitted Manuscripts (Core Content)

2.1 P4sec: Automated Deployment of 802.1X, IPsec, and MACsec Network Protection in P4-Based SDN

P4sec: Automated Deployment of 802.1X, IPsec, and MACsec Network Protection in P4-Based SDN

Frederik Hauser, Marco Häberle, Michael Menth

***Abstract*—We present a three-tier control architecture (P4sec) for networks with P4-programmable switches operated through software-defined networking (SDN) control (P4-based SDN). The networks consist of multiple sites and are operated through distributed controllers.**

802.1X, MACsec, and IPsec are widespread network security mechanisms that control network access and add encryption and authentication to L2 and L3 networking. They are standardized by IEEE and IETF, and are part of most open-source and commercial network hardware and software appliances. However, lots of manual configuration is needed for their application in traditional networks. The objective of P4sec is their automated deployment and operation in P4-based software-defined networks.

In this paper, we briefly introduce data plane programming with P4 and give an overview of 802.1X, MACsec, and IPsec. We explain the three-tier control architecture P4sec and validate it by a prototype which is published under the Apache v2 license on GitHub. Finally, we discuss opportunities and challenges.

I. Introduction

Software-defined networking (SDN) and programmable forwarding hardware facilitate disaggregation of communication network infrastructure, speed up innovation, and increase control flexibility. With SDN, communication devices are controlled by controllers through an open protocol so that controllers can be provided by other players than the vendor of the hardware. Moreover, they can include external information such as from authentication and authorization servers to steer the network, leading to better informed decisions. The forwarding behaviour and control interface of switches with a programmable data plane can be programmed, which provides additional flexibility. P4 is a programming language for such switches with lots of software and hardware support. We denote networks with P4-programmable switches steered by a controller as P4-based SDN. Both SDN and programmable data planes contribute to network softwarization.

Flexible network composition and access are getting more important. Connecting branch offices, providing access to remote workers, or connecting to public cloud workloads are common requirements for today's networks. However, the growing flexibility is a challenge for security management.

802.1X, MACsec, and IPsec are widely used, vendor-independent security mechanisms. 802.1X is an IEEE standard for port-based network access control (NAC) that ensures that network hosts can get access to a network after successful authentication and authorization (AA). MACsec is an IEEE standard for point-to-point encryption and authentication of Ethernet links between switches, switches and hosts, and between hosts. IPsec is a set of Internet standards that adds security to the Internet Protocol (IP), mostly used to set up virtual private network (VPN) tunnels. All three mechanisms benefit from cross-vendor standardization, comprehensive support in open-source and proprietary hardware/software appliances, continuous further development, and extensive experience in operation and security.

However, setting up these security mechanisms in large networks and operating them with frequent changes is difficult. Initial setup and operation are typically performed by configuration via command line interfaces (CLIs) or through network management systems. Especially in heterogeneous deployments with different implementations of these protocols, incompatibilities increase configuration complexity and error space. Therefore, many administrators hesitate to set up those mechanisms.

The contribution of this paper is P4sec. It is an integrated controller-based approach which automates the mentioned security mechanisms and facilitates their deployment. It removes the need for complex protocols with partly incompatible implementations, and ensures correct configuration according to a given higher-level specification. P4sec is a three-tier control architecture for P4-based SDN consisting of multiple sites. That is, there is a local controller (LoCo) on any P4 switch, a site controller (SiCo) in any site, and one global WAN controller (WaCo). Thus, key in this work is the SDN-based control with customized application programming interfaces (APIs) provided by programmable switches. A particular focus of P4sec is to automate the deployment and operation of 802.1X, MACsec, and IPsec, but it is inherently extensible.

The rest of this article is structured as follows. In Section II, we give an overview of data plane programming with P4 and the security mechanisms 802.1X, MACsec, and IPsec. Section III classifies P4sec with regard to related work. In Section IV, we introduce P4sec. Section V describes the implementation and evaluation of a prototype. In Section VI, we discuss opportunities and challenges. Section VII concludes this work.

II. Technologies

We give an introduction to data plane programming with P4 and to the security mechanisms 802.1X, MACsec, and IPsec.

A. Data Plane Programming with P4

Traditional software-defined networking (SDN), e.g., OpenFlow, introduces control plane programmability in networks. Network devices with fixed data plane functionality are equipped with an API that allows programmers to bypass the devices' control plane and introduce custom control plane algorithms in the form of software programs.

Data plane programming enables users to also change the algorithms that describe how packets are processed by the data plane. P4 (Programming Protocol-Independent Packet Processors) [1] is a domain-specific language (DSL) for describing data planes. P4 was introduced as research work; now it is openly evolved in the P4 Language Consortium. The survey [2] gives an overview of fundamentals, advances, and applied research with P4.

In a nutshell, P4 programs contain the description of data plane algorithms on the basis of the protocol-independent switching architecture (PISA) packet processing pipeline. A programmable parser extracts packet headers, the programmable match-action pipeline applies actions to packet header data based on matching criteria, and the programmable deparser serializes the modified packet headers for transmission. While actions are part of the P4 program, matching criteria with assigned actions and parameters are maintained by an SDN control plane. P4 programs are compiled for execution on software or hardware P4 targets that implement the functionality of the P4 language core. Additional target-specific functions, e.g., hash calculations, can be provided by the manufacturer or, on some platforms, implemented by the programmer and used as P4 externs within the P4 program.

B. Network Access Control with 802.1X

802.1X is an IEEE standard for port-based network access control (NAC). Its objective is to grant network access to network devices only after successful AA.

With 802.1X, a network device requests network access from an access node, e.g., a switch or an access point, which either grants or denies network access with the help of a remote AA server. To that end, 802.1X describes a supplicant module on the network device, an authenticator module for the access device, and a remote AA server. The supplicant uses the Extensible Authentication Protocol (EAP) to request access, and the authenticator utilizes the Remote Authentication Dial-In User Service (RADIUS) to relay the request to the AA server which authenticates the supplicant and returns the authorization result to the authenticator.

C. L2 Security with MACsec

MACsec (IEEE 802.1AE) introduces point-to-point security for LANs by adding integrity, confidentiality, and replay protection for Ethernet frames. MACsec can secure direct links between switches, switches and hosts, and between hosts. Most enterprise-grade switches support MACsec; since v4.6 it is also part of the Linux kernel.

MACsec protects links between peers by adding two unidirectional secure channels (SCs), each with multiple security associations (SAs) that comprise individual keying material. SA management can be automated by the MACsec key agreement (MKA) protocol between two MACsec peers or via EAP. However, SCs still need to be configured manually. Fully automated MACsec operation with SC setup requires a secure topology discovery mechanism that is not available on current systems.

D. L3 Security with IPsec VPN

IPsec adds security to IP traffic; it is a complex and comprehensive protocol suite standardized in the IETF. IPsec comprises two protocols, Authentication Header (AH) and Encrypted Security Payload (ESP), and two operation modes, the transport and tunnel mode. Today, IPsec is mostly used with ESP in tunnel mode to create encrypted site-to-site or host-to-site VPN tunnels.

While MACsec protects the entire traffic on a L2 link between MACsec peers, IPsec protection is applied only to selected traffic flows between IPsec peers. Therefore, security policies (SPs) stored in a security policy database (SPD) comprise matching patterns, e.g., IP source or destination addresses, protocol, and actions that define whether the traffic should be protected by IPsec, forwarded as is, or dropped.

Similar to MACsec, IPsec protection is defined in SAs that are stored in the security association database (SAD). Also similar to MACsec, the SPs in the SPD are typically configured manually while the SAs are managed via distributed protocols such as Internet Key Exchange (IKE) directly between the IPsec peers.

III. Related Work

We briefly present preliminary studies for P4sec, delimit P4sec from other SDN-based network security platforms, and give examples for other P4 data plane applications addressing entire use cases rather than selected features.

A. Preliminary Studies for P4sec

We presented a first implementation of 802.1X for OpenFlow-based SDN [3]. The 802.1X authenticator runs as an SDN application on the controller and OpenFlow switches use packet-in and -out mechanisms to hand over AA traffic to the controller. A database for user session tracking and support for AA resources other than RADIUS was introduced.

P4-MACsec [4] automates operation of MACsec for P4-based SDN. MACsec processing is part of the P4 data plane that is managed by a two-tier control plane. MACsec encryption and decryption using AES-GCM are implemented as P4 externs. We proposed secure topology detection and monitoring that are prerequisite for automated MACsec operation by encrypting Link Layer Discovery Protocol (LLDP) packets with AES-GCM.

P4-IPsec [5] supports site-to-site and host-to-site VPN with IPsec for P4-based SDN. Similar to P4-MACsec, the P4 data plane implements ESP processing with an SPD and SAD while IPsec cipher suites for encryption and decryption are realized as P4 externs. Instead of manual configuration and peer-to-peer SA management, a centralized SDN control plane manages the SPD and SAD on the P4 switches. An agent tool for Linux hosts enables the SDN control plane to manage IPsec VPN tunnels on remote clients.

B. SDN-Based Network Security Platforms

The majority of SDN-based network security platforms utilize an SDN controller for centralized network access control.

Fine-granular and possibly context-aware policies are formulated in DSLs and translated into network control. Ethane [6] is a popular example and predecessor of OpenFlow. Poise [7] is a recent P4-based work that targets BYOD scenarios. Other related work presents mitigation mechanisms for particular attacks. Qin et al. [8] utilize P4 for a scalable Intrusion Detection System (IDS) for the network edge that is based on federated learning. Scholz et al. [9] present a mechanism against Distributed Denial of Service (DDoS) attacks with SYN flooding. Those and other innovative works benefit from the flexibility and line-rate performance of P4 data plane applications coupled with the capabilities of software-defined control.

P4sec does not fall into the aforementioned categories; it does not present an alternative concept but automates the operation of existing concepts, i.e., 802.1X, MACsec, and IPsec.

C. P4 Data Plane Applications for Use Cases

Recent studies provide customized P4 programs with a set of functions for entire use cases. DC.p4 [10] proposes a P4-based data center switch with protocols and functions such as VLAN, NVGRE, and VXLAN. P4-BNG [11] presents a P4-based Broadband Network Gateway (BNG) with protocols and functions such as PPPoE, QoS, or IP multicast. The Router for Academia Research and Education (RARE) project [12] from GÉANT develops a P4-based data plane for the FreeRTR open-source control plane for use in research and education networks. It comprises protocols and functions such as SR-MPLS, VPLS, and EVPN. Similarly, P4sec aggregates basic network and security functions.

IV. P4sec Architecture

We give an overview of P4sec, introduce its three-tier control plane, and present its network security features. To simplify the description, we assume that all switches in the network are P4sec switches, i.e., they are P4-capable and implement P4sec. Later we discuss how P4sec may be deployed incrementally. Hosts are also denoted as clients in the following.

A. Overview

We explain P4sec in Figure 1. The network consists of two sites that are connected via the Internet. The P4sec switches implement L2 forwarding and L3 routing including the P4sec security features. Those are

1) 802.1X to authenticate hosts and authorize them for network access.
2) MACsec between neighboring switches as well as between switches and their attached hosts. This also includes secure link layer discovery with LLDPsec.
3) IPsec to interconnect different sites and to connect remote clients to a site using secure VPN tunnels.

This functionality is achieved by a three-tier SDN control plane that consists of local controllers (LoCos), site controllers (SiCos), and a WAN controller (WaCo). A P4sec agent running on a local or remote client holds a management connection to the SiCo or WaCo, respectively so that local and remote clients are also part of automated deployment of MACsec and IPsec.

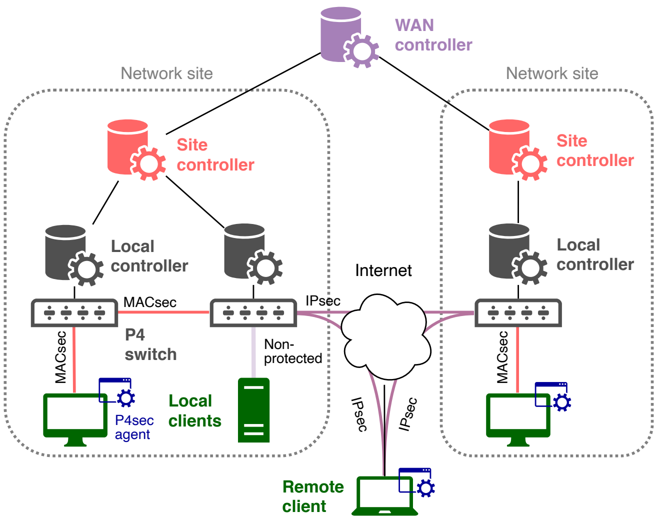


Fig. 1. P4sec architecture in an exemplary network with two sites and a remote client connected via the Internet.

B. Three-Tier Control Plane

Resilience, scalability, and low latency are particular challenges for control planes in large-scale networks. Therefore, P4sec utilizes a three-tier control plane whose architecture is depicted in Figure 2. On the lowest tier, P4sec switches are managed by dedicated LoCos. SiCos are responsible for managing a site. The WaCo is responsible for managing the entire network. Controllers of adjacent tiers communicate with each other using configured addresses. The connection is secured by TLS which adds mutual authentication with certificates and encryption.

As shown in the figure, P4sec supports six P4sec control plane functions: 802.1X, L2 forwarding, LLDP, MACsec, L3 routing, and IPsec. They are provided by one or multiple tiers. For example, L2 forwarding is supported only by the LoCo; IPsec VPN requires support by all three tiers with the WaCo as the leading system so that communication between the control layers is needed.

For inter-layer communication, the WaCo exchanges messages with the SiCos, and the SiCos with the LoCos. Requests and notifications are sent upwards while instructions are sent downwards.

Three generic control plane services simplify the implementation of the control plane functions. The *status* service offers status information for other controllers on request. The *register* service allows controller registration at upward controllers. The *execute* service provides command execution on downward controllers.

C. Protected Network Access: 802.1X

While we implemented a similar approach for OpenFlow [3], P4sec introduces a first implementation of 802.1X for

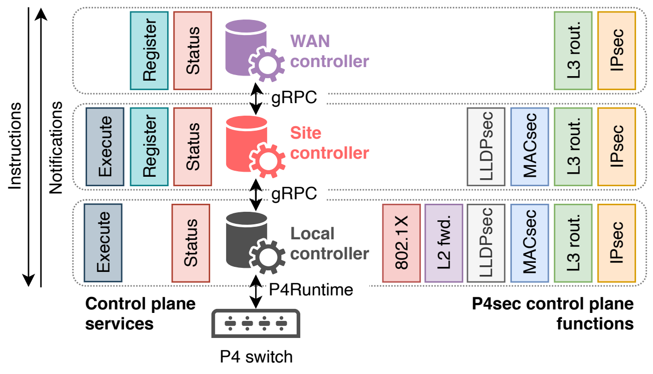


Fig. 2. Communication in the three-tier control plane.

P4-based SDN. It supports standard 802.1X supplicants on local clients and RADIUS AA servers as it implements only the 802.1X functionality that is provided by conventional switches, i.e., an 802.1X authenticator and an in-/out-port authorizer. Figure 3 illustrates the implementation concept.

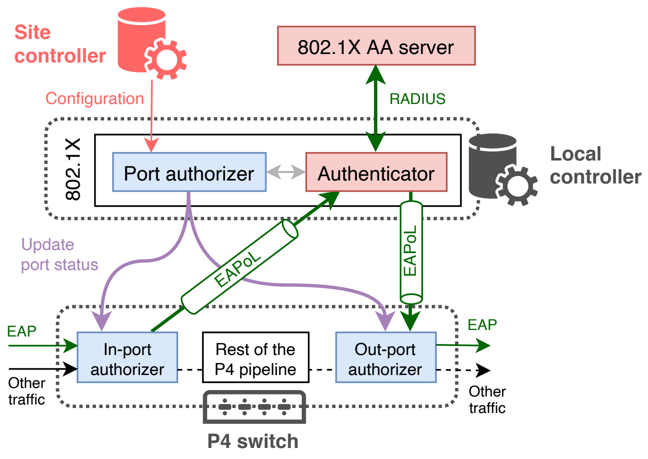


Fig. 3. Operation of 802.1X in P4sec.

1) Port Authorization: The in-/out-ports of an interface of a P4sec switch can be in three different states. In *forced authorized*, all packets are accepted by default. In *forced unauthorized*, all packets are dropped. In *auto*, AA is performed and only EAPoL and LLDPsec packets are forwarded. After successful AA for a specific MAC address, only packets from that MAC address and to that MAC address are allowed. This makes it more difficult for attackers to inject arbitrary packets via authenticated ports.

The port states are managed by the port authorizer function on the LoCo. Initially, the port states are pre-configured via security policies received from the WaCo via the SiCo. The policies may also be modified at runtime. In *auto* state, EAPoL-based AA is performed via the authenticator function and decides if the port accepts or drops packets. Therefore, the port authorizer function receives control instructions from the authenticator function. Until successful AA, all packets except EAPoL and LLDPsec are dropped.

2) Authentication: The authenticator function on the LoCo implements the functionality of an 802.1X authenticator. It relays AA traffic between the supplicant and 802.1X AA server and forwards the received authorization decision to the port authorizer function.

The 802.1X supplicant on the local client initiates AA by sending EAPoL frames. In *auto* state, the P4sec switch forwards them to the LoCo. Its authenticator function decapsulates the EAP frames, re-encapsulates them in RADIUS, and forwards them to the 802.1X AA server. Reverse traffic is treated similarly. In case of successful authentication, the port authorizer function of the LoCo updates the in-/out-port state on the P4sec switch.

D. Secure L2 Connectivity: L2 Forwarding & MACsec

We summarize how L2 forwarding and MACsec operation are automated in P4sec. We adopt the implementation of P4-MACsec (see Section III-A) but extend it by support for MACsec protection between P4sec switches and local clients. The implementation details can be found in [4].

1) L2 Forwarding: A P4sec switch performs L2 forwarding according to the L2 forwarding table that maps MAC addresses to out-ports of the switch. This L2 forwarding table is managed by the LoCo. In case of unknown mappings, the L2 forwarding function on the LoCo initiates and performs MAC learning by packet flooding, i.e., the packet is cloned and sent out on all active ports. By receiving a response to the flooded packet, the new mapping is learned by the LoCo and added to the L2 forwarding table.

2) Secure LLDP (LLDPsec): Automated protection of links requires knowledge about the network topology. LLDP is typically used for this purpose but suffers from missing encryption and authentication, which is an obvious security vulnerability. Therefore, we present LLDPsec, a security-enhanced version for P4. It encrypts the LLDP payload with AES-GCM to protect against spoofing, tampering, and replay attacks.

We implement a simple mechanism on the P4sec switches to exchange LLDPsec packets with the LoCo. The LLDPsec control plane function is implemented on the LoCo and SiCo. The LLDPsec function on the SiCo first generates a site-wide key for AES-GCM encryption/decryption of LLDP packets and distributes it to all LoCos. Then, the LoCos periodically send out encrypted LLDP packets on all active ports of the assigned P4sec switch. Other P4sec switches receive these messages and forward them to their LoCos. After reception of an LLDP packet, the LoCo decrypts and extracts its information, and records it in a local link store. Any change in this local store is communicated to the SiCo so that the SiCo always has a valid, site-wide topology map. Failing P4sec switches are automatically recognized by the LoCo and reported to the SiCo. Failing links are recognized in periodic rediscoveries.

The P4sec agent includes a similar implementation of LLDPsec. It ensures that local clients are also part of secure link discovery that is a prerequisite for automated operation of MACsec.

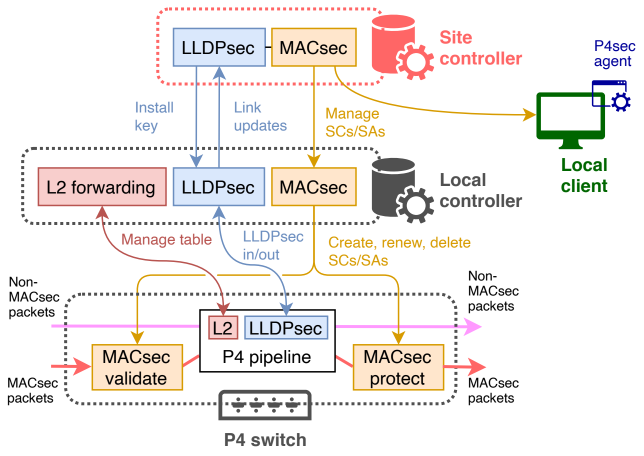


Fig. 4. Operation of MACsec in P4sec.

3) MACsec Integration with P4: As depicted in Figure 4, MACsec operation is automated by the SiCo. As for LLDPsec, the MACsec control plane function is implemented on SiCos and LoCos and on the P4sec agent of local Linux hosts. The MACsec function on the SiCo is fed with topology data from the LLDPsec function. For each link, it creates a MACsec SC and generates the required SA data. This configuration data is then forwarded to the LoCo or P4sec agent which set up the MACsec SC on the assigned P4sec switch or on the local client.

On the data plane, the MACsec protect function applies MAcsec protection on Ethernet packets and the MACsec validate function performs the inverse function. Both functions are implemented as P4 externs.

MACsec SAs are renewed regularly. Renewals may be triggered by either timeout limits that are supervised by the LoCo for each MACsec SA or packet counters for each MACsec SA on a P4sec switch or local client. When the packet counter reaches a predefined soft limit, the P4sec switch or host notifies its LoCo or P4sec agent to set up a new MACsec SA. When the packet counter reaches a hard limit, consecutive packets are dropped so that the SA is no longer valid. With each SA renewal, the packet counter starts again with zero.

E. Secure L3 Connectivity: IP Routing & IPsec

Secure L3 connectivity in P4sec comprises L3 routing, site-to-site IPsec VPN tunnels between network sites, and host-to-site IPsec VPN tunnels between remote clients and P4sec switches. Implementation details can be found in [5]. We describe how IPsec operation is automated in P4sec.

1) L3 Routing with LPM: In P4sec, each network site may be subdivided into different IP subnets. To provide connectivity among them, P4sec switches implement L3 routing with longest prefix match (LPM). Each P4sec switch has a LPM table that is maintained by the L3 routing function on the control plane that runs on all three tiers. Figure 5 depicts this process. On the WaCo, network administrators define IP address ranges and assign them to P4sec switches. The WaCo sends this information to the corresponding SiCos. The SiCo then computes the shortest paths within its site and sends this information to the LoCos. Then, the LoCo writes these LPM rules to the match-action-tables on the P4sec switch which define the L3 forwarding tables.

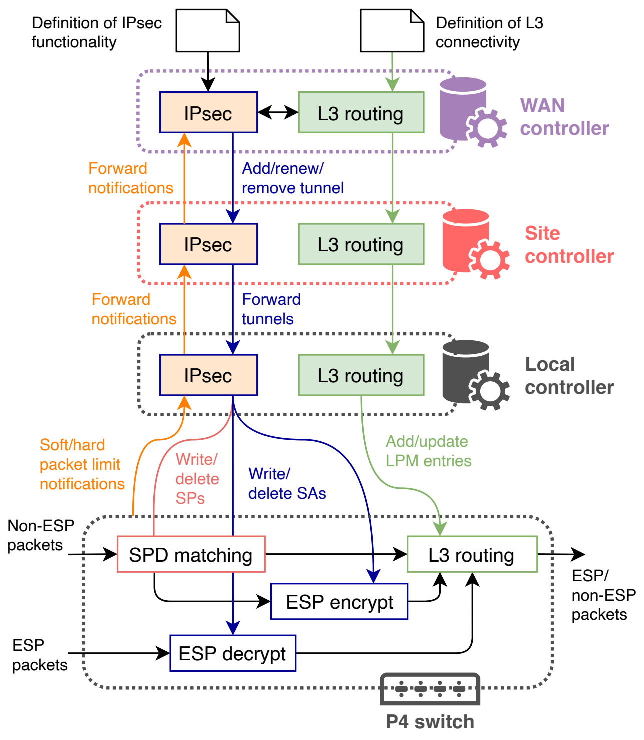


Fig. 5. Operation of IPsec site-to-site VPN tunnels between switches in P4sec.

2) Site-to-Site Tunnels between Network Sites: P4sec introduces automated setup of site-to-site IPsec VPN tunnels between P4sec switches to interconnect distant sites. As all P4sec switches implement the same functionality, each P4sec switch can act as VPN gateway to terminate IPsec VPN tunnels. As prerequisite, P4sec switches serving as VPN gateways must be reachable via the Internet.

Figure 5 depicts how site-to-site IPsec VPN tunnels are set up in P4sec. Site-to-site IPsec functionality is defined on the WaCo in a configuration file that comprises the two P4sec switches that serve as VPN endpoints. On control plane startup and whenever the configuration changes, the WaCo creates the required IPsec configuration data. The WaCo generates IPsec data, e.g., SPs and SAs, and queries information about the IP subnets to be connected from the L3 routing function. The aggregated configuration data is then forwarded via the SiCo to the LoCo of the corresponding P4sec switches. As in MACsec, the LoCos modify the match-action tables of the P4sec switch to apply the received configuration. With any new IPsec tunnel also the L3 routing information on the P4sec switches needs to be updated. Therefore, the WaCo generates L3 routing information that is sent to the SiCo and installed on the P4sec switches via the LoCo. As with MACsec, the ESP encrypt and decrypt functions on the P4 data plane are implemented as P4 externs.

Rekeying events are initiated by the P4sec switch that holds a counter for every IPsec SA. The counter is incremented with every encrypted/decrypted packet. Similarly to MACsec, soft and hard limits are set. While soft limits trigger SA renewal notifications, hard limits cause immediate packet dropping.

3) Host-to-Site Tunnels between Remote Clients and P4sec Switches: P4sec switches acting as VPN gateways are indicated in a host-to-site IPsec configuration file on the WaCo. The configuration file also contains a mapping from particular local clients to their responsible VPN-enabled P4sec switches.

As shown in Figure 6, remote clients may request an IPsec VPN tunnel for a particular local client. To facilitate that, the remote client runs a P4sec agent, just like for MACsec operation, which holds a management connection to the WaCo. VPN tunnel setup for particular targets can be requested via a CLI provided by the P4sec agent. After receiving the VPN tunnel setup request, the WaCo creates the IPsec VPN tunnel configuration data, such as in site-to-site operation, and forwards it to the LoCo of the responsible VPN-capable P4sec switch and to the P4sec agent on the remote client.

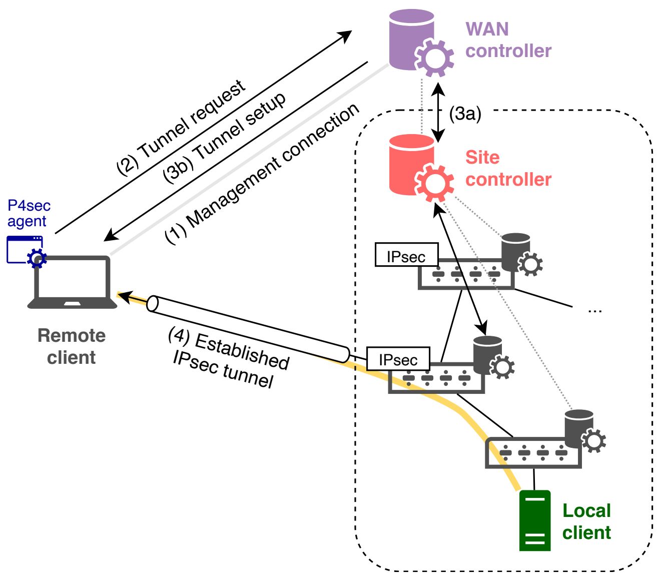


Fig. 6. Operation of IPsec host-to-site VPN tunnels between remote hosts and switches in P4sec.

V. Prototypical Implementation & Evaluation

We describe our prototype implementation and evaluation. Its source code is released under the Apache v2 license on GitHub [13].

A. Control Plane Implementation

We built the P4sec control plan from scratch, but integrated and extended functionalities of P4-MACsec and P4-IPsec [4], [5]. The different controllers of the three-tier control plane (WaCo, SiCo, LoCo) are implemented as Python 3 applications. The interfaces between them are integrated with gRPC; the control message structures, e.g., for tunnel setup descriptions, are defined with protobuf. The Scapy library is used for packet generation and parsing.

The P4sec client agent is also implemented in Python 3. It uses iproute2 commands for configuration of the MACsec and IPsec kernel functions. For MACsec, a virtual interface is created and attached to the network interface. For IPsec, VPN tunnels are created with the help of `ip xfrm`.

B. Data Plane Implementation

The data plane functions of P4sec are implemented for the bmv2 P4 software target. Its functionality is extended by four P4 externs for MACsec and IPsec that handle packet encryption and decryption. The P4 externs are implemented in C++; cryptographic operations are provided by the OpenSSL library. The P4Runtime API is used for data plane management through the LoCo. As novelty of P4sec, we extend the existing data plane implementations by 802.1X functionality and add support for MACsec on links between switches and local clients.

In previous works, we demonstrated two ways of porting P4 externs to P4 hardware targets. In P4-MACsec [4] and P4-IPsec [5], we implemented simplified versions of the MACsec and IPsec P4 externs directly on the FPGA of a NetFPGA SUME P4 hardware target. In P4-IPsec [5], we additionally converted the P4 externs to software modules. Those software modules are executed on the Linux-based computing unit that is part of P4 hardware targets based on the Tofino ASIC. Packet exchange between the P4 program and computing unit is realized via a packet-in/-out mechanism.

C. Functional Evaluation

We evaluate P4sec in a virtualized network environment similar to Figure 1. It comprises an exemplary network with two sites, each with multiple P4sec switches and local clients. The network sites are inter-connected via an IPsec VPN tunnel. A remote client establishes IPsec VPN tunnels to both sites. The exemplary setup includes six controllers and a FreeRADIUS AA server. The testbed environment is built with Mininet. It runs within an Ubuntu virtual machine that is automatically provisioned via Vagrant.

We investigated the data plane performance of IPsec and MACsec in [4], [5]. Here, we validate the functionality of the new three-tier control plane in the virtual testbed and measure response times.

Interactions between the 802.1X supplicant on the local client and the FreeRADIUS server take about 1 s. We measure the same response times in legacy 802.1X deployments, i.e., the long delay is not specific to P4sec. In real-world scenarios, link propagation delays between involved components extend the measured response times. All other operations between controllers of the three-tier control plane and P4 switches (e.g., link detection, detection of a failed link, MACsec setup) complete within about 10 ms.

VI. Opportunities and Challenges

P4sec integrates the well-understood and widely used security mechanisms 802.1X, MACsec, and IPsec with P4-based SDN. It automates their configuration while the remaining implementations are standard-compliant. This yields several benefits. Controller-based 802.1X protects against session hijacking, a session database facilitates user tracking, and AA resources other than RADIUS can be used [3]. MACsec and IPsec profit from automation as the SDN control plane relieves network administrators from low-level device configuration and devices from error-prone peer-to-peer signaling. MACsec benefits from secure topology discovery and automated configuration as it saves manual effort. IPsec management is simplified through high-level definitions on the WaCo that are the basis for automated setup and operation of VPN tunnels. This facilitates meshed interconnection of sites with low complexity, which is an objective of SD-WAN. As all P4sec switches can serve as VPN endpoints, traffic within a site can be protected very close to its destination and the bottleneck of a single VPN concentrator can be avoided.

P4sec can be deployed incrementally in production networks. Early IPsec support on P4-based core switches is already possible, e.g., on the base of Tofino ASICs and workarounds for missing encryption [5]. As soon as P4 switches support encryption at line rate, MACsec could be deployed to harden the backbone. With the advent of P4-programmable access switches, 802.1X may be rolled out with competitive costs.

Flexibility towards extensions is a main benefit of P4sec. Additional features could be integrated into the data and control plane. More recent VPN technologies such as OpenVPN or WireGuard could be added as alternative to IPsec. Network access control could be further hardened by multi-factor authentication (MFA) for users or context-aware admission as known from Zero Trust networks. Morover, a software-defined perimeter may established using SDN and possibly also P4 to protect Internet-facing ports with IDS and DDoS prevention [8], [9].

Publishing data plane programs such as P4sec as open-source software is a particular advantage for network security applications. They benefit from accelerated development cycles, peer audit, and from the ability to customize feature sets to achieve minimal attack surfaces. P4's sufficiently high abstraction level and the availability of P4 targets have created a large community making those benefits effective.

P4sec is still a research prototype and not yet production-ready as best-suited hardware is not yet available. While functions such as 802.1X or LLDPsec are implementable by standard P4 language constructs, MACsec and IPsec require cryptographic functions that are still missing on P4 hardware targets. Nevertheless, workarounds for encryption on P4 hardware targets have been demonstrated in several papers with a throughput between 1 and 11 Gbit/s using the main CPU module on P4 hardware targets [5], outsourcing to NFV nodes [5], or constrained hardware implementations [4], [5], [14]. Thus, there is substantial interest in more cryptographic externs on P4 targets. Currently, there are only P4 hardware targets for WAN and data center applications while typical access switches with lower bandwidth but more advanced functionality, more ports, and powerful computing modules are still missing.

VII. Conclusion

802.1X, MACsec, and IPsec are widespread and mature technologies for network security but complex in setup and operation. P4sec automates their operation in P4-based SDN. It supports flexible and secure networking in campus and enterprise networks with distributed sites that are connected via VPN tunnels. In P4sec, data plane functionalities are implemented on P4 targets. They are managed by a three-tier control plane that distributes functionality, which is beneficial for load balancing and scalability. We implemented a software prototype based on the BMv2 P4 software target and released its source code under the Apache v2 license on GitHub.

Acknowledgement

The authors would like to thank Arwed Mett and Mark Schmidt for fruitful discussions and programming contributions.

This work was partly supported by the bwNET2020+ project which is funded by the Ministry of Science, Research and the Arts Baden-Württemberg (MWK) as well as the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/1-2. The authors alone are responsible for the content of this paper.

References

[1] P. Bosshart *et al.*, "P4: Programming Protocol-Independent Packet Processors," *ACM SIGCOMM Computer Communication Review (CCR)*, vol. 44, no. 3, 2014.

[2] F. Hauser *et al.*, "A Survey on Data Plane Programming with P4: Fundamentals, Advances, and Applied Research," *arXiv CoRR*, vol. abs/1207.0016, 2021.

[3] ——, "Establishing a Session Database for SDN using 802.1X and Multiple Authentication Resources," in *IEEE International Conference on Communications (ICC)*, 2017.

[4] ——, "P4-MACsec: Dynamic Topology Monitoring and Data Layer Protection With MACsec in P4-Based SDN," *IEEE Access*, 2020.

[5] ——, "P4-IPsec: Site-to-Site and Host-to-Site VPN With IPsec in P4-Based SDN," *IEEE Access*, vol. 8, 2020.

[6] M. Casado *et al.*, "Ethane: Taking Control of the Enterprise," *SIGCOMM Computer Communication Review (CCR)*, vol. 37, no. 4, 2007.

[7] Q. Kang *et al.*, "Programmable In-Network Security for Context-aware BYOD Policies," in *USENIX Security Symposium (USENIX Security)*, 2020.

[8] Q. Qin, K. Poularakis, K. Leung, and L. Tassiulas, "Line-Speed and Scalable Intrusion Detection at the Network Edge via Federated Learning," in *IFIP Networking*, 2020.

[9] D. Scholz, S. Gallenmüller, H. Stubbe, and G. Carle, "SYN Flood Defense in Programmable Data Planes," in *P4 Workshop in Europe*, 2020.

[10] A. Sivaraman *et al.*, "DC.P4: Programming the Forwarding Plane of a Data-Center Switch," in *ACM SIGCOMM Symposium on Software Defined Networking Research (SOSR)*, 2015.

[11] R. Kundel *et al.*, "P4-BNG: Central Office Network Functions on Programmable Packet Pipelines," in *International Conference on Network and Service Management (CNSM)*, 2019.

[12] "GÉANT: RARE project," https://connect.geant.org/2020/01/17/rare-project, last acessed on 03-31-2021.

[13] "GitHub: uni-tue-kn/P4sec," https://github.com/uni-tue-kn/P4sec, last accessed on 03-31-2021.

[14] X. Chen, "Implementing AES Encryption on Programmable Switches via Scrambled Lookup Tables," in *Workshop on Secure Programmable Network Infrastructure (SPIN)*, 2020.

Frederik Hauser (Graduate Student Member, IEEE) studied computer science at the University of Tuebingen, Germany, and received his Master's degree. Since then, he has been a researcher at the Chair of Communication Networks at the University of Tuebingen, pursuing his PhD. His main research interests include software defined networking, network function virtualization, and network security.

Marco Haeberle (Student Member, IEEE) studied computer science at the University of Tuebingen, Germany, and received his Master's degree. Since then, he has been a researcher at the Chair of Communication Networks at the University of Tuebingen, pursuing his PhD. His main research interests include software defined networking, P4, network security, and automated network management.

Michael Menth (Senior Member, IEEE) is a professor at the Department of Computer Science at the University of Tuebingen and Chairholder of Communication Networks. His special interests are performance analysis and optimization of communication networks, resilience and routing, resource and congestion management, network softwarization, Internet protocols, industrial networking, as well as Internet of Things.

2.2 A Survey on Data Plane Programming with P4: Fundamentals, Advances, and Applied Research

A Survey on Data Plane Programming with P4: Fundamentals, Advances, and Applied Research

Frederik Hauser[a], Marco Häberle[a], Daniel Merling[a], Steffen Lindner[a], Vladimir Gurevich[b], Florian Zeiger[c], Reinhard Frank[c], Michael Menth[a]

[a]*University of Tuebingen, Department of Computer Science, Chair of Communication Networks, Tuebingen, Germany*
[b]*Intel, Barefoot Division (BXD), United States of America*
[c]*Siemens AG, Corporate Technology, Munich, Germany*

Abstract

Programmable data planes allow users to define their own data plane algorithms for network devices including appropriate data plane application programming interfaces (APIs) which may be leveraged by user-defined software-defined networking (SDN) control. This offers great flexibility for network customization, be it for specialized, commercial appliances, e.g., in 5G or data center networks, or for rapid prototyping in industrial and academic research. Programming protocol-independent packet processors (P4) has emerged as the currently most widespread abstraction, programming language, and concept for data plane programming. It is developed and standardized by an open community, and it is supported by various software and hardware platforms.

In the first part of this paper we give a tutorial of data plane programming models, the P4 programming language, architectures, compilers, targets, and data plane APIs. We also consider research efforts to advance P4 technology. In the second part, we categorize a large body of literature of P4-based applied research into different research domains, summarize the contributions of these papers, and extract prototypes, target platforms, and source code availability. For each research domain, we analyze how the reviewed works benefit from P4's core features. Finally, we discuss potential next steps based on our findings.

Keywords: P4, SDN, programmable data planes

Email addresses: `frederik.hauser@uni-tuebingen.de` (Frederik Hauser), `marco.haeberle@uni-tuebingen.de` (Marco Häberle), `daniel.merling@uni-tuebingen.de` (Daniel Merling), `steffen.lindner@uni-tuebingen.de` (Steffen Lindner), `vladimir.gurevich@intel.com` (Vladimir Gurevich), `florian.zeiger@siemens.com` (Florian Zeiger), `reinhard.frank@siemens.com` (Reinhard Frank), `menth@uni-tuebingen.de` (Michael Menth)

1. Introduction

Traditional networking devices such as routers and switches process packets using data and control plane algorithms. Users can configure control plane features and protocols, e.g., via CLIs, web interfaces, or management APIs, but the underlying algorithms can be changed only by the vendor. This limitation has been broken up by SDN and even more by data plane programming.

SDN makes network devices programmable by introducing an API that allows users to bypass the built-in control plane algorithms and to replace them with self-defined algorithms. Those algorithms are expressed in software and typically run on an SDN controller with an overall view of the network. Thereby, complex control plane algorithms designed for distributed control can be replaced by simpler algorithms designed for centralized control. This is beneficial for use cases that are demanding with regard to flexibility, efficiency and security, e.g., massive data centers or 5G networks.

Programmable data planes enable users to implement their own data plane algorithms on forwarding devices. Users, e.g., programmers, practitioners, or operators, may define new protocol headers and forwarding behavior, which is without programmable data planes only possible for a vendor. They may also add data plane APIs for SDN control.

Data plane programming changes the power of the users as they can build custom network equipment without any compromise in performance, scalability, speed, or power on appropriate platforms. There are different data plane programming models, each with many implementations and programming languages. Examples are Click [1], VPP [2], NPL [3], and SDNet [4].

Programming protocol-independent packet processors (P4) is currently the most widespread abstraction, programming language, and concept for data plane programming. First published as a research paper in 2014 [5], it is now developed and standardized in the P4 Language Consortium, it is supported by various software- and hardware-based target platforms, and it is widely applied in academia and industry.

In the following, we clarify the contribution of this survey, point out its novelty, explain its organization, and provide a table with acronyms frequently used in this work.

1.1. Contributions

This survey pursues two objectives. First, it provides a comprehensive introduction and overview of P4. Second, it surveys publications describing applied research based on P4 technology. Its main contributions are the following:

- We explain the evolution of data plane programming with P4, relate it to prior developments such as SDN, and compare it to other data plane programming models.
- We give an overview of data plane programming with P4. It comprises the P4 programming language, architectures, compilers, targets, and data

plane APIs. These sections do not only include foundations but also present related work on advancements, extensions, or experiences.

- We summarize research efforts to advance P4 data planes. It comprises optimization of development and deployment, testing and debugging, research on P4 targets, and advances on control plane operation.
- We analyze a large body of literature considering P4-based applied research. We categorize 245 research papers into different application domains, summarize their key contributions, and characterize them with respect to prototypes, target platforms, and source code availability. For each research domain, we analyze how the reviewed works benefit from P4's core features.

We consider publications on P4 that were published until the end of 2020 and selected paper from 2021. Beside journal, conference, and workshop papers, we also include contents from standards, websites, and source code repositories. The paper comprises 519 references out of which 377 are scientific publications: 73 are from 2017 and before, 66 from 2018, 113 from 2019, 116 from 2020, and 9 from 2021.

1.2. Novelty

There are numerous surveys on SDN published in 2014 [6, 7], 2015 [8, 9, 10], and 2016 [11, 12] as well as surveys on OpenFlow (OF) from 2014 [13, 14, 15]. Only one of them [12] mentions P4 in a single sentence. Two surveys of data plane programming from 2015 [10, 9] were published shortly after the release of P4, one conference paper from 2018 [16] and a survey from 2019 [17] present P4 just as one among other data plane programming languages. Likewise, Michel et al. [18] gives an overview of data plane programming in general and P4 is one among other examined abstractions and programming languages. Our survey is dedicated to P4 only. It covers more details of P4 and a many more papers of P4-based applied research which have mostly emerged only within the last two years.

A recent survey focusing on P4 data plane programming has been published in [19]. The authors introduce data plane programming with P4, review 33 research works from four research domains, and discuss research issues. Another recent technical report [20] reviews 150 research papers from seven research domains. While typical research areas of P4 are covered, others (e.g., industrial networking, novel routing and forwarding schemes, and time-sensitive networking) are not part of the literature review. The different aspects of P4, e.g., the programming language, architectures, compilers, targets, data plane APIs, and their advancements are not treated in the paper. In addition, a survey solely focusing on P4 for network security [21] was recently published. Gao et al. introduce the P4 language and review 60 research works in the field of network security applications. They analyze the core idea of the reviewed literature and point out limitations. Finally, a short comparison on P4 targets regarding throughput, delay, jitter, resource constraints, flexibility and proportion in the research literature is given. In contrast to the mentioned surveys on P4, we cover a greater level of detail of P4 technology and their advancements, and our literature review is more comprehensive.

1.3. Paper Organization

Figure 1 depicts the structure of this paper which is divided into two main parts: an *overview of P4* and a *survey of research publications.*

In the first part, Section 2 gives an introduction to network programmability. We describe the development from traditional networking and SDN to data plane programming and present the two most common data plane programming models. In Section 3, we give a technology-oriented tutorial of P4 based on its latest version $P4_{16}$. We introduce the P4 programming language and describe how user-provided P4 programs are compiled and executed on P4 targets. Section 4 presents the concept of P4 architectures as intermediate layer between the P4 programs and the targets. We introduce the four most common architectures in detail and describe P4 compilers. In Section 5, we categorize and present platforms that execute P4 programs, so-called P4 targets that are based on software, FPGAs, ASICs, or NPUs. Section 6 gives an introduction to data plane APIs. We describe their functions, present a characterization, introduce the four main P4 data plane APIs that serve as interfaces for SDN controllers, and point out controller use case patterns. In Section 7, we summarize research efforts that aim to improve P4 data plane programming.

The second part of the paper surveys P4-based applied research in communication networks. In Section 8, we classify core features of P4 that make it attractive for the implementation of data plane algorithms. We use these properties in later sections to effectively reason about P4's value for the implementation of various prototypes. We present an overview of the research domains and compile statistics about the included publications. The superordinate research domains are monitoring (Section 9), traffic management and congestion control (Section 10), routing and forwarding (Section 11), advanced networking (Section 12), network security (Section 13), and miscellaneous (Section 14) to cover additional, different topics. Each category includes a table to give a quick overview of the analyzed papers with regard to prototype implementations, target platforms, and source code availability. At the end of each section, we analyze how the reviewed works benefit from P4's core features.

In Section 15 we discuss insights from this survey and give an outlook on potential next steps. Section 16 concludes this work.

1.4. List of Acronyms

The following acronyms are used in this paper.

ACL	access control list
ALU	arithmetic logic unit
API	application programming interface

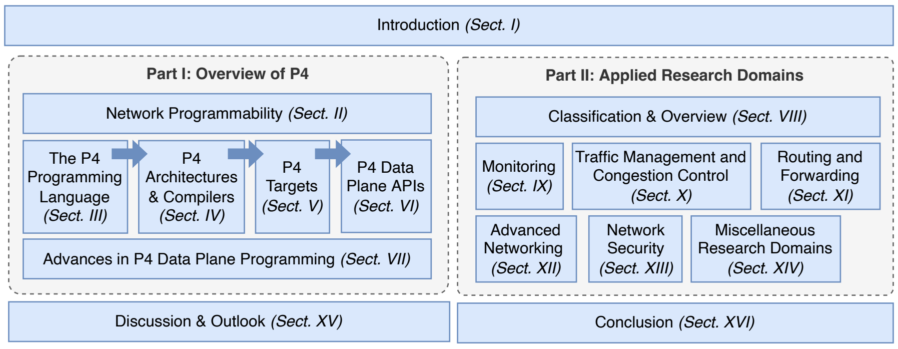

Introduction (Sect. I)
Part I: Overview of P4
Network Programmability (Sect. II)
The P4 Programming Language (Sect. III)
P4 Architectures & Compilers (Sect. IV)
P4 Targets (Sect. V)
P4 Data Plane APIs (Sect. VI)
Advances in P4 Data Plane Programming (Sect. VII)
Part II: Applied Research Domains
Classification & Overview (Sect. VIII)
Monitoring (Sect. IX)
Traffic Management and Congestion Control (Sect. X)
Routing and Forwarding (Sect. XI)
Advanced Networking (Sect. XII)
Network Security (Sect. XIII)
Miscellaneous Research Domains (Sect. XIV)
Discussion & Outlook (Sect. XV)
Conclusion (Sect. XVI)

Figure 1: The paper is organized in two parts: Part I gives an overview on P4; Part II reviews P4-based applied research in communication networks.

AQM	active queue management
ASIC	application-specific integrated circuit
AWW	adjusting advertised windows
bmv2	Behavioral Model version 2
BGP	Border Gateway Protocol
BPF	Berkeley Packet Filter
CLI	command line interface
DAG	directed acyclic graph
DDoS	distributed denial of service
DPI	deep packet inspection
DPDK	Data Plane Development Kit
DSL	domain-specific language
eBPF	Extended Berkeley Packet Filter
ECN	Explicit Congestion Notification
FPGA	field programmable gate array
FSM	finite state machine
GTP	GPRS tunneling protocol
HDL	hardware description language
HLIR	high-level intermediate representation
IDE	integrated development environment
IDL	Intent Definition Language
IDS	intrusion detection system
INT	in-band network telemetry

LDWG	Language Design Working Group
LPM	longest prefix matching
LUT	look up table
MAT	match-action-table
ML	machine learning
NDN	named data networking
NF	network function
NFP	network flow processing
NFV	network function virtualization
NIC	network interface card
NPU	network processing unit
ODM	original design manufacturer
ODP	Open Data Plane
OEM	original equipment manufacturer
OF	OpenFlow
ONF	Open Networking Foundation
OVS	Open vSwitch
PISA	Protocol Independent Switching Architecture
PSA	Portable Switch Architecture
REG	register
RPC	remote procedure call
RTL	register-transfer level
SDK	software development kit
SDN	software-defined networking
SF	service function
SFC	service function chain
SRAM	static random-access memory
TCAM	ternary content-addressable memory
TSN	Time-Sensitive Networking
TNA	Tofino Native Architecture
uBPF	user-space BPF
VM	virtual machine
VNF	virtual network function
VPP	Vector Packet Processors
WG	working group
XDP	eXpress Data Path

2. Network Programmability

In this section, we first define the notion of network programmability and related terms. Then, we discuss control plane programmability and data plane programming, elaborate on data plane programming models, and point out the benefits of data plane programming.

2.1. Definition of Terms

We define *programmability* as the ability of the software or the hardware to execute an externally defined processing algorithm. This ability separates programmable entities from *flexible* (or *configurable*) ones; the latter only allow changing different parameters of the internally defined algorithm which stays the same.

Thus, the term *network programmability* means the ability to define the processing algorithm executed in a network and specifically in individual processing nodes, such as switches, routers, load balancers, etc. It is usually assumed that no special processing happens in the links connecting network nodes. If necessary, such processing can be described as if it takes place on the nodes that are the endpoints of the links or by adding a "bump-in-the-wire" node with one input and one output.

Traditionally, the algorithms, executed by telecommunication devices, are split into three distinct classes: the data plane, the control plane, and the management plane. Out of these three classes, the management plane algorithms have the smallest effect on both the overall packet processing and network behavior. Moreover, they have been programmable for decades, e.g., SNMPv1 was standardized in 1988 and created even earlier than that. Therefore, management plane algorithms will not be further discussed in this section.

True network programmability implies the ability to specify and change both the control plane and data plane algorithms. In practice this means the ability of network operators (users) to define both data and control plane algorithms on their own, without the need to involve the original designers of the network equipment. For the network equipment vendors (who typically design their own control plane anyway), network programmability mostly means the ability to define data plane algorithms without the need to involve the original designers of the chosen packet processing application-specific integrated circuit (ASIC).

Network programmability is a powerful concept that allows both the network equipment vendors and the users to build networks ideally suited to their needs. In addition, they can do it much faster and often cheaper than ever before and without compromising the performance or quality of the equipment.

For a variety of technical reasons, different layers became programmable at different point in time. While the management plane became programmable in the 1980s, control plane programmability was not achieved until late 2000s to early 2010s and a programmable switching ASICs did not appear till the end of 2015.

Thus, despite the focus on data plane programmability, we will start by discussing control plane programmability and its most well-known embodiment, called software-defined networking (SDN). This discussion will also better prepare us to understand the significance of data plane programmability.

2.2. Control Plane Programmability and SDN

Traditional networking devices such as routers or switches have complex data and control plane algorithms. They are built into them and generally cannot be replaced by the users. Thus, the functionality of a device is defined by its vendor who is the only one who can change it. In industry parlance, vendors are often called original equipment manufacturers (OEMs).

Software-defined networking (SDN) was historically the first attempt to make the devices, and *specifically their control plane*, programmable. On selected systems, device manufacturers allowed users to bypass built-in control plane algorithms so that the users can introduce their own. These algorithms could then directly supply the necessary forwarding information to the data plane which was still non-replaceable and remained under the control of the device vendor or their chosen silicon provider.

For a variety of technical reasons, it was decided to provide an APIs that could be called remotely and that is how SDN was born. Figure 2 depicts SDN in comparison to traditional networking. Not only the control plane became programmable, but it also became possible to implement network-wide control plane algorithms in a centralized controller. In several important use cases, such as tightly controlled, massive data centers, these centralized, network-wide algorithms proved to be a lot simpler and more efficient, than the traditional algorithms (e.g. Border Gateway Protocol (BGP)) designed for decentralized control of many autonomous networks.

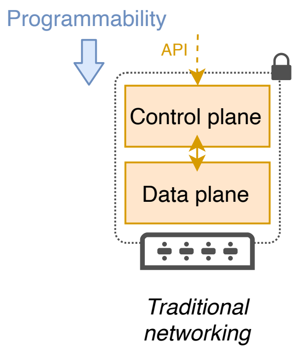
The effort to standardize this approach resulted in the development of OpenFlow (OF) [22]. The hope was that once OF standardized the messaging API to control the data plane functionality, SDN applications will be able to leverage the functions offered by this API to implement network control. There is a huge body of literature giving an overview of OF [13, 14, 15] and SDN [6, 7, 8, 9, 11, 10, 12].

However, it soon became apparent that OF assumed a specific data plane functionality which was not formally specified. Moreover, the specific data plane, that served as the basis for OF, could not be changed. It executed the sole, although relatively flexible, algorithm defined by the OF specifications.

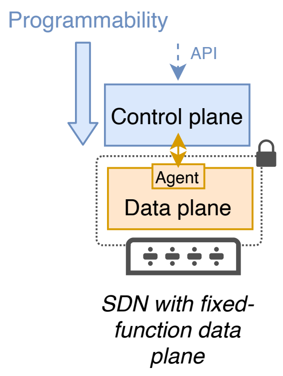
In part, it was this realization that led to the development of modern data plane programming that we discuss in the following section.

2.3. Data Plane Programming

As mentioned above, data plane programmability means that the data plane with its algorithms can be defined by the users, be they network operators or equipment designers working with a packet processing ASIC. In fact, data plane programmability existed during most of the networking industry history because data plane algorithms were typically executed on general-purpose CPUs. It is



(a) With traditional networking, programmability is limited to configuration of functionality via an API.



(b) SDN with fixed-function data planes allows full programmability of the control plane.

Figure 2: Distinction between traditional networking and SDN with fixed-function data planes.

only with the advent of high-speed links, exceeding the CPU processing capabilities, and the subsequent introduction of packet processing (switching) ASICs that data plane programmability (or lack thereof) became an issue.

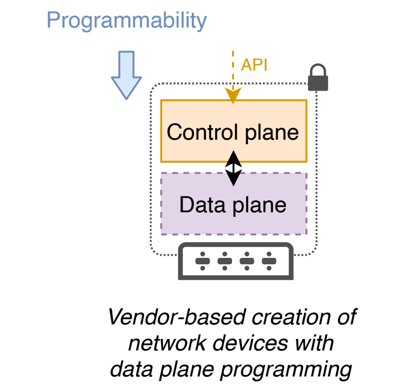
The data plane algorithms are responsible for processing all the packets that pass through a telecommunication system. Thus, they ultimately define the functionality, performance, and the scalability of such systems. Any attempt to implement data plane functionality in the control plane typically leads to significant performance degradation. When data plane programming is provided to users, it qualitatively changes their power. They can build custom network equipment without any compromise in performance, scalability, speed, or energy consumption.

For custom networks, new control planes and SDN applications can be designed and for them users can design data plane algorithms that fit them ideally. Data plane programming does not necessarily imply any provision of APIs for users nor does it require support for outside control planes as in OF. Device vendors might still decide to develop a proprietary control plane and use data plane programming only for their own benefit without necessarily making their systems more open (although many do open their systems now). Figure 3 visualizes both options.

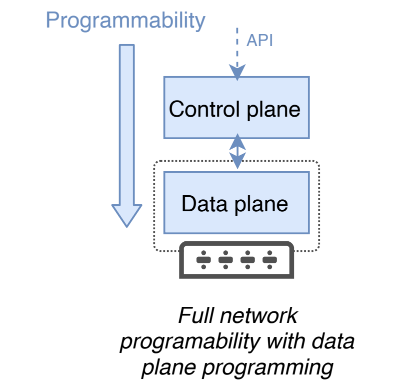
Four surveys from [10, 9, 16, 17] give an overview on data plane programming, but do not set a particular focus to P4.

2.4. Data Plane Programming Models

Data plane algorithms can and often are expressed using standard programming languages. However, they do not map very well onto specialized hardware such as high-speed ASICs. Therefore, several data plane models have been proposed as abstractions of the hardware. Data plane programming languages are tailored to those data plane models and provide ways to express algorithms


Programmability
API
Control plane
Data plane
Vendor-based creation of network devices with data plane programming

(a) Vendors utilize data plane programmability for more efficient development. Users can utilize only provided APIs to control the devices.



(b) Data plane programming is available to users. They can program the data plane and define new APIs through which they can control their devices.

Figure 3: Different usages of data plane programmability.

for them in an abstract way. The resulting code is then compiled for execution on a specific packet processing node supporting the respective data plane programming model.

Data flow graph abstractions and the Protocol Independent Switching Architecture (PISA) are examples for data plane models. We give an overview of the first and elaborate in-depths on the second as PISA is the data plane programming model for P4.

2.4.1. Data Flow Graph Abstractions

In these data plane programming models, packet processing is described by a directed graph. The nodes of the graph represent simple, reusable primitives that can be applied to packets, e.g., packet header modifications. The directed edges of the graph represent packet traversals where traversal decisions are performed in nodes on a per-packet basis. Figure 4 shows an exemplary graph for IPv4 and IPv6 packet forwarding.

Examples for programming languages that implement this data plane programming model are Click [1], Vector Packet Processors (VPP) [2], and BESS [23].

2.4.2. Protocol-Independent Switching Architecture (PISA)

Figure 5 depicts the PISA. It is based on the concept of a programmable match-action pipeline that well matches modern switching hardware. It is a generalization of reconfigurable match-action tables (RMTs) [24] and disaggregated reconfigurable match-action tables (dRMTs) [25].

PISA consists of a programmable parser, a programmable deparser, and a programmable match-action pipeline in between consisting of multiple stages.

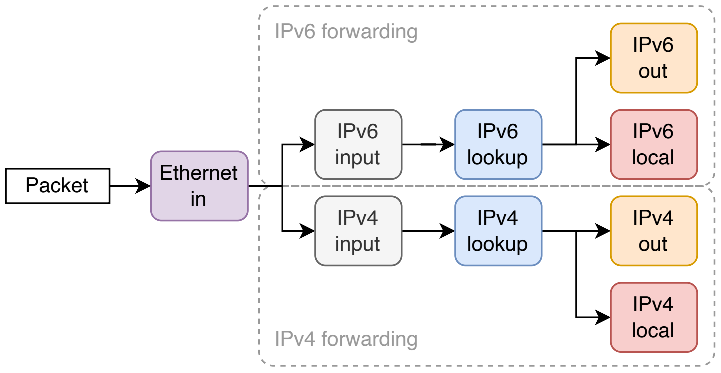


Figure 4: Example graph showing how data flow graph abstractions are applied to implement IPv4 and IPv6 forwarding.

- The *programmable parser* allows programmers to declare arbitrary headers together with a finite state machine that defines the order of the headers within packets. It converts the serialized packet headers into a well-structured form.
- The *programmable match-action pipeline* consists of multiple match-action units. Each unit includes one or more match-action-tables (MATs) to match packets and perform match-specific actions with supplied action data. The bulk of a packet processing algorithm is defined in the form of such MATs. Each MAT includes matching logic coupled with the memory (static random-access memory (SRAM) or ternary content-addressable memory (TCAM)) to store lookup keys and the corresponding action data. The action logic, e.g., arithmetic operations or header modifications, is implemented by arithmetic logic units (ALUs). Additional action logic can be implemented using stateful objects, e.g., counters, meters, or registers, that are stored in the SRAM. A control plane manages the matching logic by writing entries in the MATs to influence the runtime behavior.
- In the *programmable deparser*, programmers declare how packets are serialized.

A packet, processed by a PISA pipeline, consists of packet payload and packet metadata. PISA only processes packet metadata that travels from the parser all the way to the deparser but not the packet payload that travels separately.

Packet metadata can be divided into packet headers, user-defined and intrinsic metadata.

- *Packet headers* is metadata that corresponds to the network protocol headers. They are usually extracted in the parser, emitted in the deparser or both.

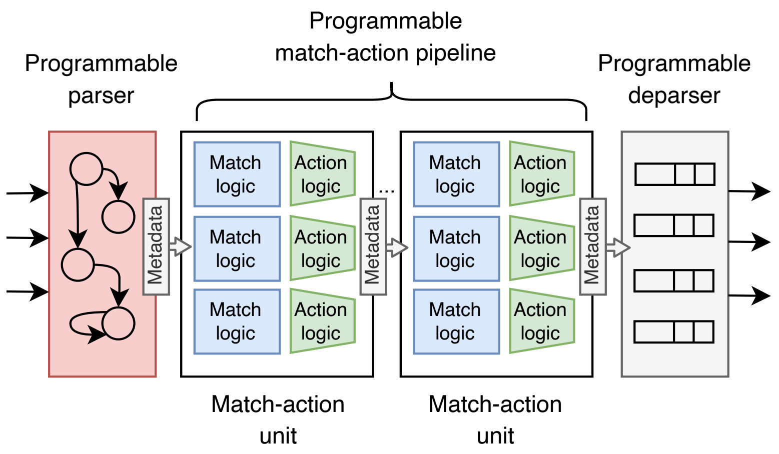


Figure 5: The Protocol-Independent Switch Architecture (PISA) contains a programmable parser, a programmable match-action pipeline, and a programmable deparser.

- *Intrinsic metadata* is metadata that relates to the fixed-function components. P4-programmable components may receive information from the fixed-function components by reading the intrinsic metadata they produce or control their behavior by setting the intrinsic metadata they consume.
- *User-defined metadata* (often referred as simply *metadata*) is a temporary storage, similar to local variables in other programming languages. It allows the developers to add information to packets that can be used throughout the processing pipeline.

All metadata, be it packet headers, user-defined or intrinsic metadata is *transient*, meaning that it is discarded when the corresponding packet leaves the processing pipeline (e.g., is sent out of an egress port or dropped).

PISA provides an abstract model that is applied in various ways to create concrete architectures. For example, it allows specifying pipelines containing different combinations of programmable components, e.g., a pipeline with no parser or deparser, a pipeline with two parsers and deparsers, and additional match-action pipelines between them. PISA also allows for specialized components that are required for advanced processing, e.g., hash/checksum calculations. Besides the programmable components of PISA, switch architectures typically also include configurable fixed-function components. Examples are ingress/egress port blocks that receive or send packets, packet replication engines that implements multicasting or cloning/mirroring of packets, and traffic managers, responsible for packet buffering, queuing, and scheduling.

The fixed-function components communicate with the programmable ones by generating and/or consuming intrinsic metadata. For example, the ingress port block generates ingress metadata that represents the ingress port number that might be used within the match-action units. To output a packet, the match-action units generates intrinsic metadata that represents an egress port number; this intrinsic metadata is then consumed by the traffic manager and/or egress port block.

Figure 6 depicts a typical switch architecture based on PISA. It comprises a programmable ingress and egress pipeline and three fixed-function components: an ingress block, an egress block, and a packet replication engine together with a traffic manager between ingress and egress pipeline.

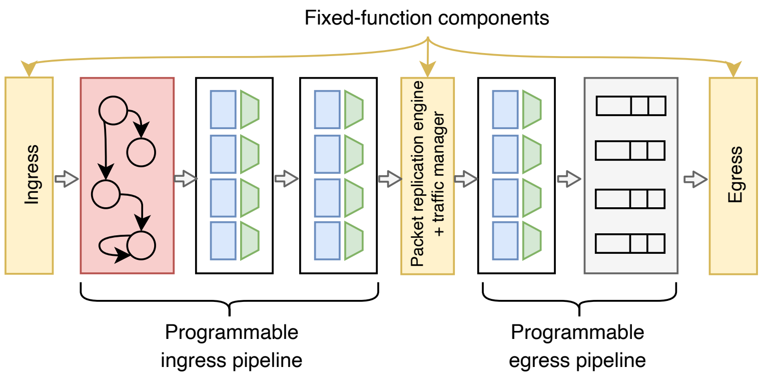

Fixed-function components
Ingress
Packet replication engine + traffic manager
Egress
Programmable ingress pipeline
Programmable egress pipeline

Figure 6: Exemplary switch architecture based on PISA with the ingress and egress pipeline as programmable parts. The ingress, the egress, the packet replication engine, and the traffic manager are fixed-function components.

P4 (Programming Protocol-Independent Packet Processors) [5] is the most widely used domain-specific programming language for describing data plane algorithms for PISA. Its initial idea and name were introduced in 2013 [26] and it was published as a research paper in 2014 [5]. Since then, P4 has been further developed and standardized by the P4 Language Consortium [27] that is part of the Open Networking Foundation (ONF) since 2019. The P4 Language Consortium is managed by a technical steering committee and hosts five working groups (WGs). $P4_{14}$ [28] was the first standardized version of the language. The current specification is $P4_{16}$ [29] which was first introduced in 2016.

Other data plane programming languages for PISA are FAST [30], Open-State [31], Domino [32], FlowBlaze [33], Protocol-Oblivious Forwarding [34], and NetKAT [35]. In addition, Broadcom [3] and Xilinx [4] offer vendor-specific programmable data planes based on match-action tables.

2.5. Benefits

Data plane programmability entails multiple benefits. In the following, we summarize key benefits.

Data plane programming introduces full flexibility to network packet processing, i.e., algorithms, protocols, features can be added, modified, or removed by the user. In addition, programmable data planes can be equipped with a user-defined API for control plane programmability and SDN. To keep complexity low, only components needed for a particular use case might be included in the code. This improves security and efficiency compared to multi-purpose appliances.

In conjunction with suitable hardware platforms, data plane programming allows network equipment designers and even users to experiment with new protocols and design unique applications; both do no longer depend on vendors of specialized packet-processing ASICs to implement custom algorithms. Compared to long development circles of new silicon-based solutions, new algorithms can be programmed and deployed in a matter of days.

Data plane programming is also beneficial for network equipment developers that can easily create differentiated products despite using the same packet processing ASIC. In addition, they can keep their know-how to themselves without the need to share the details with the ASIC vendor and potentially disclose it to their competitors that will use the same ASIC.

So far, modern data plane programs and programming languages have not yet achieved the degree of portability attained by the general-purpose programming languages. However, expressing data plane algorithms in a high-level language has the potential to make telecommunication systems significantly more target-independent. Also, data plane programming does not require but encourages full transparency. If the source code is shared, all definitions for protocols and behaviors can be viewed, analyzed, and reasoned about, so that data plane programs benefit from community development and review. As a result, users could choose cost-efficient hardware that is well suited for their purposes and run their algorithms on top of it. This trend has been fueled by SDN and is commonly known as network disaggregation.

2.6. Differences Between SDN and P4

SDN introduces *programmability on the control plane.* SDN-capable network devices such as switches include an API allowing that the device-local control plane can be substituted by an external, software-based control plane. This control plane comprises control plane algorithms managing the data plane. The centralized view of an external controller facilitates the implementation of simpler algorithms that may replace complex distributed protocols from legacy network devices. The control plane leverages an API offered by the data plane devices for control. The data plane however merely features fixed functions that can be used and configured by the control plane.

In contrast, P4 is a domain-specific language for data plane programming, i.e., *programmability is extended to the data plane.* Instead of supporting fixed functions only, the functionality of the data plane devices is described by a P4 program that is compiled into target-specific code that can be executed by the programmable network hardware. While the P4 language itself focuses on data plane programmability, P4 targets typically offer APIs so that software-based SDN control planes can manage the runtime behavior of those data plane devices.

3. The P4 Programming Language

We give an overview of the P4 programming language. We briefly recap its specification history and describe how P4 programs are deployed. We introduce the P4 processing pipeline and data types. We discuss parsers, match-action controls, and deparsers. Finally, we give an overview of tutorials and guides to P4.

3.1. Specification History

The P4 Language Design Working Group (LDWG) of the P4 Language Consortium has standardized so far two distinct standards of P4: $P4_{14}$ and $P4_{16}$. Table 1 depicts their specification history.

Table 1: Specification history of $P4_{14}$ and $P4_{16}$.

$P4_{14}$	
Version 1.0.2	03/2015
Version 1.1.0	01/2016
Version 1.0.3	11/2016
Version 1.0.4	05/2017
Version 1.0.5	11/2018

$P4_{16}$	
Version 1.0.0	05/2017
Version 1.1.0	11/2018
Version 1.2.0	11/2018
Version 1.2.1	06/2020

The $P4_{14}$ programming language dialect allows the programmers to describe data plane algorithms using a combination of familiar, general-purpose imperative constructs and more specialized declarative ones that provide support for the typical data-plane-specific functionality, e.g., counters, meters, checksum calculations, etc. As a result, the $P4_{14}$ language core includes more than 70 keywords. It further assumed a specific pipeline architecture based on PISA.

Table 2: Core differences between $P4_{14}$ and $P4_{16}$.

	$P4_{14}$	**$P4_{16}$**
Modularity	-	✓
Pipeline architectures	single	multiple
Target-specific functions	-	✓
# of language keywords	>70	<40
Strict typing	-	✓
Nested data structures	-	✓
Declarative constructs	✓	-

$P4_{16}$ has been introduced to address several $P4_{14}$ limitations that became apparent in the course of its use. Those include the lack of means to describe various targets and architectures, weak typing and generally loose semantics (caused, in part, by the above-mentioned mix of imperative and declarative programming constructs), relatively low-level constructs, and weak support for program modularity. The core differences between $P4_{14}$ and $P4_{16}$ are summarized in Table 2.

Support for multiple different targets and pipeline architecture is the major contribution of the $P4_{16}$ standard and is achieved by separating the core language from the specifics of a given architecture, thus making it architecture-agnostic. The structure, capabilities and interfaces of a specific pipeline are now encapsulated into an architecture description, while the architecture- or target-specific functions are accessible through an architecture library, typically provided by the target vendor. The core components are further structured into a small set of language constructs and a core library that is useful for most P4 programs. Compared to $P4_{14}$, $P4_{16}$ introduced strict typing, expressions, nested data structures, several modularity mechanisms, and also removed declarative constructs, making it possible to better reason about the programs, written in the language. Figure 7 illustrates the concept which is subdivided into core components and architecture components.

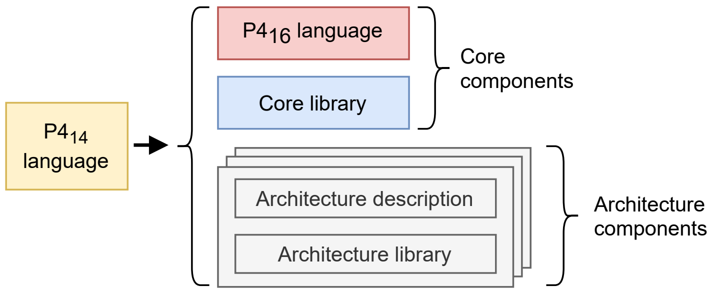


Figure 7: Evolvement from the $P4_{14}$ programming language to the $P4_{16}$ language (similar to [29]). $P4_{14}$ comprised all components as part of the programming language. In $P4_{16}$, the different parts of the programming language are split into core components and architecture components.

Due to the obvious advantages of $P4_{16}$, $P4_{14}$ development has been discontinued, although it is still supported on a number of targets. Therefore, we focus on $P4_{16}$ in the remainder of this paper where P4 implicitly stands for $P4_{16}$.

3.2. Development and Deployment Process

Figure 8 illustrates the development and deployment process of P4 programs.

P4-programmable nodes, so-called P4 targets, are available as software or specialized hardware (see Section 5). They feature packet processing pipelines consisting of both P4-programmable and fixed-function components. The exact structure of these pipelines is target-specific and is described by a corresponding P4 architecture model (see Section 4) which is provided by the manufacturer of the target.

P4 programs are supplied by the user and are implemented for a particular P4 architecture model. They define algorithms that will be executed by the P4-programmable components and their interaction with the ones implemented in the fixed-function logic. The composition of the P4 programs and the fixed-function logic constitutes the full data plane algorithm.

P4 compilers (see Section 4) are also provided by the manufacturers. They translate P4 programs into target-specific code which is loaded and executed by the P4 target.

The P4 compiler also generates a data plane API that can be used by a user-supplied control plane (see Section 6) to manage the runtime behavior of the P4 target.

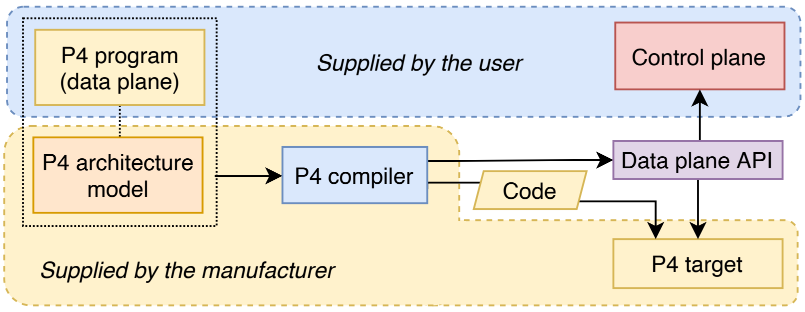


Figure 8: P4 deployment process (similar to [29]): A P4 compiler transforms a P4 program formulated for a particular P4 architecture model into code which is executed by a P4 target. The code provides a data plane API which can be leveraged by a user-supplied control plane.

3.3. Information Flow

P4$_{16}$ adopts PISA's concept of packet metadata. Figure 9 illustrates the information flow in the P4 processing pipeline. It comprises different blocks, where packet metadata (be it headers, user-defined or intrinsic metadata) is used to pass the information between them, therefore representing a uniform interface.

The parser splits up the received packet into individual headers and the remaining payload. Intrinsic metadata from the ingress block, e.g., the ingress port number or the ingress timestamp, is often provided by the hardware and can be made available for further processing. Many targets allow the user metadata to be initialized in the parser as well. Then, the headers and metadata are passed to the match-action pipeline that consists of one or more match-action units. The remaining payload travels separately and cannot be directly affected by the match-action pipeline processing.

While traversing the individual match-action pipeline units, the headers can be added, modified, or removed and additional metadata can be generated.

The deparser assembles the packet back by emitting the specified headers followed by the original packet payload. Packet output is configured with intrinsic metadata that includes information such as a drop flag, desired egress port, queue number, etc.

3.4. Data Types

P4$_{16}$ is a statically typed language that supports a rich set of data types for data plane programming.

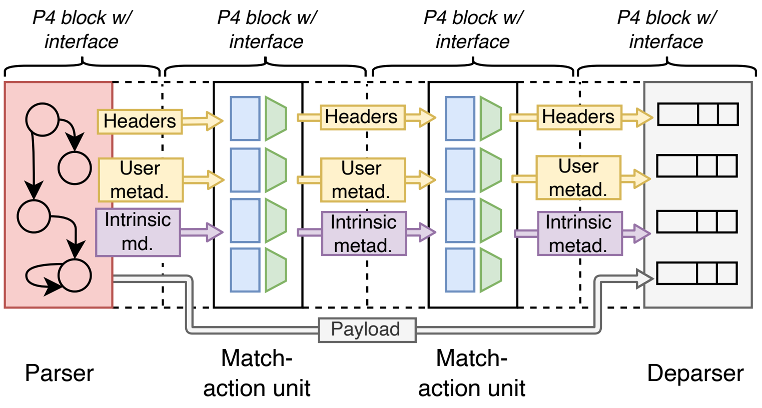


Figure 9: Information flow in the P4 processing pipeline. Metadata (headers, user metadata, intrinsic metadata) transport information between the different P4 blocks of the processing pipeline.

3.4.1. Basic Data Types

P4$_{16}$ includes common basic types such as Boolean (`bool`), signed (`int`), and unsigned (`bit`) integers which are also known as bit strings. Unlike many common programming languages, the size of these integers is specified at *bit* granularity, with a wide range of supported widths. For example, types such as `bit<1>`, `int<3>`, `bit<128>` and wider are allowed.

In addition, P4 supports bit strings of variable width, represented by a special varbit type. For example, IPv4 options can be represented as `varbit<320>` since the size of IPv4 options ranges from zero to 10 32-bit words.

P4$_{16}$ also supports enumeration types that can be serializable (with the actual representation specified as `bit<N>` or `int<N>` during the type definition) or non-serializable, where the type representation is chosen by the compiler and hidden from the user.

3.4.2. Derived Data Types

Basic data types can be composed to construct derived data types. The most common derived data types are `header`, `header stack`, and `struct`.

The `header` data type facilitates the definition of packet protocol headers, e.g., IPv4 or TCP. A header consists of one more fields of the serializable types described above, typically `bit<N>`, serializable `enum`, or `varbit`. A header also has an implicit validity field indicating whether the header is part of a packet. The field is accessible through standard methods such as *setvalid()*, *setInvalid()*, and *isValid()*. Packet parsing starts with all headers being invalid. If the parser determines that a header is present in the packet, the header fields are extracted and the header's validity field is set valid. The standard packet *emit()* method used by a deparser equips packets only with valid headers. Thus, P4 programs can easily add and remove headers by manipulating their validity bits. A sample header declaration is shown in Figure 10.

A `header stack` is used to define repeating headers, e.g., VLAN tags or MPLS labels. It supports special operations allowing headers to be "pushed" onto the stack or "popped" from it.

`Struct` in P4 is a composed data type similar to structs in programming languages like C. Unlike the `header` data type, they can contain fields of any type including other structs, headers, and others.

```
typedef bit<48> macAddr_t;

header ethernet_t {
    macAddr_t dstAddr;
    macAddr_t srcAddr;
    bit<16>   etherType;
}
```

Figure 10: Sample declaration of the Ethernet header with the help of a type definition for the MAC addresses used in the header.

3.5. Parsers

Parsers extract header fields from ingress packets into header data and metadata. P4 does not include predefined packet formats, i.e., all required header formats including parsing mechanisms need to be part of the P4 program. Parsers are defined as finite state machine (FSM) with an explicit *Start* state, two ending states (*Accept* and *Reject*), and custom states in between.

Figure 11 depicts the structure of a typical P4 parser for Ethernet, MPLS, IPv4, TCP, and UDP headers. Figure 12 shows the source code fragment of the example parser in a P4$_{16}$ program. The process starts in the *Start* state and switches to the *Ethernet* state. In this state and the following states, information from the packet headers is extracted according to the defined header structure.

State transitions may be either conditional or unconditional. In the given example, the transition from the *Start* state to the *Ethernet* state is unconditional while in the *Ethernet* state the transition to the *MPLS*, *IPv4*, or *Reject* state depends on the value of the *EtherType* field of the extracted Ethernet header. Based on previously parsed header information, any number of further headers can be extracted from the packet. If the header order does not comply with the expected order, a packet can be discarded by switching to the *Reject* state. The parser can also implicitly transition into the *Reject* state in case of a parser exception, e.g., if a packet is too short.

3.6. Match-Action Controls

Match-action controls express the bulk of the packet processing algorithm and resemble traditional imperative programs. They are executed after successful parsing of a packet. In some architectures they are also called match-action pipeline units. In the following, we give an overview of control blocks, actions, and match-action tables.

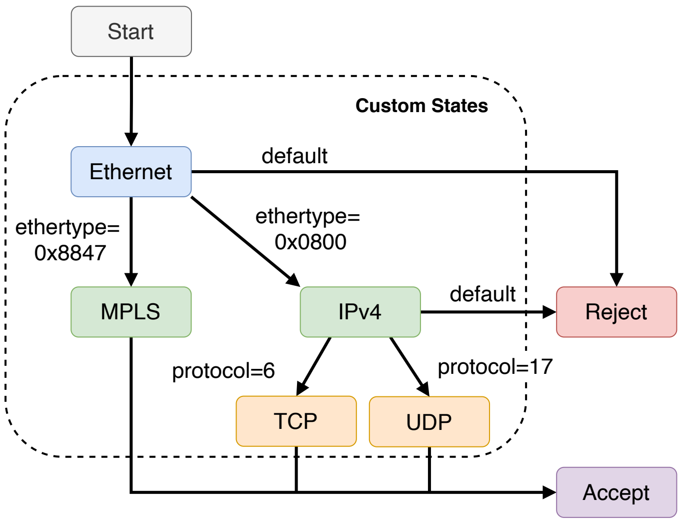


Figure 11: Example for the FSM of a P4 parser that parses packets with Ethernet, MPLS, IPv4, TCP, and UDP headers.

3.6.1. Control Blocks

Control blocks, or just `controls`, are similar to functions in general-purpose languages. They are called by an *apply()* method. They have parameters and can call also other control blocks. The body of a control block contains the definition of resources, such as tables, actions, and externs that will be used for processing. Furthermore, a single *apply()* method is defined that expresses the processing algorithm.

P4 offers statements to express the program flow within a control block. Unlike common programming languages, P4 does not provide any statements that would allow the programmer to create loops. This ensures that all the algorithms that can be coded in P4 can be expressed as directed acyclic graphs (DAGs) and thus are guaranteed to complete within a predictable time interval. Specific control statements include:

- a block statement `{}` that expresses sequential execution of instructions.
- an `if()` statement that expresses an execution predicated on a Boolean condition
- a `switch()` statement that expresses a choice from multiple alternatives
- an `exit()` statement that ends the control flow within a control block and passes the control to the end of the top-level control

Transformations are performed by several constructs, such as

- An assignment statement which evaluates the expression on its right-hand-side and assigns the result to a header or a metadata fields

```
parser SampleParser(packet_in p, out headers h) {

    state start {
        transition parse_ethernet;
    }

    state parse_ethernet {
        p.extract(h.ethernet);
        transition select(h.ethernet.etherType) {
             0x8847: parse_mpls;
             0x0800: parse_ipv4;
            default: reject;
        };
    }

    state parse_ipv4 {
        p.extract(h.ipv4);
        transition select(h.ipv4.protocol) {
                  6: parse_tcp;
                 17: parse_udp;
            default: accept;
        }
    }

    state parse_udp {
        p.extract(h.udp);
        transition accept;
    }
    /* Other states follow */
}
```

Figure 12: Sample parser implementation of the FSM in Figure 11.

- A match-action operation on a table expressed as the table's `apply()` method
- An invocation of an action or a function that encapsulate a sequence of statements
- An invocation of an extern method that represents special, target- and architecture-specific processing, often involving additional state, preserved between packets

A sample implementation of basic L2 forwarding is provided in Figure 13.

3.6.2. Actions

Actions are code fragments that can read and write packet headers and metadata. They work similarly to functions in other programming languages but have no return value. Actions are typically invoked from MATs. They can receive parameters that are supplied by the control plane as action data in MAT entries.

```
control SampleControl(inout headers h, inout standard_metadata_t
    standard_metadata) {

    action l2_forward(egressSpec_t port) {
        standard_metadata.egress_spec = port;
    }

    table l2 {
        key = {
            h.ethernet.dstAddr: exact;
        }
        actions = {
            l2_forward; drop;
        }
        size = 1024;
        default_action = drop();
    }

    apply {
        if (h.ethernet.isValid()) {
            l2.apply();
        }
    }
}
```

Figure 13: Sample control block implementing basic L2 forwarding.

As in most general-purpose programming languages, the operations are written using expressions and the results are then assigned to the desired header or metadata fields. The operations available in P4 expressions include standard arithmetic and logical operations as well as more specialized ones such as bit slicing (`field[high:low]`), bit concatenation (`field1 ++ field2`), and saturated arithmetic (`|+|` and `|-|`).

Actions can also invoke methods of other objects, such as headers and architecture-specific externs, e.g., counters and meters. Other actions can also be called, similar to nested function calls in traditional programming languages.

Action code is executed sequentially, although many hardware targets support parallel execution. In this case, the compiler can optimize the action code for parallel execution as long as its effects are the same as in case of the sequential execution.

3.6.3. Match-Action Tables (MATs)

MATs are defined within control blocks and invoke actions depending on header and metadata fields of a packet. The structure of a MAT is declared in the P4 program and its table entries are populated by the control plane at runtime. A packet is processed by selecting a matching table entry and invoking the corresponding action with appropriate parameters.

The declaration of a MAT includes the match key, a list of possible actions, and additional attributes.

The match key consists of one or more header or metadata fields (variables), each with the assigned *match type*. The P4 core library defines three standard match types: exact, ternary, and longest prefix matching (LPM). P4 architectures may define additional match types, e.g., the *v1model* P4 architecture extends the set of standard match types with the range and selector match.

The list of possible actions includes the names of all actions that can be executed by the table. These actions can have additional, directional parameters which are provided as action data in table entries.

Additional attributes may include the size of the MAT, e.g., the maximum number of entries that can be stored in a table, a default action for a miss, or static table entries.

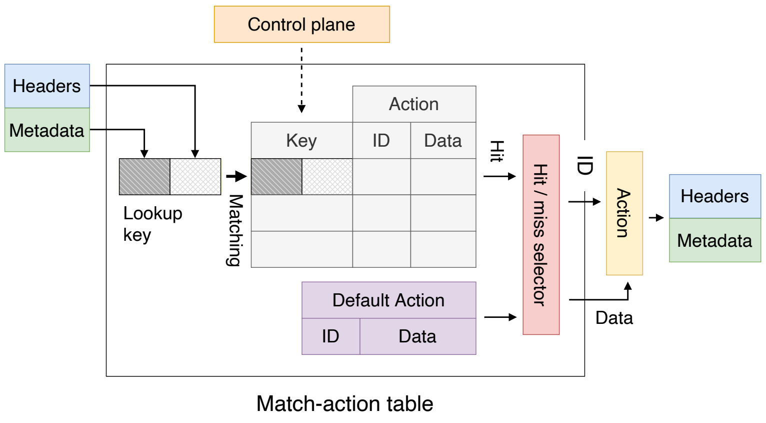


Figure 14: Structure of MATs in P4. Lookup keys are constructed based on packet metadata and used for row matching in the MAT. In case of a hit, the defined action is applied with the specified action data. In case of a miss, the default action is applied.

Figure 14 illustrates the principle of MAT operation. The MAT contains entries with values for match keys, the ID of the corresponding action to be invoked, and action data that serve as parameters for action invocation. For each packet, a lookup key is constructed from the set of header and metadata fields specified in the table definition. It is matched against all entries of the MAT using the rules associated with the individual field's match type. When the first match in the table is found, the corresponding action is called and the action data are passed to the action as directionless parameters. If no match is found in the table, a default action is applied.

As a special case, tables without a specified key always invoke the default action.

3.7. Deparser

The deparser is also defined as a control block. When packet processing by match-action control blocks is finished, the deparser serializes the packet. It reassembles the packet header and payload back into a byte stream so that the packet can be sent out via an egress port or stored in a buffer. Only valid headers are emitted, i.e., added to the packet. Thus, match-action control blocks can easily add and remove headers by manipulating their validity. Figure 15 provides a sample implementation.

```
control SampleDeparser(packet_out p, in headers h) {
    apply {
        p.emit(h.ethernet);
        p.emit(h.mpls);
        p.emit(h.ipv4);
        /* Normally, a packet can contain either
         * a TCP or a UDP header (or none at all),
         * but should never contain both
         */
        p.emit(h.tcp);
        p.emit(h.udp);
    }
}
```

Figure 15: Sample deparser implementation.

3.8. P4 Tutorials

The P4 Language Consortium provides a GitHub repository with simple programming exercises and a development VM containing all required software [36]. A guide on GitHub lists useful information for P4 newcomers, e.g. demo programs, information about other GitHub repositories, and an overview of P4 [37]. The Networked Systems Group at ETH Zürich provides resources for people who want to learn programming in P4, including lecture slides, references to useful documentation, examples and exercises [38].

4. P4 Architectures & Compilers

We present $P4_{16}$ architectures and introduce P4 compilers.

4.1. $P4_{16}$ Architectures

We summarize the concept of $P4_{16}$ architectures, describe externs, and give an overview of the most common $P4_{16}$ architectures.

4.1.1. Concept

As described before, $P4_{16}$ introduces the concept of P4 architectures as an intermediate layer between the core P4 language and the targets. A P4 architecture serves as programming models that represents the capabilities and the logical view of a target's P4 processing pipeline. P4 programs are developed for a specific P4 architecture. Such programs can be deployed on all targets that implement the same P4 architecture. The manufacturers of P4 targets provide P4 compilers that compile architecture-specific P4 programs into target-specific configuration binaries.

4.1.2. Externs

P4 architectures may provide additional functionalities that are not part of the P4 language core. Examples are checksum or hash computation units, random number generators, packet and byte counters, meters, registers, and many others. To make such extern functionalities usable, P4$_{16}$ introduces so-called *externs*.

Most of the externs have to be explicitly instantiated in P4 programs using their constructor method. The other methods provided by these externs can then be invoked on the given extern instance. Other externs (extern functions) do not require explicit instantiating.

Along with tables and value sets, P4 externs are allowed to preserve additional state between packets. That state may be accessible by the control plane, the data plane, or both. For example, the counter extern would preserve the number of packets or bytes that has been counted so that each new packet can properly increment it. The specifics of the state depend on the nature of the extern and cannot be specified in the language; this is done inside the vendor-specific API definitions.

While the P4 processing pipeline only allows packet header manipulation, extern functions may operate on packet payload as well.

4.1.3. Overview of Common P4$_{16}$ Architectures

We describe the four most common P4$_{16}$ architectures.

v1model. The v1model mimics the processing pipeline of P4$_{14}$. As depicted in Figure 16, it consists of a programmable parser, an ingress match action pipeline, a traffic manager, an egress match-action pipeline, and a deparser. It enables developers to convert P4$_{14}$ programs into P4$_{16}$ programs. Additional functionalities tracking the development of the reference P4 software switch Behavioral Model version 2 (bmv2) (see Section 5) are continuously added. All P4 examples in this paper are written using v1model.

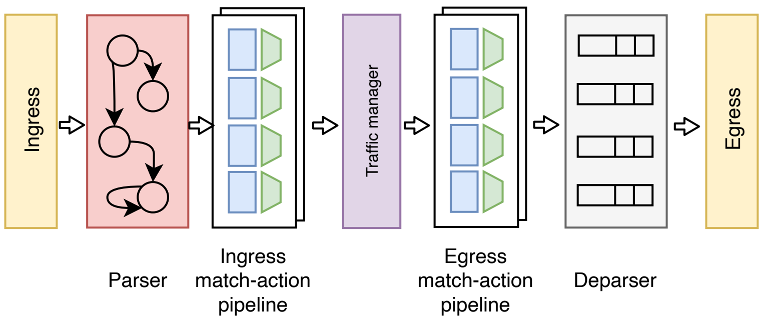


Figure 16: *v1model* architecture with a programmable parser, programmable ingress and egress match-action pipelines with a traffic manager in between, and a programmable parser.

Portable Switch Architecture (PSA). The PSA is a P4 architecture created and further developed by the Architecture WG [39] in the P4 Language Consortium. Besides, the WG also discusses standard functionalities, APIs, and externs that every target mapping the PSA should support. Its last specification is Version 1.1 [40] from November 2018. Figure 17 illustrates the P4 processing pipeline of the PSA. It is divided into an ingress and egress pipeline. Each pipeline consists of the three programmable parts: parser, multiple control blocks, and deparser. The architecture also defines configurable fixed-function components.

PSA specifies several packet processing primitives, such as:

- Sending a packet to an unicast port
- Dropping a packet
- Sending the packet to a multicast group
- Resubmitting a packet, which moves the currently processed packet from the end of the ingress pipeline to the beginning of the ingress pipeline for the purpose of packet re-parsing
- Recirculating a packet, which moves the currently processed packet from the end of the egress pipeline to the beginning of the ingress pipeline for the purposes of recursive processing, e.g., tunneling
- Cloning a packet, which duplicates the currently processed packet. *Clone ingress to egress (CI2E)* creates a duplicate of the ingress packet at the end of the ingress pipeline. *Clone egress to egress (CE2E)* creates a duplicate of the deparsed packet at the end of the egress pipeline. In both cases, cloned instances start processing at the beginning of the egress pipeline. Cloning can be helpful to implement powerful applications such as mirroring and telemetry.

SimpleSumeArchitecture. The SimpleSumeArchitecture is a simplified P4 architecture that is implemented by FPGA-based P4 targets. As depicted in Figure 18, it features a parser, a programmable match-and-action pipeline, and a deparser.

Tofino Native Architecture (TNA). TNA is a proprietary $P4_{16}$ architecture designed for Intel Tofino switching ASICs (see Section 5.3). Intel has published the architecture definitions and allows developers to publish programs written by using it.

The architecture describes a very high-performance, "industry-strength" device that is relatively complex. The basic programming unit is a so-called `Pipeline()` package that resembles an extended version of the Portable Switch Architecture (PSA) pipeline and consists of 6 top-level programmable components: the ingress parser, ingress match-action control, ingress deparser, and their egress counterparts. Since Tofino devices can have two or four processing

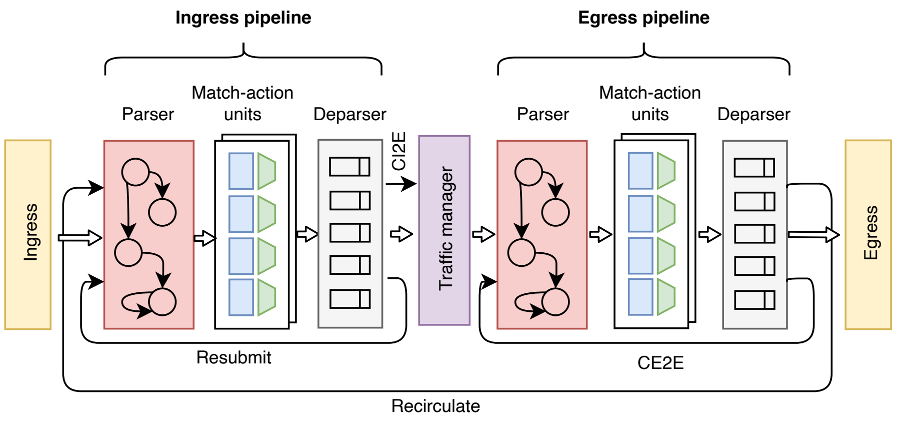


Figure 17: *Portable Switch Architecture (PSA)* with an ingress and egress pipeline and a traffic manager in between. Both include a programmable parser, programmable match-action units, a programmable deparser, fixed-function parts, and special packet processing primitives.

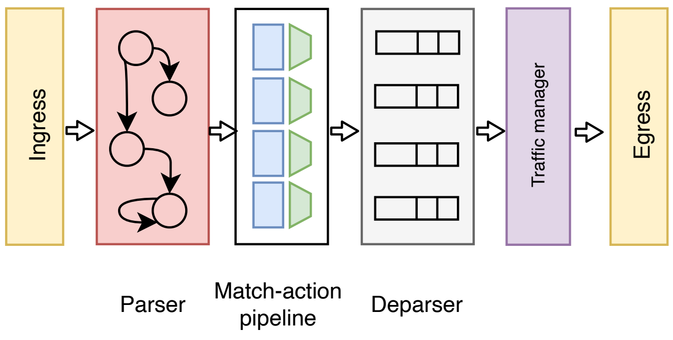


Figure 18: *SimpleSumeArchitecture* with a programmable parser, a programmable match-action pipeline, and a programmable parser followed by a traffic manager.

pipelines, the final switch package can be formed anywhere from one to four distinct pipeline packages. More complex versions of the `Pipeline()` package allow the programmer to specify different parsers for different ports.

TNA also provides a richer set of externs compared to most other architectures. Most notable is TNA `RegisterAction()` which represents a small code fragment that can be executed on the register instead of simple read/write operations provided in other architectures. TNA provides a clear and consistent interface for mirroring and resubmit with additional metadata being passed via the packet byte stream. The same technique is also used to pass intrinsic metadata which greatly simplifies the design.

Additional externs that are not present in other architectures include low-pass filters, weighted random early discard externs, powerful hash externs that can compute CRC based on user-defined polynomials, ParserCounter, and others.

The set of intrinsic metadata in Tofino is also larger than in most other P4 architectures as presented before. Notable is support for two-level multicasting with additional source pruning, copy-to-cpu functionality, and support for IEEE 1588.

4.2. P4 Compiler

P4 compilers translate P4 programs into target-specific configuration binaries that can be executed on P4 targets. We first explain compilers based on the two-layer model which are most widely in use. Then we mention other compilers in less detail.

4.2.1. Two-Layer Compiler Model

Most P4 compilers use the two-layer model, consisting of a common frontend and a target-specific backend.

The frontend is common for all the targets and is responsible for parsing, syntactic and target-independent semantic analysis of the program. The program is finally transformed into an intermediate representation (IR) that is then consumed by the target-specific backend which performs target-specific transformations.

The first-generation P4 compiler for $P4_{14}$ was written in Python and used the so-called high-level intermediate representation (HLIR) [41] that represented $P4_{14}$ program as a tree of Python objects. The compiler is referred to as p4-hlir.

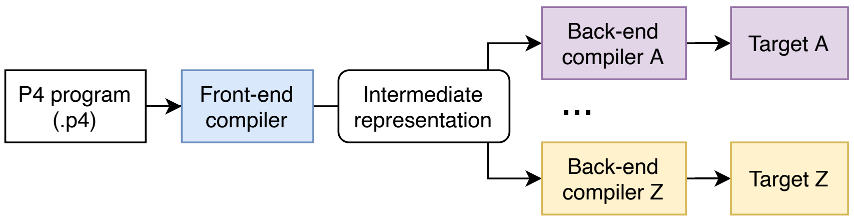


Figure 19: Structure and operation principle of P4 compilers using the two-layer model. The front-end compiler translates the given P4 program into an intermediate representation that is then compiled into target-specific code by back-end compilers.

The new P4 compiler (p4c) [42] is written in C++ and uses C++-object-based IR. As an additional benefit, the IR can be output as a $P4_{16}$ program or a JSON file. The latter allows the developers and users to build powerful tools for program analysis without the need to augment the compiler. Figure 19 visualizes its structure and operating principle. The compiler consists of a generic frontend that accepts both $P4_{14}$ and $P4_{16}$ code which may be written for any architecture. It furthermore has several reference backends for the bmv2, eBPF, and uBPF P4 targets as well as a backend for testing purposes and a backend that can generate graphs of control flows of P4 programs. In addition, p4c provides the so-called “mid-end” which is a library of generic transformation passes that are used by the reference backends and can also be used by vendor-specific backends. The compiler is developed and maintained by P4.org.

P4 target vendors design and maintain their own compilers that include the common frontend. This ensures the uniformity of the language which is accepted by different compilers.

4.2.2. Other Compilers

MACSAD [43] is a compiler that translates P4 programs into Open Data Plane (ODP) [44] programs. Jose et al. [45] introduce a compiler that maps P4 programs to FlexPipe and RMT, two common software switch architectures. P4GPU [46] is a multistage framework that translates a P4 program into intermediate representations and other languages to eventually generate GPU code.

5. P4 Targets

We describe P4 targets based on software, FPGA, ASIC, and NPU. Table 3 compiles an overview of the targets, their supported architectures, and the current state of development.

5.1. Software-Based P4 Targets

Software-based P4 targets are packet forwarding programs that run on a standard CPU. We describe the 9 software-based P4 targets mentioned in Table 3.

5.1.1. p4c-behavioural

p4c-behavioral [47] is a combined P4 compiler and P4 software target. It was introduced with the first public release of P4. p4c-behavioral translates the given $P4_{14}$ program into an executable C program.

5.1.2. Behavioral Model version 2 (bmv2)

The second version of the P4 software switch Behavioral Model (bmv2) [48] was introduced to address the limitations of p4c-behavioural (see also [49]). In contrast to p4c-behavioral, the source code of bmv2 is static and independent of P4 programs. P4 programs are compiled to a JSON representation that is loaded onto the bmv2 during runtime. External functions and other extensions can be added by extending bmv2's C++ source code. bmv2 is not a single target, but a collection of targets [50]:

- *simple_switch* is the bmv2 target with the largest range of features. It contains all features from the $P4_{14}$ specification and supports the v1model architecture of $P4_{16}$. simple_switch includes a program-independent Thrift API for runtime control.
- *simple_switch_grpc* extends simple_switch by the P4Runtime API that is based on gRPC (see Section 6.3.1).

Table 3: Overview of P4 targets.

Target	P4 Version	$P4_{16}$ Architecture	Active Development
Software			
p4c-behavioral	$P4_{14}$	n.a.	X
bmv2	$P4_{14}$, $P4_{16}$	v1model, psa	✓
eBPF	$P4_{16}$	ebpf_model.p4	✓
uBPF	$P4_{16}$	ubpf_model.p4	✓
XDP	$P4_{16}$	xdp_model.p4	✓
T4P4S	$P4_{14}$, $P4_{16}$	v1model, psa	✓
Ripple	n.a	n.a	n.a
PISCES	$P4_{14}$	n.a.	X
PVPP	n.a.	n.a.	X
ZodiacFX	$P4_{16}$	zodiacfx_model.p4	n.a.
FPGA			
P4→NetFPGA	$P4_{16}$	SimpleSumeSwitch	✓
Netcope P4	n.a.	n.a.	✓
P4FPGA	$P4_{14}$, $P4_{16}$	n.a.	X
ASIC			
Barefoot Tofino/Tofino 2	$P4_{14}$, $P4_{16}$	v1model, psa, TNA	✓
Pensando Capri	$P4_{16}$	n.a	✓
NPU			
Netronome	$P4_{14}$, $P4_{16}$	v1model	✓

- *psa_switch* is similar to simple_switch, but supports PSA instead of v1model.
- *simple_router* and *l2_switch* support only parts of the standard metadata and do not support $P4_{16}$. They are intended to show how different architectures can be implemented with bmv2.

Although bmv2 is intended for testing purposes only, throughput rates up to 1 Gbit/s for a P4 program with IPv4 LPM routing have been reported [51]. bmv2 is under active development, i.e., new functionality is added frequently.

5.1.3. BPF-based Targets

Berkeley Packet Filters (BPFs) add an interface on a UNIX system that allows sending and receiving raw packets via the data link layer. User space programs may rely on BPFs to filter packets that are sent to it. BPF-based P4 targets are mostly intended for programming packet filters or basic forwarding in P4.

eBPF. Extended Berkeley Packet Filters (eBPFs) are an extension of BPFs for the Linux kernel. eBPF programs are dynamically loaded into the Linux kernel and executed in a virtual machine (VM). They can be linked to functions in the kernel, inserted into the network data path via iproute2, or bound to sockets or network interfaces. eBPF programs are always verified by the kernel before execution, e.g., programs with loops or backward pointers would not be executed. Due to their execution in a VM, eBPF programs can only access certain regions in memory besides the local stack. Accessing kernel resources is protected by a white list. eBPF programs may not block and sleep, and usage of locks is limited to prevent deadlocks. The p4c compiler features the *p4c-ebpf* back-end to compile $P4_{16}$ programs to eBPF [52].

uBPF. user-space BPFs (uBPFs) relocate the eBPF VM from the kernel space to the user space. *p4c-ubpf* [53] is a backend for p4c that compiles P4 HLIR for uBPF. In contrast to p4c-ebpf, it also supports packet modification, checksum calculation, and registers, but no counters.

XDP. eXpress Data Path (XDP) is based on eBPF and allows to load an eBPF program into the RX queue of a device driver. p4c-xdp [54] is a backend for p4c that compiles P4 HLIR for XDP. Similar to p4c-ubpf, it supports packet modification and checksum calculation. In contrast to p4c-ebpf, it supports counters instead of registers.

5.1.4. T_4P_4S

T_4P_4S (pronounced "tapas") [55, 56] is a software P4 target that relies on interfaces for accelerated packet processing such as Data Plane Development Kit (DPDK) [57] or Open Data Plane (ODP) [44]. T_4P_4S provides a compiler that translates P4 programs into target-independent C code that interfaces a network hardware abstraction library. Hardware-dependent and hardware-independent functionalities are separated from each other. Its source code is available on GitHub [58]. Bhardwaj et al. [59] describe optimizations for improving T_4P_4S performance by up to 15%.

5.1.5. Ripple

Ripple [60] is a P4 target based on DPDK. It uses a static universal binary that is independent of the P4 program. The data plane of the static binary is configured at runtime based on P4 HLIR. This results in a shorter downtime when updating a P4 program in contrast to targets like T_4P_4S. Ripple uses vectorization to increase the performance of packet processing.

5.1.6. PISCES

PISCES [61] transforms the Open vSwitch (OVS) [62] into a software P4 target. OVS is a popular SDN software switch that is designed for high throughput on virtualization platforms for flexible networking between VMs. The PISCES compiler translates P4 programs into C code that replace parts of the source code of OVS. This makes OVS dependent on the P4 program, i.e., OVS must be recompiled with every modification of the P4 program. PISCES does not support stateful components such as registers, counters, or meters. The developers claim that PISCES does not add performance overhead to OVS. As the last commit in the public repository [63] is from 2016, PISCES seems not to be under active development.

5.1.7. PVPP

PVPP [64, 65] integrates P4 programs into plugins for Vector Packet Processors (VPP) (see Section 2.4.1). The P4-to-PVPP compiler comprises two stages. First, a modified p4c compiler translates P4 programs into target-dependent JSON code. Then, a Python compiler translates the JSON code into a VPP plugin in C source code. According to the authors, performance decreases by 5-17% compared to VPP but is still significantly better than OVS. Unfortunately, the source code and further information are not available for the public.

5.1.8. ZodiacFX

The ZodiacFX is a lightweight development and experimentation board originally designed as OF switch featuring four Fast Ethernet ports. It is based on an Atmel processor and an Ethernet switching chip [66]. The authors provided an extension [67, 68] to run P4 programs on the board. P4 programs are compiled using an extended version of p4c and the p4c-zodiacfx backend compiler. Then, the result of this compilation is used to generate a firmware image. Zanna et al. [69] compare the performance of P4 and OF on that target, and find out that differences among all test cases are small.

5.2. FPGA-Based P4 Targets

Several tool chains translate P4 programs into implementations for field programmable gate arrays (FPGAs). The process includes logic synthesis, verification, validation, and placement/routing of the logic circuit for the FPGA. We describe the P4→NetFPGA, Netcope P4, and P4FPGA tool chain. Finally, we mention research results for FPGA-based P4 targets.

5.2.1. P4→NetFPGA

The P4→NetFPGA workflow [70, 71] provides a development environment for compiling and running P4 programs on the NetFPGA SUME board that provides four SFP+ ports [72]. The development environment is built around the P4-SDnet compiler and the SDnet data plane builder from Xilinx, i.e., a full license for the Xilinx Vivado design suite is needed. Custom external functions can be implemented in a hardware description language (HDL) such as Verilog and included in the final FPGA program. This also allows external IP cores to be integrated as P4 externs in P4 programs. The P4→NetFPGA tool chain supports $P4_{16}$ based on the P4 architecture SimpleSumeSwitch (see Section 4.1).

5.2.2. Netcope P4

Netcope P4 [73] is a commercial cloud service that creates FPGA firmware from P4 programs. Knowledge of HDL development is not needed and all necessary IP cores are provided by Netcope. The cloud service can be used in conjunction with the Netcope software development kit (SDK). This combination allows developers to combine the VHDL code of the cloud service with custom HDL code, e.g., from an external function. As target platform, Netcope P4 supports FPGA boards from Netcope, Silicom, and Intel that are based on Xilinx or Intel FPGAs.

5.2.3. P4FPGA

P4FPGA [74] is a $P4_{14}$ and $P4_{16}$ compiler and runtime for the Bluespec programming language that can generate code for Xilinx and Altera FPGAs. The last commit in the archived public repository [75] is from 2017.

5.2.4. Research Results

Benácek and Kubátová [76, 77] present how P4 parse graph descriptions can be converted to optimized VHDL code for FPGAs. The authors demonstrate how a complex parser for several header fields achieves a throughput of 100 Gbit/s on a Xilinx Virtex-7 FPGA while using 2.78% slice look up tables (LUTs) and 0.76% slice registers (REGs). In a follow-up work [78], the optimized parser architecture supports a throughput of 1 Tbit/s on Xilinx UltraScale+ FPGAs and 800 Gbit/s on Xilinx Virtex-7 FPGAs. Da Silva et al. [79] also investigate the high-level synthesis of packet parsers in FPGAs. Kekely and Korenek [80] describe how MATs can be mapped to FPGAs. Iša et al. [81] describe a system for automated verification of register-transfer level (RTL) generated from P4 source code. Cao et al. [82, 83] propose a template-based process to convert P4 programs to VHDL. They use a standard P4 frontend compiler to compile the P4 program into an intermediate representation. From this representation, a custom compiler maps the different elements of the P4 program to VHDL templates which are used to generate the FPGA code.

5.3. ASIC-Based P4 Targets

5.3.1. Intel Tofino

Intel Tofino is the world's first user programmable Ethernet switch ASIC. It is designed for very high throughput of 6.5 Tbit/s (4.88 B pps) with 65 ports running at 100 Gbit/s. Its successor, the Tofino 2 ASIC, supports throughput rates of up to 12.8 Tbit/s with ports running at up to 400 Gbit/s. Tofino has been built by Barefoot Networks, a former startup company that was acquired by Intel in 2019.

The Tofino ASIC implements the TNA, a custom P4 architecture that significantly extends PSA (see Section 4.1). It provides support for advanced device capabilities which are required to implement complex, industrial-strength data plane programs. The device comes with 2 or 4 independent packet processing pipelines (pipes), each capable of serving 16 100 Gbit/s ports. All pipes can run the same P4 program or each pipe can run its own program independently. Pipes can also be connected together, allowing the programmers to build programs requiring longer processing pipelines.

The Tofino ASIC processes packets at line rate irrespective of the complexity of the executed P4 program. This is achieved by a high degree of pipelining (each pipe is capable of processing hundreds of packets simultaneously) and parallelization. In addition to standard arithmetic and logical operations, Tofino provides specialized capabilities, often required by data plane programs, such as hash computation units and random number generators. For stateful processing Tofino offers counters, meters, and registers, as well as more specialized processing units. Some of them support specialized operations, such as approximate non-linear computations required to implement state-of-the-art data plane algorithms. Built-in packet generators allow the data plane designers to implement protocols, such as BFD, without using externally running control plane processes. These and other components are exposed through TNA which is openly published by Intel [84].

Tofino fixed-function components offer plenty of advanced functionality. The buffering engine has a unified 22 MB buffer, shared by all the pipes, that can be subdivided into several pools. Tofino Traffic Manager supports both store-and-forward as well as the cut-through mode, up to 32 queues per port, precise traffic shaping and multiple scheduling disciplines. Tofino provides nanosecond-precision timestamping that facilitates both the implementation of time synchronization protocols, such as IEEE 1588, as well as precise delay measurements. Additional intrinsic metadata support a variety of telemetry applications, such as INT.

The development is conducted using Intel P4 Studio which is a software development environment containing the P4 compiler, the driver, and other software necessary to program and manage the Tofino. A special interactive visualization tool (P4i) allows the developers to see the P4 program being mapped onto the specific hardware resources further assisting them in fitting and optimizing their programs. Intel P4 compiler for Tofino has special capabilities, allowing it to parallelize the code thereby taking advantage of the highly parallel nature of Tofino hardware.

A number of original design manufacturers (ODMs) produce open systems (white boxes) with the Tofino ASIC that are used for research, development, and production of custom systems. Examples include the EdgeCore Wedge 100BF-32X [85], APS Networks BF2556-1T-A1F [86] and BF6064-T-A2F [87], NetBerg Aurora 610 [88], and others.

Most white box systems follow a modern, server-like design with a separate board management controller, responsible for handling power supplies, fans, LEDs, etc., and a main CPU, typically x86_64, running a Linux operating system. The main CPU is connected to the Tofino ASIC via a PCIe interface. Some boards also provide one or more high-speed on-board Ethernet connections for faster packet interface. External Ethernet ports support speeds from 10 Gbit/s to 100 Gbit/s using standard QSFP28 cages although some systems offer lower-speed (1 Gbit/s) ports as well. Most of these systems are also powerful enough to support running development tools natively, e.g., a P4 compiler, even though this is not necessarily required.

Tofino ASICs are also used in proprietary network switches, e.g., by Arista [89] and Cisco [90]. Some Tofino-based switches are supported by Microsoft SONiC [91].

5.3.2. Pensando Capri

The Capri P4 Programmable Processor [92, 93] is an ASIC that powers network interface cards (NICs) by Pensando Systems aimed for cloud providers. It is coupled with fixed function components for cryptography operations like AES or compression algorithms and features multiple ARM cores.

5.4. NPU-Based P4 Targets

Network processing units (NPUs) are software-programmable ASICs that are optimized for networking applications. They are part of standalone network devices or device boards, e.g., PCI cards.

Netronome network flow processing (NFP) silicons can be programmed with P4 [94] or C [95]. A C-based programming model is available that supports program functions to access payloads and allows developing P4 externs. The Agilio P4C SDK consists of a tool chain including a backend compiler, host software, and a full-featured integrated development environment (IDE). All current Agilio SmartNICs based on NFP-4000, NFP-5000, and NFP-6480 are supported. Harkous et al. [96] investigate the impact of basic P4 constructs on packet latency on Agilio SmartNICs.

6. P4 Data Plane APIs

We introduce data plane APIs for P4, present a characterization, describe the three most commonly used P4 data plane APIs, and compare different control plane use cases.

6.1. Definition & Functionality

Control planes manage the runtime behavior of P4 targets via data plane APIs. Alternative terms are *control plane APIs* and *runtime APIs*. The data plane API is provided by a device driver or an equivalent software component. It exposes data plane features to the control plane in a well-defined way. Figure 20 shows the main control plane operations. Most important, data plane APIs facilitate runtime control of P4 entities (MATs and externs). They typically also comprise a packet I/O mechanism to stream packets to/from the control plane. They also include reconfiguration mechanisms to load P4 programs onto the P4 target. Control planes can control data planes only through data plane APIs, i.e., if a data plane feature is not exposed via a corresponding API, it cannot be used by the control plane.

It is important to note that P4 does not require a data plane APIs. P4 targets may also be used as a packet processor with a fixed behavior that is defined by the P4 program where static MAT entries are part of the P4 program itself.

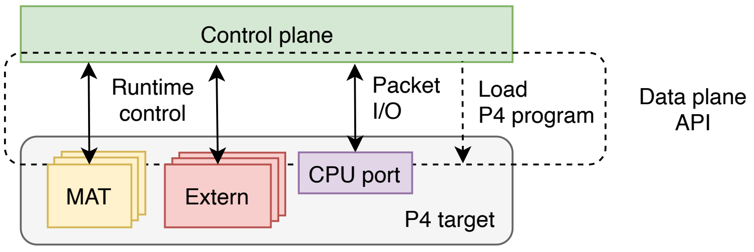


Figure 20: Runtime management of a P4 target by the control plane through the data plane API. The figure depicts the four most central operations: Runtime control of MATs and extern objects, packet-in/out, and loading of P4 programs.

6.2. Characterization of Data Plane APIs

Data plane APIs in P4 can be characterized by their level of abstraction, their dependency on the P4 program, and the location of the control plane.

6.2.1. Level of Abstraction

Data plane APIs can be characterized by their level of abstraction.

- *Device access APIs* provide direct access to hardware functionalities like device registers or memories. They typically use low-level mechanisms like DMA transactions. While this results in very low overhead, this type of API can be neither vendor- nor device-independent.

- *Data plane specific APIs* are APIs with a higher level of abstraction. They provide access to objects defined by the P4 program instead of hardware-specific parts. In contrast to device access APIs, vendor- and device-independence is possible for this type of API.

6.2.2. Dependency on the P4 Program

Data plane APIs can be characterized by their dependency on the P4 program.

- *Program-dependent APIs* have a set of functions, data structures, and other names that are derived from the P4 program itself. Therefore, they depend on the P4 program and are applicable to this P4 program only. If the corresponding P4 program is changed, function names, data structures, etc., might change, which requires a recompilation or modification of the control plane program.

- *Program-independent APIs* consist of a fixed set of functions that receives a list of P4 objects that are defined in the P4 program. Thus, the names of the API functions, data structures, etc., do not depend on the program and are universally applicable. If the corresponding P4 program changes, neither the names, nor the definitions of the API functions will change as long as the control plane “knows” the names of the right tables, fields and other object that need to be operated on. Program-independent APIs model configurable objects either with the *object-based* or the *table-based* approach. As known from object-oriented programming, the object-based approach relies on methods that are defined for each class of data plane objects. In contrast, the table-based approach treats every class of data plane object as a variation of a table. This reduces the number of API methods as only table manipulations need to be provided as methods.

6.2.3. Control Plane Location

Data plane APIs can be characterized by the location of the control plane.

- *APIs for local control* are implemented by the device driver and are executed on the local CPU of the device that hosts the programmable data plane. Usually, the APIs are presented as set of C function calls just like for other devices that operating system are accessing.
- *APIs for remote control* add the ability to invoke API calls from a separate system. This increases system stability and modularity, and is essential for SDN and other systems with centralized control. Remote control APIs follow the base methodology of remote procedure calls (RPCs) but rely on modern message-based frameworks that allow asynchronous communication and concurrent calls to the API. Examples are Thrift [97] or gRPC [98]. For example, gRPC uses HTTP/2 for transport and includes many functionalities ranging from access authentication, streaming, and flow control. The protocol’s data structures, services, and serialization schemes are described with protocol buffers (protobuf) [99].

6.3. Data Plane API Implementations

We introduce the three most common data plane APIs: P4Runtime, Barefoot Runtime Interface (BRI), and BM Runtime. All of them are data-plane specific and program-independent. Table 4 lists their properties that have been introduced before.

6.3.1. P4Runtime API

P4Runtime is one of the most commonly used data plane APIs that is standardized in the API WG [100] of the P4 Language Consortium. For implementing the RPC mechanisms, it relies on the gRPC framework with protobuf. Its most recent specification v1.3.0 [101] was published in December 2020.

Operating Principle. Figure 21 depicts the operating principle of P4Runtime. P4 targets include a gRPC server, controllers implement a gRPC client. To protect the gRPC connection, TLS with optional mutual certificate authentication can be enabled. The API structure of P4Runtime is described within the `p4runtime.proto` definition. The gRPC server on P4 targets interacts with the P4-programmable components via platform drivers. It has access to

P4 entities (MATs or externs) and can load target-specific configuration binaries. The structure of the API calls to access P4 entities are described in the `p4info.proto`. It is part of the P4Runtime but developers can extend it to use custom data structures, e.g., to implement interaction with target-specific externs. P4Runtime provides support for multiple controllers. For every P4 entity, read access is provided to all controllers whereas write access is only provided to one controller. To manage this access, P4 entities can be arranged in groups where each group is assigned to one primary controller with write access and arbitrary, secondary controllers with read access. Interaction between controllers and P4 targets works as follows. P4 compilers (see Section 4.2) with support for P4Runtime generate a P4Runtime configuration. It consists of the target-specific configuration binaries and P4Info metadata. P4Info describes all P4 entities (MATs and externs) that can be accessed by controllers via P4Runtime. Then, the controllers establish a gRPC connection to the gRPC server on the P4 target. The target-specific configuration is loaded onto the P4 target and P4 entities can be accessed.

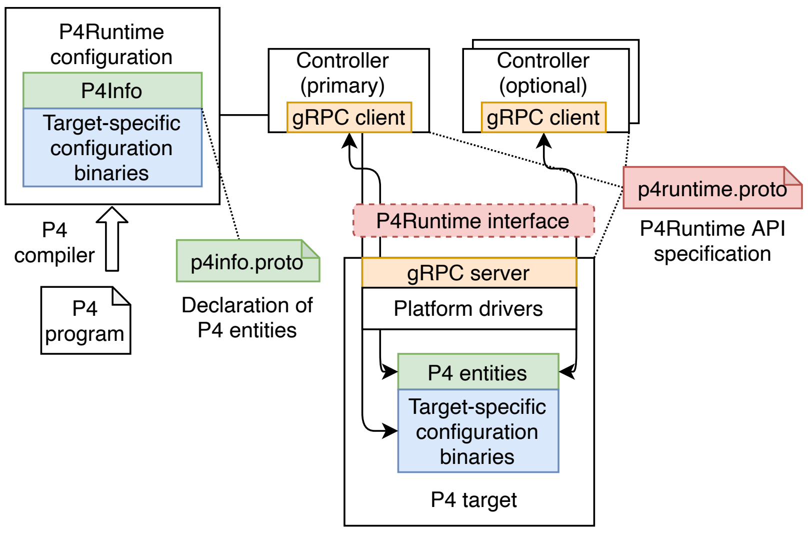


Figure 21: P4Runtime architecture (similar to [101]). P4 targets can be managed by a primary controller and multiple, optional controllers. The P4 entities and P4Runtime API specification is part of protocol definitions.

Implementations. gRPC and protobuf libraries are available for many high-level programming languages such as C++, Java, Go, or Python. Thereby, P4Runtime can be implemented easily on both controllers and P4 targets.

- *Controllers*: P4Runtime is supported by most common SDN controllers. P4 brigade [102] introduces support for P4Runtime on the Open Network

Operating System (ONOS). OpenDaylight (ODL) introduces support for P4Runtime via a plugin [103]. Stratum [104] is an open-source network operating system that includes an implementation of the P4Runtime and OpenConfig interfaces. Custom controllers, e.g., for P4 prototypes, can be implemented in Python with the help of the p4runtime_lib [105].

- *Targets*: The *PI Library* [106] is the open-source reference implementation of a P4Runtime gRPC server in C. It implements functionality for accessing MATs and supports extensions for target-specific configuration objects, e.g., registers of a hardware P4 target. The PI Library is used by many P4 targets including bmv2 [107] and the Tofino.

6.3.2. Barefoot Runtime Interface (BRI)

The BRI consists of two independent APIs that are available on Tofino-based P4 hardware targets. The *BfRt API* is an API for local control. It includes C, C++ and Python bindings that can be used to implement control plane programs. The *BF Runtime* is an API for remote control. As for P4Runtime, it is based on the gRPC RPC framework and protobuf, i.e., bindings for different languages are available. An additional Python library implements a simpler, BfRt-like interface for cases where simplicity is more essential than the performance of BF Runtime.

6.3.3. BM Runtime API

BM Runtime API is a program-independent data plane API for the bmv2 software target. It relies on the Thrift RPC framework. bmv2 includes a command line interface (CLI) program [108] to manipulate MATs and configure the multicast engine of the bmv2 P4 software target via this API.

Table 4: Characterization of data plane specific APIs.

API	**Program independence**	**Control plane location**
P4Runtime	✓	Remote (gRPC)
BF Runtime	✓	Remote (gRPC)
BfRt API	✓	Local (C, C++ and Python bindings)
BM Runtime	✓	Remote (Thrift RPC)

6.4. Controller Use Case Patterns

We present three use case patterns which are abstractions of the controller use cases introduced in the P4Runtime specification [101]. However, these are neither conclusive nor complete as derivations or extensions are possible.

6.4.1. Embedded/Local Controller

P4 hardware targets (see Section 5) comprise or are attached to a computing platform. This facilitates running controllers directly on the P4 target. Figure 22 depicts this setup. The controller application may either use a local API, e.g., C calls, or just execute a controller application that interfaces the data plane via an RPC channel.

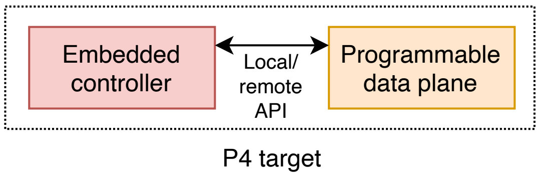


Figure 22: Embedded/local controller use case pattern. The P4 target comprises an embedded controller that is running a control plane program.

6.4.2. Remote Controllers

Remote controllers resemble the typical SDN setup where data plane devices are managed by a centralized control plane with an overall view on the network. Controllers need to be protected against outages and capacity overload, i.e., they need to be replicated for fail-safety and scalability. Figure 23 depicts two possible use cases. In the first shown use case (a), the programmable data plane on the P4 target is managed by remote controllers. In the second shown use case (b), the P4 target is managed by both, the embedded controller and remote controllers. Remote controllers might be interfaced using the remote API of the programmable data plane or an arbitrary API that is provided by the embedded controller. This option is often used for the implementation of so-called *hierarchical control plane* structures where control plane functionality is distributed among different layers. Control plane functions that do not require a global view of the network, e.g., link discovery, MAC learning for L2 forwarding, or port status monitoring, can be solely performed by the embedded/local controller. Other control plane functions that require an overall view of the network, e.g., routing applications, can be performed by the remote controller, possibly in cooperation with the embedded/local controller where the local controller acts as proxy, i.e., it relays control plane messages between the P4 target and the global controller. Hierarchical control planes improve load distribution as many tasks can be performed locally, which reduces load on the remote controllers. In particular, time-critical operations may benefit from local controllers as additional delays caused by the communication between a P4 target and a global controller are avoided.

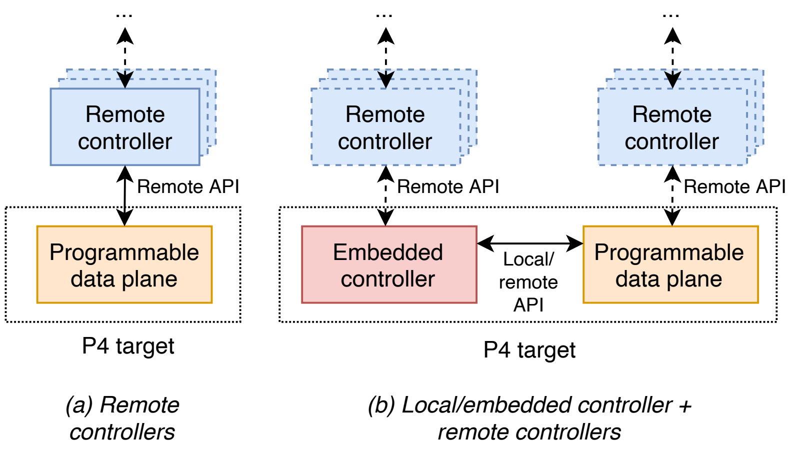


Figure 23: Two use case patterns for remote controllers: The P4 target may be solely managed by remote controllers or it may be managed by an embedded controller and remote controllers.

7. Advances in P4 Data Plane Programming

We give an overview on research to improve P4 data plane programming. Figure 24 depicts the structure of this section. We describe related work on optimization of development and deployment, testing and debugging, research on P4 targets, and research on control plane operation.

7.1. Optimization of Development and Deployment

We describe research work on optimizing the development & deployment process of P4.

7.1.1. Program Development

Graph-to-P4 [109] generates P4 program code for given parse graphs. This introduces a higher abstraction layer that is particularly helpful for beginners. Zhou et al. [110] introduce a module system for P4 to improve source code organization. DaPIPE [111] enables incremental deployment of P4 program code on P4 targets. SafeP4 [112] adds type safety to P4. P4I/O [113] presents a framework for intent-based networking with P4. Network operator describe their network functions with an Intent Definition Language (IDL) and P4I/O generates a complete P4 program accordingly. To that end, P4I/O provides a P4 action repository with various network functions. During reconfiguration, table and register state are preserved by applying backup mechanisms. P4I/O is implemented for a custom bmv2. Mantis [114] is a framework to implement fast reactions to changing network conditions in the data plane without controller interaction. To that end, annotations in the P4 code specify dynamic components and a quick control loop of those components ensure timely adjustments

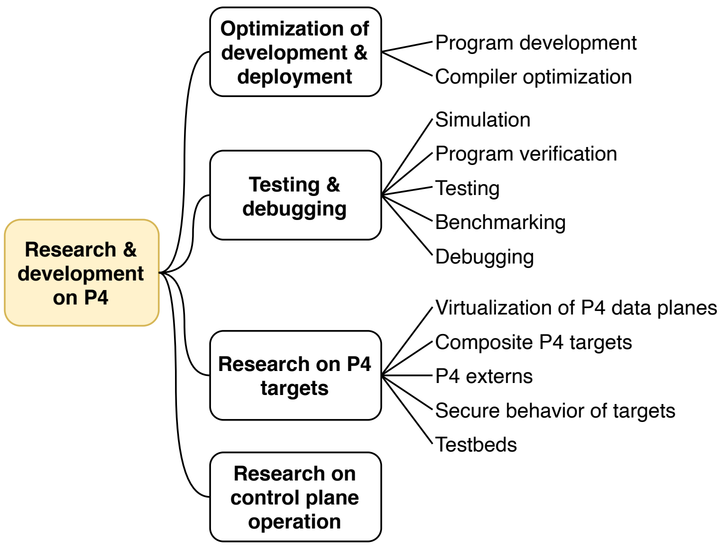


Figure 24: Organization of Section 7.

if necessary. Lyra [115] is a pipeline abstraction that allows developers to use simple statements to describe their desired data plane without low-level target-specific knowledge. Lyra then compiles that description to target-specific code for execution. GP4P4 [116] is a programming framework for self-driven networks. It generates P4 code from behavioral rules defined by the developer. To that end, GP4P4 evaluates the quality of the automatically generated programs and improves them based on genetic algorithms. FlowBlaze.p4 [117, 118, 119] implements an executor for FlowBlaze, an abstraction based on an extended finite state machine for building stateful packet processing functions, in P4. This library maps FlowBlaze elements to P4 components for execution on the bmv2. It also provides a GUI for defining the extended finite state machine. Flightplan [120] is a programming tool chain that disaggregates a P4 program into multiple P4 programs so that they can be executed on different targets. The authors state that this improves performance, resource utilization, and cost.

7.1.2. Compiler Optimization

pcube [121] is a preprocessor for P4 that translates primitive annotations in P4 programs into P4 code for common operations such as loops. CacheP4 [122] introduces a behavior-level cache in front of the P4 pipeline. It identifies flows and performs a compound of actions to avoid unnecessary table matches. The cache is filled during runtime by a controller that receives notifications from the switch. P5 [123] optimizes the P4 pipeline by removing inter-feature dependencies. dRMT [25] is a new architecture for programmable switches that introduces deterministic throughput and latency guarantees. Therefore, it generates schedules for CPU and memory resources from a P4 program. P2GO [124] leverages monitored traffic information to optimize resource allocation during compilation. It adjusts table and register size to reduce the pipeline length, and offloads rarely used parts of the program to the control plane. Yang et al. [125] propose a compiler module that optimizes lookup speed by reorganizing flow tables and prioritization of popular forwarding rules. Vass et al. [126] analyze and discuss algorithmic aspects of P4 compilation.

7.2. Testing and Debugging

We describe research work on simulation, program verification, testing, benchmarking, and debugging.

7.2.1. Simulation

PFPSim [127] is a simulator for validation of packet processing in P4. NS4 [128, 129] is a network simulator for P4 programs that is based on the network simulator NS3.

7.2.2. Program Verification

McKeown et al. [130] introduce a tool to translate P4 to the Datalog declarative programming language. Then, the Datalog representation of the P4 program can be analyzed for well-formedness. Kheradmand et al. [131] introduce a tool for static analysis of P4 programs that is based on formal semantics. P4v [132] adapts common verification methods for P4 that are based on annotations in the P4 program code. Freire et al. [133, 134] introduce assertion-based verification with symbolic execution. Stoenescu et al. [135] propose program verification based on symbolic execution in combination with a novel description language designed for the properties of P4. P4AIG [136] proposes to use hardware verification techniques where developers have to annotate their code with First Order Logic (FOL) specifications. P4AIG then encodes the P4 program as an Advanced-Inverter-Graph (AIG) which can be verified by hardware verification techniques such as circuit SAT solvers and bounded model checkers. bf4 [137] leverages static code verification and runtime checks of rules that are installed by the controller to confirm that the P4 program is running as intended. netdiff [138] uses symbolic execution to check if two data planes are equivalent. This can be useful to verify if a data plane behaves correctly by comparing it with a similar one, or to verify that optimizations of a data plane do not change its behavior. Yousefi et al. [139] present an abstraction for liveness verification of stateful network functions (NFs). The abstraction is based on boolean formulae. Further, they provide a compiler that translates these formulae into P4 programs.

7.2.3. Testing

P4pktgen [140] generates test cases for P4 programs by creating test packets and table entries. P4Tester [141] implements a detection scheme for runtime faults in P4 programs based on probe packets. P4app [142] is a partially automated open source tool for building, running, debugging, and testing P4 programs with the help of Docker images. P4RL [143] is a reinforcement learning based system for testing P4 programs and P4 targets at runtime. The correct behavior is described in a simple query language so that a reinforcement agent based on Double DQN can learn how to manipulate and generate packets that contradict the expected behavior. P4TrafficTool [144] analyzes P4 programs to produce plugin code for common traffic analyzers and generators such as Wireshark.

7.2.4. Benchmarking

Whippersnapper [145] is a benchmark suite for P4 that differentiates between platform-independent and platform-specific tests. BB-Gen [146] is a system to evaluate P4 programs with existing benchmark tools by translating P4 code into other formats. P8 [147] estimates the average packet latency at compilation time by analyzing the data path program.

7.2.5. Debugging

Kodeswaran et al. [148] propose to use Ball-Larus encoding to track the packet execution path through a P4 program for more precise debugging capabilities. p4-data-flow [149] detects bugs by creating a control flow graph of a P4 program and then identifies incorrect behavior. P4box [150] extends the $P4_{16}$ reference compiler by so-called *monitors* that insert code before and after programmable blocks, e.g., control blocks, for runtime verification. P4DB [151] [152] introduces a runtime debugging system for P4 that leverages additional debugging snippets in the P4 program to generate reports during runtime. Neves et al. [153] propose a sandbox for P4 data plane programs for diagnosis and tracing. P4Consist [154] verifies the consistency between control and data plane. Therefore, it generates active probe-based traffic for which the control and data plane generate independent reports that can be compared later. KeySight [155] is a troubleshooting platform that analyzes network telemetry data for detecting runtime faults. Gauntlet [156] finds both crash bugs, i.e., abnormal termination of compilation operation, and semantic bugs, i.e., miscompilation, in compilers for programmable packet processors.

7.3. Research on P4 Targets

We describe research work on virtualization of P4 data planes, composite targets, P4 externs, secure behavior of targets, and testbeds.

7.3.1. Virtualization of P4 Data Planes

P4 targets are designed to execute one P4 program at any given time. Virtualization aims at sharing the resources of P4 targets for multiple P4 programs. Krude et al. [157] provide theoretical discussions on how ASIC- and FPGA-based P4 targets can be shared between different tenants and how P4 programs can be made hot-pluggable.

HyPer4 [158] introduces virtualization for P4 data planes. It supports scenarios such as network slicing, network snapshotting, and virtual networking. To that end, a compiler translates P4 programs into table entries that configure the HyPer4 *persona*, a P4 program that contains implementations of basic primitives. However, HyPer4 does not support stateful memory (registers, counters, meters), LPM, range match types, and arbitrary checksums. The authors describe an implementation for bmv2 and perform experiments that reveal 80 to 90% lower performance in comparison to native execution.

HyperV [159, 160, 161] is a hypervisor for P4 data planes with modular programmability. It allows isolation and dynamic management of network functions. The authors implemented a prototype for the bmv2 P4 target. In comparison to Hyper4, HyperV achieves a 2.5x performance advantage in terms of bandwidth and latency while reducing required resources by a factor of 4. HyperVDP [162] extends HyperV by an implementation of a dynamic controller that supports instantiating network functions in virtual data planes.

P4VBox [163], also published as VirtP4 [164], is a virtualization framework for the NetFPGA SUME P4 target. It allows executing virtual switch instances in parallel and also to hot-swap them. In contrast to HyPer4, HyperV and HyperVDP, P4VBox achieves virtualization by partially re-configuring the hardware.

P4Visor [165] merges multiple P4 programs. This is done by program overlap analysis and compiler optimization. Programming In-Network Modular Extensions (PRIME) [166] also allows combining several P4 programs to a single program and to steer packets through the specific control flows.

P4click [167] does not only merge multiple P4 programs, but also combines the corresponding control plane blocks. The purpose of P4click is to increase the use of data plane programmability. P4click is currently in an early stage of development.

The Multi Tenant Portable Switch Architecture (MTPSA) [168] is a P4 architecture that offers performance isolation, resource isolation, and security isolation in a switch for multiple tenants. MTPSA is based on the PSA. It combines a *Superuser* pipeline that acts as a hypervisor with multiple user pipelines. User pipelines may only perform specific actions depending on their privileges. MTPSA is implemented for bmv2 and NetFPGA-SUME [169].

Han et al. [170] provide an overview of virtualization in programmable data planes with a focus on P4. They classify virtualization schemes into hypervisor and compiler-based approaches, followed by a discussion of pros and cons of the different schemes. The aforementioned works on virtualization of P4 data planes are described and compared in detail.

7.3.2. Composite P4 Target

Da Silva et al. [171] introduce the idea of composite P4 targets. This tries to solve the problem of target-dependent support of features. The composed data plane appears as one P4 target; it is emulated by a P4 software target but relies on an FPGA and ASIC for packet processing.

eXtra Large Table (XLT) [172] introduces gigabyte-scale MATs by leveraging FPGA and DRAM capabilities. It comprises a P4-capable ASIC and multiple FPGAs with DDR4 DRAM. The P4-capable ASIC pre-constructs the match key field and sends it with the full packet to the FPGA. The FPGA sends back the original packet with the search results of the MAT lookup. The authors implement a DPDK based prototype for the T_4P_4S P4 software target.

HyMoS [173] is a hybrid software and hardware switch to support NFV applications. The authors create a switch by using P4-enabled Smart NICs as line cards and the PCIe interface of a computer as the switch fabric. P4 is used for packet switching between the NICs. Additional processing may be done using DPDK or applications running on a GPU.

7.3.3. P4 Externs

Laki et al. [174, 175] investigate asynchronous execution of externs. In contrast to common synchronous execution, other packets may be processed by the pipeline while the extern function is running. The authors implement and evaluate a prototype for T4P4S. Scholz et al. [176] propose that P4 targets should be extended by cryptographic hash functions that are required to build secure applications and protocols. The authors propose an extension of the PSA and discuss the PoC implementation for a CPU-, network processing unit (NPU)-, and FPGA-based P4 target. Da Silva et al. [177] investigate the implementation of complex operations as extensions to P4. The authors perform a case study on integrating the Robust Header Compression (ROHC) scheme and conclude that an implementation as extern function is superior to an implementation as a new native primitive.

7.3.4. Secure Behaviour of Targets

Gray et al. [178] demonstrate that hardware details of P4 targets influence their packet processing behavior. The authors demonstrate this by sending a special traffic pattern to a P4 firewall. It fills the cache of this target and results in a blocking behavior although the overall data rate is far below the capacity of the used P4 target. Dumitru et al. [179] investigate the exploitation of programming bugs in bmv2, P4-NetFPGA, and Tofino. The authors demonstrate attack scenarios by header field access on invalid headers, the creation of infinite loops and unintentionally processing of dropped packets in the P4 targets.

7.3.5. Testbeds

Large testbeds facilitate research and development on P4 programs. The i-4PEN (International P4 Experimental Networks) [180] is an international P4 testbed operated by a collaboration of network research institutions from the USA, Canada, and Taiwan. Chung et al.[181] describe how multi-tenancy is achieved in this testbed. The 2STiC testbed [182], a national testbed in the Netherlands comprising six sites with at least one Tofino-based P4 target, is connected to i-4PEN.

7.4. Research on Control Plane Operation

When new forwarding entries are computed by the controller, the data plane has to be updated. However, updating the targets has to be performed in a manner that prevents negative side effects. For example, microloops may occur if packets are forwarded according to new rules at some targets while at other devices old rules are used because updates have to arrive yet.

Sukapuram et al. [183, 184] introduce a timestamp in the packet header that contains the sending time of a packet. When switches receive a packet during an update period, they compare the timestamp of both the packet and the update to determine whether a packet has been sent before the update, and thus, old rules should be used for forwarding.

Liu et al. [185] introduce a mechanism where once a packet is matched against a specific forwarding rule, it cannot be matched downstream on a rule that is older. To that end, the packet header contains a timestamp field that records when the last applied forwarding rule has been updated. If the packet is matched against an older rule, the packet is dropped, otherwise the timestamp is updated and the packet is forwarded.

Ez-Segway [186] facilitates updating by including data plane devices in the update process. When a data plane device receives an update, it determines which of its neighbors is affected by the update as well, and forwards the update to that neighbor. This prevents loops and black holes.

TableVisor [187] is a transparent proxy-layer between the control plane and data plane. It provides an abstraction from heterogeneous data plane devices. This facilitates the configuration of data plane switches with different properties, e.g., forwarding table size.

Molero et al. [188] propose to offload tasks from the control plane to the data plane. They show that programmable data planes are able to run typical control plane operations like failure detection and notification, and connectivity retrieval. They discuss trade-offs, limitations and future research opportunities.

8. Applied Research Domains: Classification & Overview

In the following sections, we give an overview of applied research conducted with P4. In this section, we classify P4's core features that make it attractive for the implementation of data plane algorithms. We define research domains, visualize them in a compact way, and explain our method to review corresponding research papers in the subsequent sections. Finally, we delimit the scope of the surveyed literature.

8.1. Classification of P4's Core Features

We identify P4's core features for the implementation of prototypes. We classify them in the following to effectively reason about P4's usefulness for the surveyed research works.

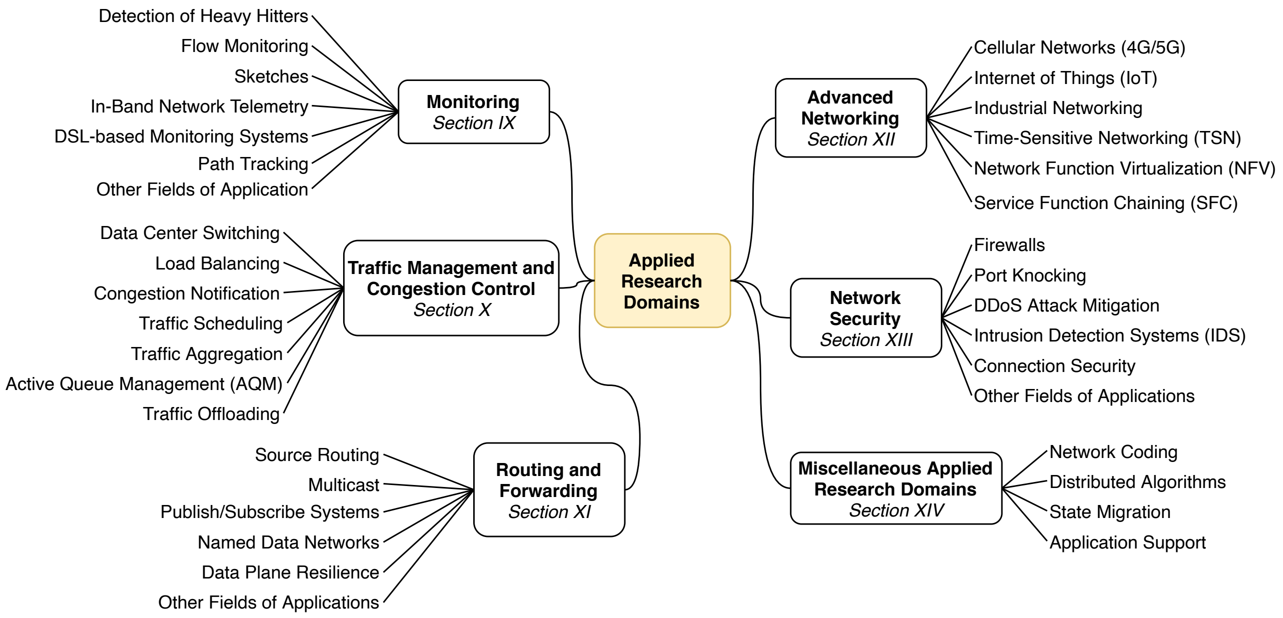

Applied Research Domains
Monitoring Section IX
Detection of Heavy Hitters
Flow Monitoring
Sketches
In-Band Network Telemetry
DSL-based Monitoring Systems
Path Tracking
Other Fields of Application
Traffic Management and Congestion Control Section X
Data Center Switching
Load Balancing
Congestion Notification
Traffic Scheduling
Traffic Aggregation
Active Queue Management (AQM)
Traffic Offloading
Routing and Forwarding Section XI
Source Routing
Multicast
Publish/Subscribe Systems
Named Data Networks
Data Plane Resilience
Other Fields of Applications
Advanced Networking Section XII
Cellular Networks (4G/5G)
Internet of Things (IoT)
Industrial Networking
Time-Sensitive Networking (TSN)
Network Function Virtualization (NFV)
Service Function Chaining (SFC)
Network Security Section XIII
Firewalls
Port Knocking
DDoS Attack Mitigation
Intrusion Detection Systems (IDS)
Connection Security
Other Fields of Applications
Miscellaneous Applied Research Domains Section XIV
Network Coding
Distributed Algorithms
State Migration
Application Support

Figure 25: Categorization of the surveyed works into applied research domains and subdomains – they correspond to sections and subsections in the remainder of this paper.

8.1.1. Definition and Usage of Custom Packet Headers

P4 requires the definition of packet headers (Section 3.5). These may be headers of standard protocols, e.g., TCP, use-case-specific protocols, e.g., GTP in 5G, or new protocols. As P4 supports the definition of custom headers, it is suitable for the implementation of data plane algorithms using new protocols or extensions of existing protocols, e.g., for in-band signalling.

8.1.2. Flexible Packet Header Processing

Control blocks with MATs (Section 3.6) comprise the packet processing logic. Packet processing includes default actions, e.g., forwarding and header field modifications, or custom, user-defined actions. Both may be parameterized via MATs or metadata. Entries in the MATs are maintained by a data plane API (Section 6). The flexible use of actions, the definition of new actions, and their parameterization offer high flexibility for header processing, which is often needed for research prototypes.

8.1.3. Target-Specific Packet Header Processing Functions

While the above-mentioned features are part of the P4 core language and supported by any P4-capable platform, devices may offer additional architecture- or target-specific functionality which is made available as *P4 extern* (Section 4). Typical externs include components for stateful processing, e.g., registers or counters, operations to resubmit/recirculate the packet in the data plane, multicast operations, or more complex operations, e.g., hashing and encryption/decryption. P4 software targets allow users to integrate custom externs and use them within P4 programs. While this is also possible to some extent on some P4 hardware targets, e.g., the NetFPGA SUME board, high-throughput P4 targets based on the Tofino ASIC have only a fixed set of externs (Section 5.3). Depending on the use case, the availability of externs may be essential for the implementation of prototypes. Thus, externs facilitate the implementation of more complex algorithms but make implementations platform-dependent.

8.1.4. Packet Processing on the Control Plane

Similar to control plane SDN (e.g., OF), more complex, and optionally centralized packet processing can be outsourced to an SDN control plane; packet exchange and data plane control is performed via a data plane API (Section 6). While OF only allows the exchange of complete packets, P4 enables the end-users to define the packet formats.

8.1.5. Flexible Development and Deployment

Users are able to easily change the P4 programs on P4 targets that are installed in a network. This facilitates agile development with frequent deployments and incremental functionality extensions by deploying new versions of a P4 programs.

8.2. Categorization of Research Domains

To organize the survey in the following sections, we define research domains and structure them in a two-level hierarchy as depicted in Figure 25. This categorization helps the reader to get a quick overview in certain applied areas and improves the readability of this survey. The choice of the research domains is dominated by the fields of applications, but the summaries of the sections will show that the prototypes in these areas benefit from different core features of P4.

For each research domain, we provide a table that lists the publications with publication year, P4 target platforms, and source code availability. This supports efficient browsing of the content and backs our conclusions in the section-specific summaries.

8.3. Scope of the Surveyed Literature

We consider the literature until the end of 2020 and selected papers from 2021, including journal papers, conference papers, workshop papers, and preprints. Out of the 377 scientific publications we surveyed in this work (see Section 1), 245 fall in the area of applied research. 68 of those research papers were published in 2018 or before, 80 were published in 2019, 93 were published in 2020, and 4 were published in 2021. 60 out of all 245 research publications released the source code of their prototype implementations.

Table 5 depicts a statistic on major publication venues for the papers of applied research domains. It helps the reader to identify potential venues for prospective own publications based on P4 technology.

9. Applied Research Domains: Monitoring

We describe applied research on detection of heavy hitters, flow monitoring, sketches, in-band network telemetry, and other areas of application. Table 6 shows an overview of all the work described. At the end of the section, we summarize the work and analyze it with regard to P4's core features described in Section 8.1.

9.1. Detection of Heavy Hitters

Heavy hitters [269] (or "elephant flows") are large traffic flows that are the major source of network congestion. Detection mechanisms aim at identifying heavy hitters to perform extra processing, e.g., queuing, flow rate control, and traffic engineering.

HashPipe [189] integrates a heavy hitter detection algorithm entirely on the P4 data plane. A pipeline of hash tables acts as a counter for detected flows. To fulfill memory constraints, the number of flows that can be stored is limited. When a new flow is detected, it replaces the flow with the lowest count. Thus, light flows are replaced, and heavy flows can be detected by a high count. Lin et al. [191] describe an enhanced version of the algorithm.

Popescu et al. [192] introduce a heavy hitter detection mechanism. The controller installs TCAM entries for specific source IP prefixes on the switch. If one of these entries matches more often than a threshold during a given time frame, the entry is split into two entries with a larger prefix size. This procedure is repeated until the configured granularity is reached.

Harrison et al. [193] presents a controller-based and distributed detection scheme for heavy hitters. The authors make use of counters for the match key values, e.g., source and destination IP pair or 5-tuple, that are maintained by P4 switches. If a counter exceeds a certain threshold, the P4 switch sends a notification to the controller. The controller generates more accurate status reports by combining the notifications received from the switches.

Kucera et al. [194] describe a system for detecting traffic aggregates. The authors propose a novel algorithm that supports hierarchical heavy hitter detection, change detection, and super-spreader detection. The complete mechanism is implemented on the P4 data plane and uses push notifications to a controller.

IDEAFIX [195] is a system that detects elephant flows at edge switches of Internet exchange point networks. The proposed system analyzes flow features, stores them with hash keys as indices in P4 registers, and compares them to thresholds for classification.

Turkovic et al. [196] propose a streaming approach for detecting heavy hitters via sliding windows that are implemented in P4. According to the authors, interval methods that are typically used to detect heavy hitters are not suitable

Table 5: Statistics of scientific publications regarding applied research conducted with P4.

Venue	**#Publications**
Journals	**41**
IEEE ACCESS	9
IEEE/ACM ToN	7
IEEE TNSM	6
JNCA	4
Miscellaneous	15
Conferences	**168**
ACM SOSR	14
IEEE NFV-SDN	12
IEEE ICNP	12
IEEE ICC	10
ACM SIGCOMM	10
IEEE/IFIP NOMS	8
ACM CoNEXT	7
IEEE NetSoft	7
USENIX NSDI	6
IEEE INFOCOM	6
ACM/IEEE ANCS	5
IFIP Networking	5
IEEE GLOBECOM	4
CNSM	4
IEEE CloudNet	3
APNOMS	3
IFIP/IEEE IM	3
Miscellaneous	49
Workshops	**36**
EuroP4	11
Morning Workshop on In-Network Computing	5
SPIN	3
ACM HotNets	3
INFOCOM Workshops	3
Miscellaneous	11

Table 6: Overview of applied research on monitoring (Section 9).

Research work	**Year**	**Targets**	**Code**
Detection of Heavy Hitters (Section 9.1)			
HashPipe [189]	2017	bmv2	[190]
Lin et al. [191]	2019	Tofino	
Popescu et al. [192]	2017	-	
Harrison et al. [193]	2018	Tofino	
Kucera et al. [194]	2020	bmv2	
IDEAFIX [195]	2018	-	
Turkovic et al. [196]	2019	Netronome	
Ding et al. [197]	2020	bmv2	[198]
Flow Monitoring (Section 9.2)			
TurboFlow [199]	2018	Tofino, Netronome	[200]
∗Flow [201]	2018	Tofino	[202]
Hill et al. [203]	2018	bmv2	
FlowStalker [204]	2019	bmv2	
ShadowFS [205]	2020	bmv2	
FlowLens [206]	2021	bmv2, Tofino	[207]
SpiderMon [208]	2020	bmv2	
ConQuest [209]	2019	Tofino	
Zhao et al. [210]	2019	bmv2, Tofino	
Sketches (Section 9.3)			
SketchLearn [211]	2018	Tofino	[212]
MV-Sketch [213]	2020	bmv2, Tofino	[214]
Hang et al. [215]	2019	Tofino	
UnivMon [216]	2016	p4c-behavioural	
Yang et al. [217, 218]	2018/19	Tofino	[219]
Pereira et al. [220]	2017	bmv2	
Martins et al. [221]	2018	bmv2	
Lai et al. [222]	2019	Tofino	
Liu et al. [223]	2020	Tofino	
SpreadSketch [224]	2020	Tofino	[225]

Research work	**Year**	**Targets**	**Code**
In-Band Network Telemetry (Section 9.4)			
Vestin et al. [226]	2019	Netronome	
Wang et al. [227]	2019	Tofino	
IntOpt [228]	2019	P4FPGA	
Jia et al. [229]	2020	bmv2	[230]
Niu et al. [231]	2019	Tofino, Netronome	
CAPEST [232]	2020	bmv2	[233]
Choi et al. [234]	2019	bmv2	
Sgambelluri et al. [235]	2020	bmv2	
Feng et al. [236]	2020	Netronome	
IntSight [237]	2020	bmv2, NetFPGA-SUME	[238]
Suh et al. [239]	2020	-	
DSL-Based Monitoring Systems (Section 9.5)			
Marple [240, 241]	2017	bmv2	[242]
MAFIA [243]	2019	bmv2	[244]
Sonata [245]	2018	bmv2, Tofino	[246]
Teixeira et al. [247]	2020	bmv2, Tofino	
Path Tracking (Section 9.6)			
UniRope [248]	2018	bmv2, PISCES	
Knossen et al. [249]	2019	Netronome	
Basuki et al. [250]	2020	bmv2	
Other Areas of Application (Section 9.7)			
BurstRadar [251]	2018	Tofino	[252]
Dapper [253]	2017	-	
He et al. [254]	2018	Tofino	
Riesenberg et al. [255]	2019	bmv2	[256]
Wang et al. [257]	2020	Tofino	
P4STA [258]	2020	bmv2, Netronome	[259]
Hark et al. [260]	2019	-	
P4Entropy [261]	2020	bmv2	[262]
Taffet et al. [263]	2019	bmv2	
NetView [264]	2020	bmv2, Tofino	
FastFE [265]	2020	Tofino	
Unroller [266]	2020	bmv2, Netcope P4-to-VHDL	
Hang et al. [267]	2019	Tofino	
FlowSpy [268]	2019	bmv2	

for programmable data planes because of high hardware resources, bad accuracy, or a need for too much intervention by the control plane.

Ding et al. [197] propose an architecture for network-wide heavy hitter detection. The authors' main focuses are hybrid SDN/non-SDN networks where programmable devices are deployed only partially. To that end, they also present an algorithm for an incremental deployment of programmable devices with the goal of maximizing the number of network flows that can be monitored.

9.2. Flow Monitoring

In flow monitoring, traffic is analyzed on a per-flow level. Network devices are configured to export per-flow information, e.g., packet counters, source and target IP addresses, ports, or protocol types, as flow records to a flow collector. These flow records are often duplicates of network packets without payload data. The flow collector then performs centralized analysis on this data. The three most widely deployed protocols are Netflow [270], sFlow [271], and IPFIX [272].

TurboFlow [199] is a flow record generator designed for P4 switches that does not have to make use of sampling or mirroring. The data plane generates micro-flow records with information about the most recent packets of a flow. On the CPU module of the switch, those micro-flow records are aggregated and processed into full flow records.

"$*$Flow" [201] partitions measurement queries between the data plane and a software component. A switching ASIC computes grouped packet vectors that contain a flow identifier and a variable set of packet features, e.g. packet size and timestamps, while the software component performs aggregation. "$*$Flow" supports dynamic and concurrent measurement applications, i.e., measurement applications that operate on the same flows without impacting each other.

Hill et al. [203] implement Bloom filters on P4 switches to prevent sending duplicate flow samples. Bloom filters are a probabilistic data structure that can be used to check whether an entry is present in a set or not. It is possible to add elements to that set, but it is not possible to remove entries from it. For flow tracking, Bloom filters test if a flow has been seen before without control plane interaction. Thereby, only flow data is forwarded to the collector from flows that were not seen before.

FlowStalker [204] is a flow monitoring system running on the P4 data plane. The monitoring operations on a packet are divided in two phases, a proactive phase that identifies a flow and keeps a per-flow packet counter and a reactive phase that runs for large flows only and gathers metrics of the flow, e.g., byte counts and packet sizes. The controller gathers information from a cluster of switches by injecting a crawler packet that travels through the cluster at one switch. ShadowFS [205] extends FlowStalker with a mechanism to increase the throughput of the monitored flows. It achieves this by dividing forwarding tables into two tables, a faster and a slower one. The most utilized flows are moved to the faster table if necessary.

FlowLens [206] is a system for traffic classification to support security network applications based on machine learning algorithms. The authors propose a novel memory-efficient representation for features of flows called *flow marker*. A profiler running in the control plane automatically generates an application-specific flow marker that optimizes the trade-off between resource consumption and classification accuracy, according to a given criterion selected by the operator.

SpiderMon [208] monitors network performance and debugs performance failures inside the network with little overhead. To that end, SpiderMon monitors every flow in the data plane and recognizes if the accumulated latency exceeds a certain threshold. Furthermore, SpiderMon is able to trace back the path of interfering flows, allowing to analyze the cause of the performance degradation.

ConQuest [209] is a data plane mechanism to identify flows that occupy large portions of buffers. Switches maintain snapshots of queues in registers to determine the contribution to queue occupancy of the flow of a received packet.

Zhao et al. [210] implement flow monitoring using hash tables. Using a novel strategy for collision resolution and record promotion, accurate records for elephant flows and summarized records for other flows are stored.

9.3. Sketches

Flow monitoring as described in Section 9.2 requires high sampling rates to produce sufficiently detailed data. As an alternative, streaming algorithms process sequential data streams and are subject to different constraints like limited memory or processing time per item. They approximate the current network status based on concluded summaries of the data stream. The streaming algorithms output so-called sketches that contain summarized information about selected properties of the last n packets of a flow.

SketchLearn [211] is a sketch-based approach to track the frequency of flow records. It features multilevel sketches that aim for small memory usage, fast per-packet processing, and real-time response. Rather than finding the perfect resource configuration for measurement traffic and regular traffic, SketchLearn characterizes the statistical error of resource conflicts based on Gaussian distributions. The learned properties are then used to increase the accuracy of the approximated measurements.

Tang et al. [213] present MV-Sketch, a fast and compact invertible sketch. MV-Sketch leverages the idea of majority voting to decide whether a flow is a heavy hitter or heavy changer. Evaluations show that MV-Sketch achieves a 3.38 times higher throughput than existing invertible sketches.

Hang et al. [215] try to solve the problem of inconsistency when a controller needs to collect the data from sketches on one or more switches. As accessing and clearing the sketches on the switches is always subject to latency, not all sketches are reset at the same time, and there might be some delay between accessing and clearing the sketches. The authors propose to use two asymmetric sketches on the switches that are used in an interleaved way. Furthermore, the authors propose to use a distributed control plane to keep latency low.

UnivMon [216] is a flow monitoring system based on sketches. After sampling the traffic, the data plane produces sketches and determines the top-k heaviest flows by comparing the number of sketches for each flow. Those flows are passed to the control plane which processes the data for the specific application.

Yang et al. [217, 218] propose to adapt sketches according to certain traffic characteristics to increase data accuracy, e.g., during congestion or distributed denial of service (DDoS) attacks. The mechanism is based on compressing and merging sketches when resources in the network are limited due to high traffic volume. During periods with high packet rates, only the information of elephant flows is recorded to trade accuracy for higher processing speed.

Pereira et al. [220] propose a secured version of the Count-Min sketch. They replace the function with a cryptographic hash function and provide a way for secret key renewal.

Martins et al. [221] introduce sketches for multi-tenant environments. The authors implement bitmap and counter-array sketches using a new probabilistic data structure called BitMatrix that consists of multiple bitmaps that are stored in a single P4 register.

Lai et al. [222] use a sketch-based approach to estimate the entropy of network traffic. The authors use CRC32 hashes of header fields as match keys for match-action tables and subsequently update k-dimensional data sketches in registers. The content of the registers is then processed by the control plane CPU which calculates the entropy value.

Liu et al. [223] use sketches for performance monitoring. They introduce lean algorithms to measure metrics like loss or out-of-order packets.

SpreadSketch [224] is a sketch data structure to detect superspreaders. The sketch data structure is invertible, i.e., it is possible to extract the identification of superspreaders from the sketch at the end of an epoch.

9.4. In-Band Network Telemetry

Barefoot Networks, Arista, Dell, Intel and VMware specified in-band network telemetry (INT) specifically for P4 [273]. It uses a pure data plane implementation to collect telemetry data from the network without any intervention by the control plane. It was specified by INT is the main focus of the *Applications WG* [274] of the P4 Language Consortium. Instructions for INT-enabled devices that serve as traffic sources are embedded as header fields either into normal packets or into dedicated probe packets. Traffic sinks retrieve the results of instructions to traffic sources. In this way, traffic sinks have access to information about the data plane state of the INT-enabled devices that forwarded the packets containing the instructions for traffic sources. The authors of the INT specification name network troubleshooting, advanced congestion control, advanced routing, and network data plane verification as examples for high-level use cases.

In two demos, INT was used for diagnosing the cause of latency spikes during HTTP transfers [275] and for enforcing QoS policies on a per-packet basis across a metro network [276].

Vestin et al. [226] enhance INT traffic sinks by event detection. Instead of exporting telemetry items of all packets to a stream processor, exporting has to be triggered by an event. Furthermore, they implement an INT report collector for Linux that can stream telemetry data to a Kafka cluster.

Wang et al. [227] design an INT system that can track which rules in MATs matched on a packet. The resulting data is stored in a database to facilitate visualization in a web UI.

IntOpt [228] uses INT to monitor service function chains. The system computes minimal monitoring flows that cover all desired telemetry demands, i.e., the number of INT-sources, sinks, and forwarding nodes that are covered by this flow is minimal. IntOpt uses active probing, i.e., monitoring probes for the monitoring flows are periodically inserted into the network.

Jia et al. [229] use INT to detect gray failures in data center networks using probe packets. Gray failures are failures that happen silently and without notification.

Niu et al. [231] design a multilevel INT system for IP-over-optical networks. Their goal is to monitor both the IP network and the optical network at the same time. To that end, they implement optical performance monitors for bandwidth-variable wavelength selective switches. Their measurements can be queried by a P4 switch that is connected directly to it.

CAPEST [232] leverages P4-enabled switches to estimate the network capacity and available bandwidth of network links. The approach is passive, i.e., it does not disturb the network. A controller sends INT probe packets to trigger statistical analysis and export results.

Choi et al. [234] leverage INT for run-time performance monitoring, verification, and healing of end-to-end services. P4-capable switches monitor the network based on INT information and the distributed control plane verifies that SLAs and other metrics are fulfilled. They leverage metric dynamic logic (MDL) to specify formal assertions for SLAs.

Sgambelluri et at. [235] propose a multi-layer monitoring system that uses an OpenConfig NETCONF agent for the optical layer an P4-based INT for the packet layer. In their prototype, they use INT to measure the delay of packets by computing the processing time at each switch.

Feng et al. [236] implement an INT sink for Netronome Smart NICs. After parsing the INT headers using P4, they use algorithms written in C to perform INT tasks like aggregation and notification. Compared to a pure P4 implementation, this increases the performance.

IntSight [237] is a system for detecting and analyzing violations of service-level objects (SLOs). SLOs are performance guarantees towards a network, e.g., concerning bandwidth and latency. IntSight uses INT to monitor the performance of the network during a specific period of time. Egress devices gather this information and produce a report at the end of the period if an SLO has been violated.

Suh et al. [239] explore how a sampling mechanism can be added to INT. Their solution supports rate-based and event-based sampling. Based on these sampling strategies, INT headers are only added to a fraction of the packets to reduce overhead.

9.5. DSL-Based Monitoring Systems

Monitoring tasks can often be broken down in a set of several basic operations, e.g., map, filter, or groupby. A domain-specific language (DSL) allows to combine these basic operations in more complex tasks.

Marple [240, 241] is a performance query language that supports existing constructs like map, filter, groupby, and zip. A query compiler translates the queries either to P4 or to a simulator for programmable switch hardware. Stateless constructs of the query language, e.g., filters, are executed on the data plane. Stateful constructs, e.g., groupby, use a programmable key-value store that is split between a fast on-chip SRAM cache and a large off-chip DRAM backing store. The results are streamed from the switch to a collection server.

MAFIA [243] is a DSL to describe network measurement tasks. They identify several fundamental primitive operations, examples are match, tag, timestamp, sketch, or counter. MAFIA is a high-level language to describe more complex measurement tasks composed of those primitives. The authors provide a Python-based compiler that translates MAFIA code into a P4 program in $P4_{14}$ or $P4_{16}$ for a PISA-based P4 target.

Sonata [245] is a query-driven telemetry system. It provides a query interface that provides common operators like map and reduce that can be applied on arbitrary packet fields. Sonata combines the capabilities of both programmable switches and stream processors. The queries are partitioned between the programmable switches and the stream processors to reduce the load on the stream processors. Teixeira et al. [247] extend the Sonata prototype by functionalities to monitor the properties of packet processing inside switches, e.g., delay.

9.6. Path Tracking

In path tracking, or packet trajectory tracing, information about the path a packet has taken in a network is gathered.

UniRope [248] consists of two different algorithms for packet trajectory tracing that can be selected dynamically to be able to choose the trade-off between accuracy and efficiency. These two algorithms are *compact hash matching* and *consecutive bits filling*. With compact hash matching, the forwarding switch calculates a hash value and stores it in the packet. With consecutive bits filling, the packet trajectory is recorded in the packet hop by hop and reconstructed at the controller.

Knossen et al. [249] present two different approaches for path tracking in P4. In *hop recording*, all forwarding P4 nodes record their ID in the header of the target packet. The last node can then reconstruct the path. In *forwarding state logging*, the first P4 node records the current version of the global forwarding state of the network and its node identifier in a header of the target packet. If the version of the global forwarding state does not change while the packet flows through the network, the last P4 node in the network can reconstruct the path using the information in the header.

Basuki et al. [250] propose a privacy-aware path-tracking mechanism. Their goal is that the trajectory information in the packets cannot be used to draw conclusions about the network topology or routing information. They achieve this by recording the information in an in-packet bloom filter.

9.7. Other Fields of Application

BurstRadar [251] is a system for microburst detection for data center networks that runs directly on P4 switches. If queue-induced delay is above a certain threshold, BurstRadar reports a microburst and creates a snapshot of the telemetry information of involved packets. This telemetry information is then forwarded to a monitoring server. As it is not possible to gather telemetry information of packets that are already part of the egress queue, the telemetry information of all packets and their corresponding egress port are temporarily stored in a ring buffer that is implemented using P4 registers.

Dapper [253] is a P4 tool to evaluate TCP. It implements TCP in P4 and analyzes header fields, packets sizes, and timestamps of data and ACK packets to detect congestion. Then, flow-dependent information are stored in registers.

He et al. [254] propose an adaptive expiration timeout mechanism for flow entries in P4 switches. The switches implement a mechanism to detect the last packet of a TCP flow. In case of a match, it notifies the controller to delete the corresponding flow entries.

Riesenberg et al. [255] implement alternate marking performance measurement (AM-PM) for P4. AM-PM measures delay and packet loss in-band in a network using only one or two bit overhead per packet. These bits are used for coordination and signalling between measurement points (MPs).

Wang et al. [257] describe how TCP-friendly meters can be designed and implemented for P4-based switches. According to their findings, meters in commercial switches interact with TCP streams in such a way that these streams can only reach about 10% of the target rate. The experimental evaluation of their TCP-friendly meters shows achieved rates of up to 85% of the target rate.

P4STA [258] is an open-source framework that combines software-based traffic load generation with accurate hardware packet timestamps. Thereby, P4STA aggregates multiple traffic flows to generate high traffic load and leverage programmable platforms.

Hark et al. [260] use P4 to filter data plane measurements. To save resources, only relevant measurements are sent to the controller. The authors implement a prototype and demonstrate the system by filtering measurements for a bandwidth forecast application.

P4Entropy [261] presents an algorithm to estimate the entropy of network traffic within the P4 data plane. To that end, they also developed two new algorithms, P4Log and P4Exp, to estimate logarithms and exponential functions within the data plane as well.

Taffet et al. [263] describe a P4-based implementation of an in-band monitoring system that collects information about the path of a packet and whether it encountered congestion. For this purpose, the authors repurpose previously unused fields of the IP header.

NetView [264] is a network telemetry framework that uses proactive probe packets to monitor devices. Telemetry targets, frequency, and characteristics can be configured on demand by administrators. The probe packets traverse arbitrary paths by using source routing.

FastFE [265] is a system for offloading feature extraction, i.e., deriving certain information from network traffic, for machine learning (ML)-based traffic analysis applications. Policies for feature extraction are defined as sequential programs. A policy enforcement engine translates these policies into primitives for either a programmable switch or a program running on a commodity server.

Unroller [266] detects routing loops in the data plane in real-time. It achieves this by encoding a subset of the path that a packet takes into the packet.

Hang et al. [267] use a time-based sliding window approach to measure packet rates. The goal is to record statistics entirely inside the data plane without having to use the CPU of a switch. Their approach is able to measure traffic size without sampling.

FlowSpy [268] is a network monitoring framework that uses load balancing. Different monitoring tasks are distributed among all available switches by an ILP solver. This reduces the workload on single switches in contrast to monitoring frameworks that perform all monitoring tasks on ingress or egress switches only.

9.8. Summary and Analysis

This research domain greatly benefits from all five core features described in Section 8.1. *Definition and usage of custom packet headers* enables new monitoring schemes where relevant information can be added to packets while it travels through a P4-enabled network. One example is In-band Network Telemetry (INT) (Section 9.4) that has been specified specifically for P4. Another example are path tracking mechanisms (Section 9.6) where the path of a packet is recorded in a dedicated header of the packet. In the case of INT, this goes hand in hand with *flexible packet header processing* as INT headers may contain instructions that other INT-enabled switches need to execute. *Target-specific packet header processing functions* in the form of stateful packet processing using, e.g., registers, is used by all areas of monitoring as it is necessary to gather data over a certain time frame instead of just looking at a single packet. Because the register space is severely limited on most hardware targets, an efficient usage of the available resources is of great importance. Sketches (Section 9.3) is one approach to solve this. After monitoring data is gathered on the control plane, the result is often *processed on the control plane*. This can range from simple notifications to splitting operations between data plane and control plane where the resources on the data plane are not sufficient. Some DSL-based monitoring approaches (Section 9.5) make use of *flexible development and deployment*. With these approaches, a P4 program is generated automatically on the basis of a monitoring workflow defined by an administrator.

10. Applied Research Domains: Traffic Management and Congestion Control

We describe applied research on data center switching, load balancing, congestion notification, traffic scheduling, traffic aggregation, active queue management (AQM), and traffic offloading. Table 7 shows an overview of all the work described. At the end of the section, we summarize the work and analyze it with regard to P4's core features described in Section 8.1.

10.1. Data Center Switching

Trellis [277, 278] is an open-source multipurpose L2/L3 spine-leaf switch fabric for data center networks. It is designed to run on white box switches in conjunction with the ONOS controller where its main functionality is implemented. It supports typical data center functionality such as bridging using VLANs, routing (IPv4/IPv6 unicast/multicast routing, MPLS segment routing), and vRouter functionality (BGBv4/v6, static routes, route black-holing). Trellis is part of the CORD platform that leverages SDN, network function virtualization (NFV), and Cloud technologies for building agile data centers for the network edge.

DC.p4 [280] implements typical features of data center switches in P4. The list of features includes support for VLAN, NVGRE, VXLAN, ECMP, IP forwarding, access control lists (ACLs), packet mirroring, MAC learning, and packet-in/-out messages to the control plane.

Fabric.p4 is [282, 278] the underlying reference data plane pipeline implemented in P4. By introducing support for P4 switches, the authors aim at increasing the platform heterogeneity for the CORD fabric. Fabric.p4 is currently based on the V1Model switch architecture, but support for PSA is planned. It is inspired by the OpenFlow data plane abstraction (OF-DPA) and currently supports L2 bridging, IPv4/IPv6 unicast/multicast routing, and MPLS segment routing. Fabric.p4 comes with capability profiles such as *fabric* (basic profile), *spgw* (S/PGW), and INT. For control plane interaction, ONOS is extended by the P4Runtime.

RARE [284] (Router for Academia, Research & Education) is developed in the GÉANT project GN4-3 and implements a P4 data plane for the FreeRouter open-source control plane. Its feature list includes routing, bridging, ACLs, VLAN, VXLAN, MPLS, GRE, MLDP, and BIER among others.

10.2. Load Balancing

SHELL [286] implements stateless application-aware load balancing in P4. A load balancer forwards new connections to a set of randomly chosen application instances by adding a segment routing (SR) header. Each application instance makes a local decision to either decline or accept the connection attempt. After connection initiation, the client includes a previously negotiated identifier in all subsequent packets. In the prototypical implementation, the authors use TCP time stamps for communicating the identifier, alternatives are identifiers of QUIC or TCP sequence numbers.

SilkRoad [287] implements stateful load balancing on P4 switches. SilkRoad implements two tables for stateful processing. One table maps virtual IP addresses of services to server instances, another table records active connections identified by hashes of 5-tuples to forward subsequent flows. It applies a Bloom

Table 7: Overview of applied research on traffic management and congestion control (Section 10).

Research work	Year	Targets	Code
Data Center Switching (Section 10.1)			
Trellis [277, 278]	2019	bmv2	[279]
DC.p4 [280]	2015	bmv2	[281]
Fabric.p4 [282]	2018	bmv2	[283]
RARE [284]	2019	bmv2, Tofino	[285]
Load Balancing (Section 10.2)			
SHELL [286]	2018	NetFPGA-SUME	
SilkRoad [287]	2017	Tofino	
HULA [288]	2016	-	
MP-HULA [289]	2018	-	
Chiang et al. [290]	2019	bmv2	
W-ECMP [291]	2018	bmv2	
DASH [292]	2020	bmv2	
Pizzutti et al. [293, 294]	2018/20	bmv2	
LBAS [295]	2020	Tofino	
DPRO [296]	2020	bmv2	
Kawaguchi et al. [297]	2019	bmv2	
AppSwitch [298]	2017	PISCES	
Beamer [299]	2018	bmv2, NetFPGA-SUME	[300]
Congestion Notification (Section 10.3)			
P4QCN [301]	2019	bmv2	
Jiang et al. [302]	2019	-	
EECN [303]	2020	bmv2	
Chen et al. [304]	2020	bmv2	
Laraba et al. [305]	2020	bmv2	
Traffic Scheduling (Section 10.4)			
Sharma et al. [306]	2018	bmv2	
Cascone et al. [307]	2017	-	
Bhat et al. [308]	2019	bmv2	
Kfoury et al. [309]	2019	bmv2	
Chen et al. [310]	2019	Tofino	
Lee et al. [311]	2019	bmv2	
Traffic Aggregation (Section 10.5)			
Wang et al. [312]	2020	Tofino	
RL-SP-DRR [313]	2019	bmv2	

Research work	Year	Targets	Code
Active Queue Management (AQM) (Section 10.6)			
Turkovic et al. [314]	2018	bmv2, Netronome	
P4-Codel [315]	2018	bmv2	[316]
P4-ABC [317]	2019	bmv2	
P4air [318]	2020	bmv2, Tofino	
Fernandes et al. [319]	2020	bmv2	
Wang et al. [320]	2018	bmv2, Tofino	
SP-PIFO [321]	2020	Tofino	
Kunze et al. [322]	2021	Tofino	[323]
Harkous et al. [324]	2021	bmv2, Netronome	
Traffic Offloading (Section 10.7)			
Andrus et al. [325]	2019	-	
Ibanez et al. [326]	2019	NetFPGA-SUME	
Kfoury et al. [327]	2020	Tofino	
Falcon [328]	2020	Tofino	
Osiński et al. [329]	2020	Tofino	

filter to identify new connection attempts and to record those requests in registers to remember client requests that arrive while the pool of server instances changes. In [330], the accompanying demo is described.

HULA [288] implements a link load-based distance vector routing mechanism. Switches in HULA do not maintain the state for every path but the next hops. They send out probes to gather link utilization information. Probe packets are distributed throughout the network on node-specific multicast trees. The probes have a header that contains a destination field and the currently best path utilization to that destination. When a node receives a probe, it updates the best path utilization if necessary, sends one packet clone upstream back to the origin, and forwards copies along the multicast tree further downstream. This way the origin will receive multiple probe packets with different path utilization to a specific destination. Then, flowlets are forwarded onto the best currently available path to its destination.

MP-HULA [289] extends HULA by using load information for n best next hops and compatibility with multipath TCP (MP-TCP). It tracks subflows of MP-TCP with individual flowlets per sub-flow. MP-HULA aims at distributing those subflows on different paths to aggregate bandwidth. To that end, it is necessary to keep track of the best n next-hops which is done with additional registers and forwarding rules.

Chiang et al. [290] propose a cost-effective congestion-aware load balancing scheme (CCLB). In contrast to HULA, CCLB replaces only the leaf switches with programmable switches, and thus is more cost-effective. They leverage Explicit Congestion Notification (ECN) information in probe packets to recognize congestion in the network and to adapt the load balancing. CCLB further uses flowlet forwarding and is implemented for the bmv2.

W-ECMP [291] is an ECMP-based load balancing mechanism for data centers implemented for P4 switches. Weighted probabilities based on path utilization, are used to randomly choose the best path to avoid congestion. A local agent on each switch computes link utilization for the ports. Regular traffic carries an additional custom packet header that keeps track of the current maximum link utilization on a path. Based on the maximum link utilization, the switches update port weights if necessary.

DASH [292] is an adaptive weighted traffic splitting mechanism that works entirely in the data plane. In contrast to popular weighted traffic splitting strategies such as WCMP, DASH does not require multiple hash table entries. DASH splits traffic based on link weights by portioning the hash space into unique regions.

Pizzutti et al. [293, 294] implement congestion-aware load balancing for flowlets on P4 switches. Flowlets are bursts of packets that are separated by a time gap, e.g., as caused by factors such as TCP dynamics, buffer availability, or link congestion. For distributing subflows on different paths, the congestion state of the last route is stored in a register.

LBAS [295] implements a load balancer to minimize the processing latency at both load balancers and application servers. LBAS does not only reduce the processing latency at load balancers but also takes the application servers' state into account. It is implemented for the Tofino and its average response time is evaluated.

DPRO [296] combines INT with traffic engineering (TE) and reinforcement learning (RL). Network statistics, such as link utilization and switch load, are gathered using an adapted INT approach. An RL-agent inside the controller adapts the link weights based on the minimization of a max-link-utilization objective.

Kawaguchi et al. [297] implement Unsplittable flow Edge Load factor Balancing (UELB). A controller application monitors the link utilization and computes new optimal paths upon congestion. The path computation is based on the UELB problem. The forwarding is implemented in P4 for the bmv2.

AppSwitch [298] implements a load balancer for key-value storage systems. However, the focus lies on a local agent and the control plane communication with the storage server.

Beamer [299] operates in data centers and prevents interruption of connections when they are load-balanced to a different server. To that end, the Beamer controller instructs the new target server to forward packets of the load-balanced connection to the old target server until the migration phase is over.

10.3. Congestion Notification

P4QCN [301] proposes a congestion feedback mechanism where network nodes check the egress ports for congestion before forwarding packets. If a node detects congestion, it calculates a feedback value that is propagated upstream. The mechanism clones the packet that caused the congestion, updates the feedback value in the header, changes the origin of the flow, and forwards it as a feedback packet to the sender. The sender adjusts its sending rate to reduce congestion downstream. The authors describe an implementation where bmv2 is extended by P4 externs for floating-point calculations.

Jiang et al. [302] introduce a novel adjusting advertised windows (AWW) mechanism for TCP. The authors argue that the current calculation of the advertised window in the TCP header is inaccurate because the source node does not know the actual capacity of the network. AWW dynamically updates the advertised window of ACK packets to feedback the network capacity indirectly to the source nodes. Each P4 switch calculates the new AWW value and writes it into the packet header.

EECN [303] presents an enhanced ECN mechanism which piggybacks congestion information if the switch notices congestion. To that end, the ECN-Echo bit is set for traversing ACKs as soon as congestion occurs for a given flow. This enables fast congestion notification without the need for additional control traffic.

Chen et al. [304] present QoSTCP, a TCP version with adapted congestion window growth that enables rate limiting. QoSTCP is based on a marking approach similar to ECN. When a flow exceeds a certain rate, the packet gets marked with a so-called Rate-Limiting Notification (RLN) and the congestion window growth is adapted proportional to the RLN-marked packet rate. Metering and marking is done using P4.

Laraba et al. [305] detect ECN misbehavior with the help of P4 switches. They model ECN as extended finite state machine (EFSM) and store states and variables in registers. If end hosts do not conform to the specified ECN state machine, packets are either dropped or, if possible, the misbehavior is corrected.

10.4. Traffic Scheduling

Sharma et al. [306] introduce a mechanism for per flow fairness scheduling in P4. The concept is based on round-robin scheduling where each flow may send a certain number of bytes in each round. The switch assigns a round number for each arriving packet that depends on the number of sent bytes of flow in the past.

Cascone et al. [307] introduce bandwidth sharing based on sending rates between TCP senders. P4 switches use statistical byte counters to store the sending rate of each user. Depending on the recorded sending rate of the user, arriving packets are pushed into different priority queues.

Bhat et al. [308] leverage P4 switches to translate application layer header information into link-layer headers for better QoS routing. They use Q-in-Q tunneling at the edge to forward packets to the core network and present a bmv2 implementation for HTTP/2 applications, as HTTP/2 explicitly defines a Stream ID that can directly be translated in Q-in-Q tags.

Kfoury et al. [309] present a method to support dynamic TCP pacing with the aid of network state information. A P4 switch monitors the number of active TCP flows, i.e., they monitor the SYN, SYN-ACK, and ACK flags and notify senders about the current network state if a new flow starts or another terminates. To that end, they introduce a new header and show by simulations that the overall throughput increases.

Chen et al. [310] present a design for bandwidth management for QoS with SDN and P4-programmable switches. Their design classifies packets based on a two-rate three-color marker and assigns corresponding priorities to guarantee certain per flow bandwidth. To that end, they leverage the priority queuing capabilities of P4-switches based on the assigned color. Guaranteed traffic goes to a high-priority queue, best-effort traffic goes to a low-priority queue, and traffic that exceeds its bandwidth is simply dropped.

Lee et al. [311] implement a multi-color marker for bandwidth guarantees in virtual networks. Their objective is to isolate bandwidth consumption of virtual networks and provide QoS for its serving flows.

10.5. Traffic Aggregation

Wang et al. [312] introduce aggregation and dis-aggregation capabilities for P4 switches. To reduce the header overhead in the network, multiple small packets are thereby aggregated to a single packet. They leverage multiple register arrays to store incoming small packets in 32 bit chunks. If enough small packets are stored, a larger packet gets assembled with the aid of multiple recirculations; each recirculation step appends a small packet to the aggregated large packet.

RL-SP-DRR [313] is a combination of strict priority scheduling with rate limitation (RL-SP) and deficit round-robin (DRR). RL-SP ensures prioritization of high-priority traffic while DRR enables fair scheduling among different priority classes. They extend bmv2 to support RL-SP-DRR and evaluate it against strict priority queuing and no active queuing mechanism.

10.6. Active Queue Management (AQM)

Turkovic et al. [314] develop an active queue management (AQM) mechanism for programmable data planes. The switches are programmed to collect metadata associated with packet processing, e.g., queue size and load, that are used to prevent, detect, and dissolve congestion by forwarding affected flows on an alternate path. Two possible mechanisms for rerouting in P4 are described. In the first mechanism, primary and backup entries are installed in the forwarding tables and according to the gathered metadata, the suitable action is selected. The second mechanism leverages a local controller on each switch that monitors flows and installs updated forwarding rules when congestion is noticed.

P4-CoDel [315] implements the CoDel AQM mechanism specified in RFC 8289 [331]. CoDel leverages a target and an interval parameter. As long as the queuing delay is shorter than the target parameter, no packets are dropped. If the queuing delay exceeds the target by a value that is at least as large as the interval, a packet is dropped, and the interval parameter is decreased. This procedure is repeated until the queuing delay is under the target threshold again. The interval is then reset to the initial value. To avoid P4 externs, the authors use approximated calculations for floating-point operations.

P4-ABC [317] implements activity-based congestion management (ABC) for P4. ABC is a domain concept where edge nodes measure the activity, i.e., the sending rate, of each user and annotate the value in the packet header. Core nodes measure the average activity of all packets. Depending on the current queue status, the average activity, and activity value in the packet header, a drop decision is made for each packet to prevent congestion. The P4_{16} implementation for the bmv2 requires externs for floating-point calculations.

P4air [318] attempts to provide more fairness for TCP flows with different congestion control algorithms. To that end, P4air groups flows into different categories based on their congestion control algorithm, e.g., loss-, delay- and loss-delay-based. Afterwards, the most aggressive flows are punished based on the previous categorization with packet drops, delay increase, or adjusted receive windows. P4air leverages switch metrics and flow reactions, such as queuing delay and sending rate, to determine the congestion control algorithm used by the flows.

Fernandes et al. [319] propose a bandwidth throttling solution in P4. Incoming packets are dropped with a certain probability depending on the incoming rate of the flow and the defined maximum bandwidth. Rates are measured using time windows and byte counters. Fernandes et al. extend the bmv2 for this purpose.

Wang et al. [320] present an AQM mechanism for video streaming. Data packets are classified as base packets (basic image information) or enhancement packets (additional information to improve the image quality). When the queue size exceeds a certain threshold, enhancement packets are preferably dropped.

SP-PIFO [321] features an approximation of Push-In First-Out (PIFO) queues which enables programmable packet scheduling at line rate. SP-PIFO dynamically adapts the mapping between packet ranks and available strict-priority queues.

Kunze et al. [322] analyze the design of three popular AQM algorithms (RED, CoDel, PIE). They implement PIE in three different variants for Tofino-based P4 hardware targets and show that implementation trade-offs have significant performance impacts.

Harkous et al. [324] use virtual queues implemented in P4 for traffic management. A traffic classifier in the form of MATs assigns a data plane slice identifier to traffic flows. P4 registers are used to implement virtual queues for each data plane slice for traffic management.

10.7. Traffic Offloading

Andrus et al. [325] propose to offload video stream processing of surveillance cameras to P4 switches. The authors propose to offload stream processing for storage to P4 switches. In case the analytics software detected an event, it enables a multistage pipeline on the P4 switch. In the first step, video stream data is replicated. One stream is further sent to the analytics software, the other stream is dedicated to the video storage. The P4 switch filters out control packets and rewrites the destination IP address of all video packets to the video storage.

Ibanez et al. [326] try to tackle the problem of P4's packet-by-packet programming model. Many tasks, such as periodic updates, require either hardware-specific capabilities or control-plane interaction. Processing capabilities are limited to enqueue events, i.e., data plane actions are only triggered if packets arrive. To eliminate this problem, the authors propose a new mechanism for event processing using the P4 language.

Kfoury et al. [327] propose to offload media traffic to P4 switches which act as relay servers. A SIP server receives the connection request, replaces IP and port information with the relay server IP and port, and forwards the request to the receiver. Afterwards, the media traffic is routed through the relay server.

Falcon [328] offloads task scheduling to programmable switches. Job requests are sent to the switch and the switch assigns a task in first-come-first-serve order to the next executor in a pool of computation nodes. Falcon reduces the scheduling overhead by a factor of 26 and increase scheduling throughput by a factor of 25 compared to state-of-the-art schedulers.

Osinski et al. [329] present vBNG, a virtual Broadband Network Gateway (BNG). Some components, such as PPPoE session handling, are offloaded to programmable switches.

10.8. Summary and Analysis

The research domain of traffic management and congestion control benefits from three core properties of P4: *custom packet headers*, *flexible header processing* and *target-specific packet header processing functions*. Data center switching mainly relies on packet header parsing of well-known protocols, such as IPv4/v6 or MPLS. More advanced protocol solutions, such as VXLAN and BIER, can be implemented by leveraging the *flexible packet header processing* property of P4. The presented efforts on load balancing (Section 10.2) also use this property of P4 to implement novel approaches. *Target-specific packet header processing functions* such as externs are widely used in Section 10.3. Most works leverage externs such as metering and marking which may not be supported on all hardware targets. A similar phenomenon appears in Section 10.4. Here, many papers are based on priority queues. The approaches on AQM in Section 10.6 encounter similar limitations. Floating-point operations are not part of the P4 core. Some targets may provide an extern for this functionality. Multiple works avoid this problem by either using approximations or by relying on self-defined externs in software.

11. Applied Research Domains: Routing and Forwarding

We describe applied research on source routing, multicast, publish-subscribe-systems, named data networking, data plane resilience, and other fields of application. Table 8 shows an overview of all the work described. At the end of the section, we summarize the work and analyze it with regard to P4's core features described in Section 8.1.

Table 8: Overview of applied research on routing and forwarding (Section 11).

Research work	Year	Targets	Code
Source Routing (11.1)			
Lewis et al. [332]	2018	bmv2	[333]
Luo et al. [334]	2019	bmv2	[335]
Kushwaha et al. [336]	2020	Xilinx Virtex-7	
Abdelsalam et al. [337]	2020	bmv2	
Multicast (11.2)			
Braun et al. [338]	2017	bmv2	[339]
Merling et al. [340, 341]	2020/21	bmv2, Tofino	[342, 343]
Elmo [344]	2019	-	[345]
PAM [346]	2020	bmv2	
Publish/Subscribe Systems (11.3)			
Wernecke et al. [347, 348, 349, 350]	2018/19	bmv2	
Jepsen et al. [351]	2018	Tofino	
Kundel et al. [352]	2020	bmv2	[353]
FastReact-PS [354]	2020	-	
Named Data Networks (11.4)			
NDN.p4 [355, 356]	2016/18	bmv2	[357, 358]
ENDN [359]	2020	bmv2	
Data Plane Resilience (11.5)			
Sedar et al. [360]	2018	bmv2	[361]
Giesen et al. [362]	2018	Tofino, Xilinx SDNet	
SQR [363]	2019	bmv2, Tofino	[364]
P4-Protect [365]	2020	bmv2, Tofino	[366, 367]
Hirata et al. [368]	2019	-	
Lindner et al. [369]	2020	bmv2, Tofino	[370, 371]
D2R [372]	2019	bmv2	
PURR [373]	2019	bmv2, Tofino	
Blink [374]	2019	bmv2, Tofino	[375]

Research work	Year	Targets	Code
Other Fields of Applications (11.6)			
Contra [376]	2019	-	
Michel et al. [377]	2016	bmv2	
Baktir et al. [378]	2018	bmv2	
Froes et al. [379]	2020	bmv2	
QROUTE [380]	2020	bmv2	
Gimenez et al. [381]	2020	bmv2	
Feng et al. [382]	2019	bmv2	
PFCA [383]	2020	bmv2	
McAuley et al. [384]	2019	bmv2	
R2P2 [385]	2019	Tofino	[386]

11.1. Source Routing

With source routing, the source node defines the processing of the packet throughout the network. To that end, a header stack is often added to the packet to specify the operations the other network devices should execute.

Lewis et al. [332] implement a simple source routing mechanism with P4 for the bmv2. The authors introduce a header stack to specify the processing of the packet towards its destination. That header stack is constructed and pushed onto the packet by the source node. Network devices match the header segments to determine how the packet should be processed.

Luo et al. [334] implement segment routing with P4. They introduce a header which contains segments that identify certain operations, e.g., forwarding the packet towards a specific destination or over a specific link, updating header fields, etc. Network nodes process packets according to the topmost segment in the segment routing header and remove it after successful execution.

Kushwaha et al. [336] implement bitstream, a minimalistic programmable data plane for carrier-class networks, in P4 for FPGAs. The focus of bitstream is to provide a programmable data plane while ensuring several carrier-grade properties, like deterministic latencies, short restoration time, and per-service measurements. To that end, the authors implement a source routing approach in P4 which leaves the configuration of the header stack to the control plane.

The authors of [337] show a demo of segment routing over IPv6 data plane (SRv6) implementation in P4. It leverages the novel uSID instruction set for SRv6 to improve scalability and MTU efficiency.

11.2. Multicast

Multicast efficiently distributes one-to-many traffic from the source to all subscribers. Instead of sending individual packets to each destination, multicast packets are distributed in tree-like structures throughout the network.

Bit Index Explicit Replication (BIER) [387] is an efficient transport mechanism for IP multicast traffic. In contrast to traditional IP multicast, it prevents subscriber-dependent forwarding entries in the core network by leveraging a BIER header that contains all destinations of the BIER packet. To that end, the BIER header contains a bit string where each bit corresponds to a specific destination. If a destination should receive a copy of the BIER packet, its corresponding bit is activated in the bit string in BIER header of the packet. Braun et al. [338] present a demo implementation of BIER-based multicast in P4. Merling et al. [340] implement BIER-based multicast with fast reroute capabilities in P4 for the bmv2 and for the Tofino [341].

Elmo [344] is a system for scalable multicast in multi-tenant datacenters. Traditional IP multicast maintains subscriber dependent state in core devices to forward multicast traffic. This limits scalability, since the state in the core network has to be updated every time subscribers change. Elmo increases scalability of IP multicast by moving a certain subscriber-dependent state from the core devices to the packet header.

Priority-based adaptive multicast (PAM) [346] is a control protocol for data center multicast which is implemented by the authors in P4. Network administrators define different policies regarding priority, latency, completion time, etc., which are installed on the core switches. The network devices than monitor link loads and adjust their forwarding to fulfill the policies.

11.3. Publish/Subscribe Systems

Publish/subscribe systems are used for data distribution. Subscribers are able to subscribe to announced topics. Based on the subscriptions, the data packets are distributed from the source to all subscribers.

Wernecke et al. [347, 348, 349, 350] implement a content-based publish/subscribe mechanism with P4. The distribution tree to all subscribers is encoded directly in the header of the data packets. To that end, the authors introduce a header stack which is pushed onto the packet by the source. Each element in the stack consists of an ID and a value. When a node receives a packet, it checks whether the header stack contains an element with its own ID. If so, the value determines to which neighbors the packet has to be forwarded.

Jepsen et al. [351] introduce a description language to implement publish/subscriber systems. The data plane description is translated into a static pipeline and dynamic filters. The static pipeline is a P4 program that describes a packet processing pipeline for P4 switches, the dynamic filters are the forwarding rules of the match-action tables that may change during operation, e.g., when subscriptions change.

Kundel et al. [352] propose two approaches for attribute/value encoding in packet headers for P4-based publish/subscribe systems. This reduces the header overhead and facilitates adding new attributes which can be used for subscription by hosts.

FastReact-PS [354] is a P4-based framework for event-based publish/subscribe in industrial IoT networks. It supports stateful and stateless processing of complex events entirely in the data plane. Thereby, the forwarding logic can be dynamically adjusted by the control plane without the need for recompilation.

11.4. Named Data Networking

Named data networking (NDN) is a content-centric paradigm where information is requested with resource identifiers instead of destinations, e.g., IP addresses. Network devices cache recently requested resources. If a requested resource is not available, network devices forward the request to other nodes.

NDN.p4 [355] implements NDN without caching for P4. However, the implementation cannot cache requests because of P4-related limitations with stateful storage. Miguel et al. [356] leverage the new functionalities of $P4_{16}$ to extend NDN.p4 by a caching mechanism for requests and optimize its operation. The caching mechanism is implemented with P4 externs.

Enhanced NDN (ENDN) [359] is an advanced NDN architecture. It offers a larger catalog of content delivery features like adaptive forwarding, customized monitoring, in-network caching control, and publish/subscribe forwarding.

11.5. Data Plane Resilience

Sedar et al. [360] implement a fast failover mechanism without control plane interaction for P4 switches. The mechanism uses P4 registers or metadata fields for bit strings that indicate if a particular port is considered up or down. In a match-action table, the port bit string provides an additional match field to determine whether a particular port is up or down. Depending on the port status, default or backup actions are executed. The authors rely on a local P4 agent to populate the port bit strings.

Giesen et al. [362] introduce a forward error correction (FEC) mechanism for P4. Commonly, unreliable but not completely broken links are avoided. As this happens at the cost of throughput, the proposed FEC mechanism facilitates the usage of unreliable links. The concept features a link monitoring agent that polls ports to detect unreliable connections. When a packet should be forwarded over such a port, the P4 switch calculates a resilient encoding for the packet which is then decoded by the receiving P4 switch.

Shared Queue Ring (SQR) [363] introduces an in-network packet loss recovery mechanism for link failures. SQR caches recent traffic inside a queue with slow processing speed. If a link failure is detected, the cached packets can be sent over an alternative path. While P4 does not offer the possibility to store packets for a certain amount of time, the authors leverage the cloning operation of P4 to keep packets inside the buffer. If a cached packet has not yet met its delay, it gets cloned to another egress port which takes some time. This procedure is repeated until the packet has been stored for a given time span.

P4-Protect [365] implements 1+1 protection for IP networks. Incoming packets are equipped with a sequence number, duplicated, and sent over two disjoint paths. At an egress point, the first version of each packet is accepted and forwarded. As a result, a failure of a single path can be compensated without additional signaling or reconfiguration. P4-Protect is implemented for the bmv2 and the Tofino. Evaluations show that line-rate processing with 100 Gbit/s can be achieved with P4-Protect at the Tofino.

Hirata et al. [368] implement a data plane resilience scheme based on multiple routing configurations. Multiple routing configurations with disjoint paths are deployed, and a header field identifies the routing configuration according to which packets are forwarded. In the event of a failure, a routing configuration is chosen that avoids the failure.

Lindner et al. [369] present a novel prototype for in-network source protection in P4. A P4-capable switch receives sensor data from a primary and secondary sensor, but forwards only the data from the primary sensor if available. It detects the failure of the primary sensor and then transparently forwards data from a secondary sensor to the application. Two different mechanisms are presented. The *counter-based* approach stores the number of packets received from the secondary sensor since the last packet from the primary sensor has been received. The *timer-based* approach stores the time of the last arrival of a packet from the primary sensor and considers the time since then. If certain thresholds are exceeded, the P4-switch forwards the data from the secondary sensor.

D2R [372] is a data-plane-only resilience mechanism. Upon a link failure, the data plane calculates a new path to the destination using algorithms like breadth-first search and iterative deepening depth-first search. As one pipeline iteration has not enough processing stages to compute the path, recirculation is leveraged. In addition, *Failure Carrying Packets (FCP)* is used to propagate the link failure inside the network. While the authors claim that their architecture works with hardware switches, e.g., the Tofino, they only present and evaluate a bmv2 implementation.

Chiesa et al. [373] propose a primitive for reconfigurable fast ReRoute (PURR) which is a FRR primitive for programmable data planes, in particular for P4. For each destination, suitable egress ports are stored in bit strings. During packet processing, the first working suitable egress port is determined by a set of forwarding rules. Encoding based on *Shortest Common Supersequence* guarantees that only few additional forwarding rules are required.

Blink [374] detects failures without controller interaction by analyzing TCP signals. The core concept is that the behavior of a TCP flow is predictable when it is disrupted, i.e., the same packet is retransmitted multiple times. When this information is aggregated over multiple flows, it creates a characteristic failure signal that is leveraged by data plane switches to trigger packet rerouting to another neighbor.

11.6. Other Fields of Applications

Contra [376] introduces performance-aware routing with P4. Network paths are ranked according to policies that are defined by administrators. Contra applies those policies and topology information to generate P4 programs that define the behavior of forwarding devices. During runtime, probe packets are used to determine the current network state and update forwarding entries for best compliance with the defined policies.

Michel et al. [377] introduce identifier-based routing with P4. The authors argue that IP addresses are not fine-granular enough to enable adequate forwarding, e.g., in terms of security policies. The authors introduce a new header that contains an identifier token. Before sending packets, applications transmit information on the process and user to a controller that returns an identifier that is inserted into the packet header. P4 switches are programmed to forward packets based on that identifier.

Baktir et al. [378] propose a service-centric forwarding mechanism for P4. Instead of addressing locations, e.g., by IP addresses, the authors propose to use location-independent service identifiers. Network hosts write the identifier of the desired service into the appropriate header field, the switches then make forwarding decisions based on the identifier in the packet header. With this approach, the location of the service becomes less important since the controller simply updates the forwarding rules when a service is migrated or load balancing is desired.

Froes et al. [379] classify different traffic classes which are identified by a label. Packet forwarding is based on that controller-generated label instead of IP addresses. The traffic classes have different QoS properties, i.e., prioritization of specific classes is possible. To that end, switches leverage multiple queues to process traffic of different traffic classes.

QROUTE [380] is a quality of service (QoS) oriented forwarding scheme in P4. Network devices monitor their links and annotate values, e.g., jitter or delay, in the packet header so that downstream nodes can update their statistics. Furthermore, packet headers contain constraints like maximum jitter or delay. According to those values, forwarding decisions are made by the network devices.

Gimenez et al. [381] implement the recursive internet-work architecture (RINA) in P4 for the bmv2. RINA is a networking architecture which sees computer networking as a type of inter-process communication where layering should be based on scope/scale instead of function. In general, efficient implementations require hardware support. However, up to date only software-based implementations are available. The authors hope that with the advance of programmable hardware in the form of P4, hardware-based RINA will soon be possible.

Feng et al. [382] implement information-centric network (ICN) based forwarding for HTTP. To that end, they propose mechanisms to convert packets from ICN to HTTP packets and vice-versa.

PFCA [383] implements a forwarding information base (FIB) caching architecture in the data plane. To that end, the P4 program contains multiple MATs that are mapped to different memory, i.e., TCAM, SRAM, dynamic random access memory (DRAM), with different properties regarding lookup speed. Counters keep track of cache hits to move (un)popular rules to other tables.

McAuley et al. [384] present a hybrid error control booster (HEC) that can be deployed in wireless, mobile, or hostile networks that are prone to link or transport layer failures. HECs increase the reliability by applying a modified Reed-Solomon code that adds parity packets or additional packet block acknowledgments. P4 targets include an error control processor that implements this functionality. It is integrated into the P4 program as P4 extern so that the data plane can exchange HEC packets with it. A remote control plane includes the booster manager that controls HEC operations and parameters on the P4 targets via a data plane API.

R2P2 [385] is a transport protocol based on UDP for latency-critical RPCs optimized for datacenters or other distributed infrastructure. A router module implemented in P4 or DPDK is used to relay requests to suitable servers and perform load balancing. It may also perform queuing if no suitable server is available. The goal of R2P2 is to overcome problems that typically come with TCP-based RPC systems, e.g., problems with load distribution and head-of-line-blocking.

11.7. Summary and Analysis

The research domain of routing and forwarding greatly benefits from P4's core features. First, the *definition and usage of custom packet headers* enables administrators to tailor the packet header to the specific use case. Two examples are source routing (Section 11.1) and multicast (Section 11.2). Both areas leverage custom headers to define lightweight mechanisms based on additional information in the packet header which are not part of any standard protocol. Although most of the projects were developed only for the bmv2, they should be easily portable to hardware platforms as more complex, target specific operations are not required. Second, users are able to define *flexible packet header processing* depending on the information in the packet header, e.g., publish/-subscribe systems (Section 11.3), named data networks (Section 11.4), and data plane resilience (Section 11.5). Parametrized custom actions and (conditional) application of multiple MATs allow for adaptable packet processing for many specific use cases. Similar to the previous P4 core feature, most projects were developed for the bmv2 but they should be easy to transfer if no target-specific actions are used. Third, we found that many papers in the area of data plane resilience (Section 11.5) leverage *target-specific packet header processing functions.* Often registers are used to store information whether egress ports are up or down to execute backup actions if necessary. Most projects were implemented for the hardware platform Tofino. As a result, the implementations are highly target-specific and transferring them to other hardware platforms highly depends on the capabilities of the target platform and the used externs.

12. Applied Research Domains: Advanced Networking

We describe applied research on cellular networks (4G/5G), Internet of things (IoT), industrial networking, Time-Sensitive Networking (TSN), network function virtualization (NFV), and service function chains (SFCs). Table 9 shows an overview of all the work described. At the end of the section, we summarize the work and analyze it with regard to P4's core features described in Section 8.1.

12.1. Cellular Networks (4G/5G)

P4EC [388] builds a local exit for LTE deployments with cloud-based EPC services. A programmable switch distinguishes traffic and reroutes traffic for edge computing. Non-critical traffic is forwarded to the cloud-based EPC.

Table 9: Overview of applied research on advanced networking (Section 12).

Research work	**Year**	**Targets**	**Code**
Cellular Networks (4G/5G) (12.1)			
P4EC [388]	2020	Tofino	
Trellis [282]	-	-	[389]
SMARTHO [390]	2018	bmv2	
Aghdai et al. [391, 392]	2018/19	Netronome	
GRED [393]	2019	bmv2	
HDS [394]	2020	-	
Shen et al. [395]	2019	Xilinx SDNet	
Lee et al. [396]	2019	Tofino	
Ricart-Sanchez et al. [397]	2019	NetFPGA-SUME	
Singh et al. [398]	2019	Tofino	
TurboEPC [399]	2020	Netronome	
Vörös et al. [400]	20200	Tofino	
Lin et al. [401]	2019	Tofino	
Internet of Things (12.2)			
BLESS [402]	2017	PISCES	
Muppet [403]	2018	PISCES	
Wang et al. [404]	2019	Tofino	
Madureira et al. [405]	2020	bmv2	
Engelhard et al. [406]	2019	bmv2	
Industrial Networking (12.3)			
FastReact [407]	2018	bmv2	
Cesen et al. [408]	2020	bmv2	
Kunze et al. [409]	2020	Tofino, Netronome	
Time-Sensitive Networking (TSN) (12.4)			
Rüth et al. [410]	2018	Netronome	
Kannan et al. [411]	2019	Tofino	
Kundel et al. [412]	2019	Tofino	

Research work	**Year**	**Targets**	**Code**
Network Function Virtualization (NFV) (12.5)			
Kathará [413]	2018	-	
P4NFV [414]	2018	bmv2	
Osiński et al. [415]	2019	-	
Moro et al. [416]	2020	-	
DPPx [417]	2020	bmv2	
Mohammadkhan et al. [418]	2019	Netronome	
FOP4 [419, 420]	2019	bmv2, eBPF	
PlaFFE [421]	2020	Netronome	
Service Function Chains (SFCs) (12.6)			
P4SC [422, 423]	2019	bmv2, Tofino	[424]
Re-SFC [425]	2019	bmv2	
FlexMesh [426]	2020	bmv2	
P4-SFC [427]	2019	bmv2, Tofino	[428]

The Trellis switch fabric (introduced in Section 10.1) features the spgw.p4 profile [282, 278], an implementation of a Serving and PDN Gateway (SPGW) for 5G networking. ONOS runs an SPGW-u application that implements the 3GPP control and user plane separation (CUPS) protocol to create, modify, and delete GPRS tunneling protocol (GTP) sessions. It provides support for GTP en- and decapsulation, filtering, and charging.

SMARTHO [390] proposes a handover framework for 5G. Distributed units (DUs) include real-time functions for multiple 5G radio stations. Several DUs are controlled by a central unit (CU) that includes non-real-time control functions. P4 switches are part of the CU and all DU nodes. SMARTHO introduces a P4-based mechanism for preparing handover sequences for user devices that take a fixed path among 5G radio stations controlled by DUs. This decreases the overall handover time, e.g., for users traveling in a train.

Aghdai et al. [391] propose a P4-based transparent edge gateway (EGW) for mobile edge computing (MEC) in LTE or 5G networks. Delay-sensitive and bandwidth-intense applications need to be moved from data centers in the core network to the edge of the radio access network (RAN). 5G networks rely on GTP-U for encapsulating IP packets from the mobile user to the core network. IP routers in between forward packets based on the outer IP address of GTP-U frames. The authors deploy EGWs as P4 switches at the edge of the IP transport network where service operators can deploy scalable network functions or services. Each MEC service gets a virtual IP address, the P4-based EGWs parse the inner IP destination address of GTP-U. If it sees traffic targeting a virtual IP address of a MEC service, it forwards it to the IP address of one of the serving instances of the MEC application. In their follow-up work [392], the authors extend EGWs by a handover mechanism for migrating network state.

GRED [393] is an efficient data placement and retrieval service for edge computing. It tries to improve routing path lengths and forwarding table sizes. They follow a greedy forwarding approach based on DT graphs, where the forwarding table size is independent of the network size and the number of flows in the network. GRED is implemented in P4, but the authors do not specify on which target.

HDS [394] is a low-latency, hybrid, data sharing framework for hierarchical mobile edge computing. The data location service is divided into two parts: intra-region and inter-region. The authors present a data sharing protocol called Cuckoo Summary for fast data localization for the intra-region part. Further, they developed a geographic routing scheme to achieve efficient data location with only one overlay hop in the inter-region part.

Shen et al. [395] present an FGPA-based GTP engine for mobile edge computing in 5G networks. Communication between the 5G back-haul and the conventional Ethernet requires de- and encapsulation of traffic with GTP. As most network entities do not have the capability to process GTP, the authors leverage P4-programmable hardware for this purpose.

Lee et al. [396] evaluate the performance of GTP-U and SRv6 stateless translation as GPT-U cannot be replaced by SRv6 without a transition period. To that end, they implement GTP and SRv6 on P4-programmable hardware. They found that there are no performance drops if stateless translation is used and that SRv6 stateless translation is acceptable for the 5G user plane.

Ricart-Sanchez et al. [397] propose an extension for the P4-NetFPGA framework for network slicing between different 5G users. The authors extend the capabilities of the P4 pipeline and implement their mechanism on the NetFPGA-SUME. However, the authors do not provide any details about their implementation.

Singh et al. [398] present an implementation for the Evolved Packet Gateway (EPG) in the Mobile Packet Core of 5G. They show that they can offload the functionality to programmable switching ASICs and achieve line rate with low latency and jitter while scaling up to 1.7 million active users.

TurboEPC [399] presents a redesign of the mobile packet core where parts of the control plane state is offloaded to programmable switches. State is stored in MATs. The switches then process a subset of signaling messages within the data plane itself, which leads to higher throughput and reduced latency.

Vörös et al. [400] propose a hybrid approach for the next generation NodeB (gNB) where the majority of packet processing is done by a high-speed P4-programmable switch. Additional functions, such as ARQ or ciphering, are offloaded to external services such as DPDK implementations.

Lin et al. [401] enhance the Content Permutation Algorithm (eCPA) for secret permutation in 5G. Packet payloads are split into code words and shuffled according to a secret cipher. They implement eCPA for switches of the Inventec D5264 series.

12.2. Internet of Things (IoT)

BLESS [402] implements a Bluetooth low energy (BLE) service switch based on P4 that acts as a proxy enabling flexible, policy-based switching and in-network operations of IoT devices. BLE devices are strictly bound to a central device such as a smartphone or tablet. IoT usage requires cloud-based solutions where central devices connect to an IoT infrastructure. The authors propose a BLE service switch (BLESS) that is transparently inserted between peripheral and central devices and acts like a transparent proxy breaking up the peer-to-peer model. It maintains BLE link layer connections to peripheral devices within its range. A central controller implements functionalities such as service discovery, access policy enforcement, and subscription management so that features like service slicing, enrichment, and composition can be realized by BLESS.

Muppet [403] extends BLESS by supporting the Zigbee protocol in parallel to BLE. In addition to the features of BLESS, inter-protocol services between Zigbee and BLE and BLE/Zigbee and IP protocols are introduced. An example for the latter are HTTP transactions that are automatically sent out by the switch if it sees a specified set of BLE/Zigbee transactions. The data plane implementation of BLESS is extended by protocol-dependent packet parsers and processing and support for encrypted Zigbee packets via packet recirculation.

Wang et al. [404] implement aggregation and disaggregation of small IoT packets on P4 switches. For a small IoT packet, the header holds a large proportion of the packet's total size. In large streams of IoT packets, this causes high overhead. The current aggregation techniques for IoT packets are implemented by external servers or on the control plane of switches, both resulting in low throughput and added latency. Therefore, the authors propose an implementation directly on P4 switches where IoT packets are buffered, aggregated, and encapsulated in UDP packets with a custom flag-header, type, and padding. In disaggregation, the incoming packet is cloned to stripe out the single messages until all messages are separated.

Madureira et al. [405] present the *Internet of Things Protocol (IoTP)*, an L2 communication protocol for IoT data planes. The main purpose of IoTP is data aggregation at the network level. IoTP introduces a new, fixed header and is compatible with any forwarding mechanism. The authors implemented IoTP for the bmv2 and store single packets of a flow in registers until the data can be aggregated.

Engelhard et al. [406] present a system for massive wireless sensor networks. They implement a physically distributed, and logically centralized wireless access systems to reduce the impairment by collisions. P4 is leveraged as connection between a physical access point and a virtual access point. To that end, they extend the bmv2 to provide additional functionality. However, they give information about their P4 program only in form of a decision flow graph.

12.3. Industrial Networking

FastReact [407] outsources sensor data packet processing from centralized controllers to P4 switches. The sensor data is recorded in variable-length time series data stores where an additional field holds the current moving average calculated on the time series. Both data for all sensors can be polled by a central controller. For controlling actuators directly on the data plane, FastReact supports the formulation of control logic in conjunctive normal form (CNF). It is mapped to actions to either forward signal data to the controller, discard it, or directly send it to the actuator. FastReact also features failure recovery directly on the switch. For every sensor and actuator, timestamps for the last received packets along a timeout limit is recorded. If failures are detected, sensor data are forwarded following failover rules with backup actuators for particular sensors.

Cesen et al. [408] leverage P4-capable switches to move control logic to the network. Control applications reside in controllers that are responsible for emergency intervention, e.g., if a given threshold is exceeded. The connection to the controller may be faulty and, therefore, controller intervention may not be fast enough. In this work, the authors generate emergency packets, i.e., stop commands, directly in the data plane. The action is triggered if the switch receives a packet with a specific payload.

Kunze et al. [409] investigate the applicability of in-network computing to industrial environments. They offload a simple task, i.e., coordinate transformation, to different programmable P4 targets. They come to the conclusion, that, while in general possible, even simple task have heavy demands on programmable network devices and that offloading may lead to inaccurate results.

12.4. Time-Sensitive Networking (TSN)

Rüth et al. [410] introduce a scheme for implementing in-network control mechanisms for linear quadratic regulators (LQR). LQRs can be described by a multiplication of a matrix and a vector. The vector describes the control of the actuator, the matrix describes the current system state. The result of the multiplication is a control command. The destination of a switch describes a specific actuator. When a switch receives a control packet, it matches the destination of the packet onto a match-and-action table. The lookup provides the control vector for the actuator. The control vector from the lookup is then multiplied with the system state matrix that is stored in a register to calculate the control command for the actuator. The resulting control command is written into the packet header and the packet is forwarded to the target actuator.

Kannan et al. [411] introduce the Data Plane Time synchronization Protocol (DPTP) for distributed applications with computations directly on the P4 data plane. DPTP follows a request-response model, i.e., all P4 switches request the global time from a designated master switch. Therefore, each switch features a local control plane that generates time requests sent to the master switch. Additionally, the control plane handles overflows in time calculation for administration.

Kundel et al. [412] demonstrate timestamping with nanosecond accuracy. They describe a simple setup with a Tofino-based switch and a breakout cable to connect two ports of the switch. In the experiment, timestamps at the moment of sending and reception are recorded in the packet header. The authors compare those two timestamps to show that very fine-grained measurements are possible.

12.5. Network Function Virtualization (NFV)

Kathará [413] runs NFs as P4 programs either on software or hardware targets. For software-based deployment, the framework leverages Docker containers that run NFs as container images or individual setups for Quagga, Open vSwitch, or bmv2 container images. For hardware-based deployment on P4 switches, NFs are either replicated on every P4 switch or distributed on multiple P4 switches as needed. In both cases, a load balancer or service classifier forwards flows to the appropriate P4 switch. As a main advantage, P4 programs can be shifted between the bmv2-based P4 software targets and hardware targets depending on the required performance.

P4NFV [414] also deals with the idea of running NFs either on software- or hardware-based P4 targets. The authors adopt the ETSI NFV architecture with control and monitoring entities and add a layer that abstracts various types of software- and hardware-based P4 targets as P4 nodes. For optimized deployment, the targets performance characteristics are part of the P4 node description. For runtime reconfiguration, the authors propose two approaches. In pipeline manipulation, the P4 program features multiple match-action pipelines that can be enabled or disabled by setting register flags. In program reload, a new P4 program is compiled and loaded to the P4 target. The authors propose to perform state management and migration either directly on the data plane or via a control plane.

Osiński et al. [415] use P4 to offload the data plane of virtual network functions (VNFs) into a cloud infrastructure by allowing VNFs to inject small P4 programs into P4 devices like SmartNICs or top-of-rack switches. This results in better performance and a microservice-based approach for the data plane. A new P4 architecture model that integrates abstractions used to develop VNF data planes was developed.

Moro et al. [416] present a framework for NF decomposition and deployment. They split NFs into components that can run on CPUs or that can be offloaded to specific programmable hardware, e.g., P4 programmable switches. The presented orchestrator combines multiple functions into a single P4 program that can be deployed to programmable switches.

DPPx [417] implements a framework for P4-based data plane programmability and exposure which allows enhancing NFV services. They introduce data plane modules written in P4 which can be leveraged by the application plane. As an example, a dynamic optimization of packet flow routing (DOPFR) is implemented using DPPx.

Mohammadkhan et al. [418] provide a unified P4 switch abstraction framework where servers with software NFs and P4-capable SmartNICs are seen as one logical entity by the SDN controller. They further leverage Mixed Integer Linear Programming (MILP) to determine partitioning of P4 tables for optimal placement of NFs.

FOP4 [419] [420] implements a rapid prototyping platform that supports container-based, P4-switch-based, and SmartNIC-based NFs. They argue that a prototyping platform is needed to quickly develop and evaluate new NFV use cases.

PlaFFE [421] introduces NFV offloading where some features of VNFs or embedded Network Functions (eNFs) are executed on SmartNICs using P4. Additionally, P4 is used to steer traffic either through the eNFs or through VNFs using SR-IOV.

12.6. Service Function Chains (SFCs)

P4SC [422] [423] implements a SFC framework for P4 targets. SFCs are described as directed acyclic graph of service functions (SFs). In P4SC, SFs are represented by blocks. Each block has a unique identifier, a P4 program for ingress processing, and a P4 program for egress processing. P4SC includes 15 SF blocks, e.g., L2 forwarding, which are extracted from switch.p4. After the user specified all SFCs for a particular P4 target, the P4SC converter merges the directed acyclic graphs of all SFCs with an LCS-based algorithm into an intermediate representation. Then, the P4SC generator creates the final P4 program based on the intermediate representation to be deployed onto the P4 target. P4 program generation includes runtime management, i.e., the generator creates one API per SFC while hiding SF-specific details, e.g., names of particular match-and-action tables.

Re-SFC [425] improves P4SC's resource usage by using resubmit operations. If the specified order of SFs in an SFC does not match the pre-embedded SF of the P4 switch, incoming flows cannot be processed. P4SC solves this problem by permitting redundant NF embeds, i.e., if SFs of one SFC are required by another SFCs, those SFs are just replicated. To reduce the costly usage of match-and-action tables, Re-SFC introduces resubmit actions where packets are re-bounced to the ingress.

FlexMesh [426] tackles the problem of fixed SFC flow control, i.e., when the specified order of SFs does not match the pre-embedded SF, by leveraging MATs. SFs can be dynamically bypassed, and recirculation is used to build any desired SF chain.

P4-SFC [427] is an SFC framework based on MPLS segment routing and NFV. P4 is used to implement a traffic classifier. A central orchestrator deploys service functions as VNFs and configures the traffic classifier based on definitions of SFCs.

12.7. Summary and Analysis

As the research domain of advanced networking covers different topics, almost all core properties of P4 are covered. The area of cellular networks (Section 12.1) greatly benefits from the *definition and usage of custom packet headers* as many works are based on tunneling technologies, such as GTP. Further, *flexible packet header processing* allows implementing new 5G concepts such as gNB or EPG. Some use cases still require offloading tasks to specialized hardware or software by leveraging the *target-specific packet header processing function* property of P4, e.g., for ARQ or ciphering in the context of gNB. Network function virtualization (NFV) (Section 12.5) benefits from *flexible development and deployment* as single network functions (NFs) can be replaced or relocated during operation. New protocols and extensions to existing protocols presented in Section 12.6 rely on *definition and usage of custom packet headers* and *flexible packet header processing.*

13. Applied Research Domains: Network Security

We describe applied research on firewalls, port knocking, DDoS attack mitigation, intrusion detection systems, connection security, and other fields of application. Table 10 shows an overview of all the work described. At the end of the section, we summarize the work and analyze it with regard to P4's core features described in Section 8.1.

13.1. Firewalls

Ricart-Sanchez et al. [429] present a 5G firewall that analyzes GTP data transmitted between edge and core networks. P4 allows an implementation of parsing and matching GTP header fields such as 5G user source IP, 5G user destination IP, and identification number of the GTP tunnel. The P4 pipeline implements an allow-by-default policy, DROP actions for specific sets of keys can be installed via a data plane API. In a follow-up work [430], the authors extend the 5G firewall by support for multi-tenancy with VXLAN.

CoFilter [431] implements an efficient flow identification scheme for stateful firewalls in P4. To solve the problem of limited table sizes on SDN switches, flow identifiers are calculated by applying a hashing function to the 5-tuple of every packet directly on the switch. The proposed concept includes a novel hash rewrite function that is implemented on the data plane. It resolves hash commission and hash table optimization using an external server.

P4Guard [432] replaces software-based firewalls by P4-based virtual firewalls in the VNGuard [480] system. VNGuard introduces controller-based deployment and management of virtual firewalls with the help of SDN and NFV. The P4-based firewall comprises a single MAT that allows ALLOW/DROP decision for Layer 3/4 header fields as match keys. The flow statistics are recorded with the help of counters. Another MAT allows enabling/disabling the firewall at runtime.

Vörös and Kiss [433] present a firewall implemented in P4. The parser supports Ethernet, IPv4/IPv6, UDP, and TCP headers. A ban list comprises MAC address/IP address entries that represent network hosts. Packets matching this ban list are directly dropped. To mitigate port scan or DDoS attacks, counters track packet rate and byte transfer statistics. Another MAT implements whitelist filtering.

Table 10: Overview of applied research on network security (Section 13).

Research work	Year	Targets	Code
Firewalls (13.1)			
Ricart-Sanchez et al. [429, 430]	2018/19	NetFPGA-SUME	
CoFilter [431]	2018	Tofino	
P4Guard [432]	2018	bmv2	
Vörös and Kiss [433]	2016	p4c-behavioral	
Port Knocking (13.2)			
P4Knocking [434]	2020	bmv2	
Almaini et al. [435]	2019	bmv2	
DDoS Mitigation Mechanisms (13.3)			
LAMP [436]	2018	bmv2	
TDoSD@DP [437, 438]	2018/19	bmv2	
Kuka et al. [439]	2019	Xilinx UltraScale+, Intel Stratix 10	
Paolucci et al. [440, 441]	2018/19	bmv2, NetFPGA-SUME	
ML-Pushback [442]	2019	-	
Afek et al. [443]	2017	p4c-behavioral	
Cardoso Lapolli et al. [444]	2019	bmv2	[445]
Cai et al. [446]	2020	-	
Lin et al. [447]	2020	bmv2	
Musumeci et al. [448]	2020	bmv2	
DIDA [449]	2020	bmv2	
Dimolianis et al. [450]	2020	Netronome	
Scholz et al. [451]	2020	bmv2, T_4P_4S, Netronome, NetFPGA SUME	[452]
Friday et al. [453]	2020	bmv2	
NetHide [454]	2018	-	
Intrusion Detection Systems & Deep Packet Inspection (13.4)			
P4ID [455]	2019	bmv2	
Kabasele and Sadre [456]	2018	bmv2	
DeepMatch [457]	2020	Netronome	[458]
Qin et al. [459]	2020	bmv2, Netronome	[460]
SPID [461]	2020	bmv2	

Research work	Year	Targets	Code
Other Fields of Application (13.6)			
Chang et al. [462]	2019	bmv2	
Clé [463]	2019	-	
P4DAD [464]	2020	bmv2	
Chen [465]	2020	Tofino	[466]
Gondaliya et al. [467]	2020	NetFPGA SUME	
Poise [468]	2020	Tofino	[469]
Connection Security (13.5)			
P4-MACsec [470]	2020	bmv2, NetFPGA-SUME	[471]
P4-IPsec [472]	2020	bmv2, NetFPGA-SUME, Tofino	[473]
SPINE [474]	2019	bmv2	[475]
Qin et al. [476]	2020	bmv2	
P4NIS [477]	2020	bmv2	[478]
LANIM [479]	2020	bmv2	

13.2. Port Knocking

Port knocking is a simple authentication mechanism for opening network ports. Network hosts send TCP SYN packets in predefined sequences to certain ports. If the sequence is completed correctly, the server opens up a desired port. Typically, port knocking is implemented in software on servers.

P4Knocking [434] implements port knocking on P4 switches. The authors propose four different implementations for P4. In the first implementation, P4 switches track the state of knock sequences in registers where the source IP address is used as an index. The second implementation uses a CRC-hash of the source IP address as index for the knocking state registers. To resolve the problem of hash collisions, the third implementation relies on identifiers that are calculated and managed by the controller. The fourth implementation solely relies on the controller, i.e., P4 switches forward all knocking packets to the controller.

Almaini et al. [435] implement port knocking with a ticket mechanism on P4 switches. Traffic is only forwarded if the sender has a valid ticket. Predefined trusted nodes have a ticket by default, untrustworthy nodes must obtain a ticket by successful authentication via port knocking. The authors use the HIT/MISS construct of P4 as well as stateful P4 components to implement the concept. Port knocking sequences and trusted/untrusted hosts can be maintained by the control plane.

13.3. DDoS Attack Mitigation

LAMP [436] presents a cooperative mitigation mechanism for DDoS attacks that relies on information from the application layer. Ingress P4 switches add a unique identifier to the IP options header field of any processed packet. The last P4 switch ahead of the target host stores this mapping and empties the IP options header field. If a network hosts, e.g., a database server, detects an ongoing DDoS attack on the application layer, it adds an attack flag to the IP options header field and sends it back to the switch. The switch forwards this packet to the ingress switch to enable dropping of all further packets of this flow.

TDoSD@DP [437] is a P4-based mitigation mechanism for DDoS attacks targeting SIP proxies. Stateful P4 registers record the number of SIP INVITE and SIP BYE messages. Then, a simple state machine monitors sequences of INVITE and BYE messages. Many INVITES followed by zero BYE messages lead to dropping SIP INVITE packets where valid sequences of INVITE and BYE messages will keep the port open. In a follow-up work [438], the authors present an alternative approach where P4 switches act as distributed sensors. An SDN controller periodically collects data from counters of P4 switches to perform centralized attack detection. Then, attack mitigation is performed by installing DROP rules on the P4 switches.

Kuka et al. [439] present a DDoS mitigation system that targets volumetric DDoS attacks called reflective amplification attacks. The authors port an existing VHDL implementation into a P4 program that runs on FPGA targets. The implementation selects the affected subset of the incoming traffic, extracts packet data, and forwards it as a digest to an SDN controller. The SDN controller continuously evaluates this information; a heuristic algorithm identifies aggressive IP addresses by looking at the volumetric contribution of source IP addresses to the attack. In case of a detected attack, the SDN controller installs DROP rules.

Paolucci et al. [440, 441] present a stateful mitigation mechanism for TCP SYN flood attacks. It is part of a P4-based edge packet-over-optical node that also comprises traffic engineering functionality. P4 registers keep per-session statistics to detect TCP SYN flood attacks. One register records the port number of the last TCP SYN packet, the another one records the number of attempts matching the TCP SYN flood behavior. If the latter one exceeds a defined threshold, the packets are dropped.

ML-Pushback [442] proposes an extension of the Pushback DDoS attack mitigation mechanism by machine learning techniques. P4 switches implement a data collector mechanism that collects dropped packets and forwards them as digest messages to the control plane. On the control plane, a deep learning module extracts signatures and classifies the collected digest with a decision tree model. Attack mitigation is performed by throttling attacker traffic via rate limits.

Afek et al. [443] implement known mitigation mechanisms for SYN and DNS spoofing in DDoS attacks for OpenFlow and P4 targets. The OpenFlow implementation targets Open vSwitch and OpenFlow 1.5 where P4 implementations are compiled for p4c-behavioral without control plane involvement. In addition, the authors implemented a set of algorithms and methods for dynamically distributing the rule space over multiple switches.

Cardoso Lapolli et al. [444] describe an algorithmic approach to detect and stop DDoS attacks on P4 data planes. The algorithm was specifically created under the functional constraints of P4 and is based on the calculation of the Shannon entropy.

Cai et al. [446] propose a novel method for collecting traffic information to detect TCP port scanning attacks. The authors propose the "0-replacement" method as an efficient alternative to existing sampling and aggregation methods. It introduces a pending request counter (PRcounter) and relies on registers to bind hashing identifiers of the attackers' IP addresses to PRcounter values. The authors describe the concept as compliant to PSA, but only simulation results are given.

Lin et al. [447] present a comparison of OF- and P4-based implementations of basic mitigation mechanisms against SYN flooding and ARP spoofing attacks.

Musumeci et al. [448] present P4-assisted DDoS attack mitigation using an ML classifier. An ML-based DDoS attack detection module with a classifier is running on a controller. The P4 switch forwards traffic to the module; the DDoS attack detection module responds with a decision. The authors consider three use cases: packet mirroring + header mirroring + metadata extraction. In metadata extraction, P4 switches implement counters that store occurrences of IP, UDP, TCP, and SYN packets. In the case that one of the counters exceeds a defined threshold, the P4 switch inserts a custom header with the counter values and sends it to the DDoS attack detection module.

DIDA [449] presents a distributed mitigation mechanism against amplified reflection DDoS attacks. In this type of DDoS attack, spoofed requests lead to responses that are by magnitude larger. An example is a DNS ANY query. The authors rely on count-min sketch data structures and monitoring intervals to put the number of requests and responses into relation. In case of a detected DDoS attack, ACLs are used to block the traffic near to the attacker.

Dimolianis et al. [450] introduce a multi-feature DDoS detection scheme for TCP/UDP traffic. It considers the total number of incoming traffic for a particular network, the significance of the network, and the symmetry ratio of incoming and outgoing traffic for classifications. The feature analysis is time-dependent and focuses on distinct time intervals.

Scholz et al. [451] propose a SYN proxy that relies on SYN cookies or SYN authentication as protection against SYN flooding DDoS attacks. The authors present a software implementation based on DPDK and compare it to a bmv2-based P4 implementation that is ported to the T_4P_4S P4 software target, Netronome P4 hardware target, and NetFPGA SUME P4 hardware target. Evaluation results, benefits, and challenges for each platform are discussed.

Friday et al. [453] present a two-part DDoS detection and mitigation scheme. In the first part, a P4 target applies a one-way traffic analysis using bloom filters and time-dependent statistics such as moving averages. In the second part, the P4 target analyzes the bandwidth and transport protocols used by various applications to perform a volumetric analysis. The processing pipeline then decides about malicious traffic to be dropped. Administrators may supply custom network parameters used for dynamic threshold calculation that are then installed via an API on the data plane. The authors demonstrate the effectiveness of the proposed approach by three use cases: UDP amplification DDoS attacks, SYN flooding DDoS attacks, and slow DDoS attacks.

NetHide [454] prevents link-flooding attacks by obfuscating the topology of a network. It achieves this by modifying path tracing probes in the data plane while preserving their usability.

13.4. Intrusion Detection Systems (IDS) & Deep Packet Inspection (DPI)

P4ID [455] reduces intrusion detection system (IDS) processing load by apply pre-filtering on P4 switches (IDS offloading/bypassing). P4ID features a rule parser that translates Snort rules with a multistage mechanism into MAT entries. The P4 processing pipeline implements a stateless and a stateful stage. In the stateless stage, TCP/ICMP/UDP packets are matched against a MAT to decide if traffic should be dropped, forwarded to the next hop, or forwarded to the IDS. In the stateful stage, the first n packets of new flows are forwarded to the IDS. This allows that traffic targeting well-known ports can be also analyzed. Combining the feedback of the IDS for packet samples with the stateless stage is future work.

Kabasele and Sadre [456] present a two-level IDS for industrial control system (ICS) networks. The IDS targets the Modbus protocol that runs on top of TCP in SCADA networks. The first level comprises two whitelists: a flow whitelist for filtering on the TCP layer and a Modbus whitelist. If no matching entry is found for a given packet, it is forwarded to the second layer. This is in stark contrast to legacy whitelisting where packets are just dropped. In the second level, a Zeek network security analyzer acts as deep packet inspector running on a dedicated host. It analyzes the given packet, makes a decision, and instructs the controller to update filters on the switch.

DeepMatch [457] introduces deep packet inspection (DPI) for packet payloads. The concept is implemented with the help of network processors; its prototype is built with the Netronome NFP-6000 SmartNIC P4 target. The authors present regex matching capabilities that are executed in 40 Gbit/s (line rate of the platform) for stateless intra-packet matching and about 20 Gbit/s for stateful inter-packet matching. The DeepMatch functionalities are natively implemented in Micro-C for the Netronome platform and integrated into the P4 processing pipeline with the help of P4 externs.

Qin et al. [459] present an IDS based on binarized neural networks (BNN) and federated learning. BNNs compress neural networks into a simplified form that can be implemented on P4 data planes. Weights are compressed into single bits and computations, e.g., activation functions, are converted into bit-wise operations. P4 targets at the network edge then apply BNNs to classify incoming packets. To continuously train the BNNs on the P4 targets, the authors propose a federated learning scheme. Each P4 target is connected to a controller that trains an equally structured neural network with samples received from the P4 target. A cloud service aggregates local updates received from the controllers and responds with weight updates that are processed into the local model.

In the Switch-Powered Intrusion Detection (SPID) framework [461], switches compute and store flow statistics, and perform traffic change detection. If a relevant change in traffic is detected, measurement data is pushed to the control plane. In the control plane, the measurement data is fed to a ML-based anomaly detection pipeline to detect potential attacks.

13.5. Connection Security

P4-MACsec [470] presents an implementation of IEEE 802.1AE (MACsec) for P4 switches. A two-tier control plane with local switch controllers and a central controller monitor the network topology and automatically set up MACsec on detected links between P4 switches. For link discovery and monitoring, the authors implement a secured variant of LLDP that relies on encrypted payloads and sequence numbers. MACsec is directly implemented on the P4 data plane; encryption/decryption using AES-GCM is implemented on the P4 target and integrated in the P4 processing pipeline as P4 externs.

P4-IPsec [472] presents an implementation of IPsec for P4 switches. IPsec functionality is implemented in P4 and includes ESP in tunnel mode with support for different cipher suites. As in P4-MACsec, the cipher suites are implemented on the P4 target and integrated as P4 externs. In contrast to standard IPsec operation, IPsec tunnels are set up and renewed by an SDN controller without IKE. Site-to-site operation mode supports IPsec tunnels between P4 switches. Host-to-site operation mode supports roadwarrior access to an internal network via a P4 switch. To make the roadwarrior host manageable by the controller, the authors introduce a client agent tool for Linux hosts.

SPINE [474] introduces surveillance protection in the network elements by IP address obfuscation against surveillance in intermediate networks. In contrast to software-based approaches such as TOR, SPINE runs entirely on the data plane of two nodes with intermediate networks in between. It applies a one-time-pad-based encryption scheme with key rotation to encrypt IP addresses and, if present, TCP sequence and acknowledgment numbers. The SPINE nodes add a version number representing the encryption key index to each packet by which the receiving switch can select the appropriate key for decryption. The key sets required for the key rotation are maintained by a central controller.

Qin et al. [476] introduce encryption of TCP sequence numbers using substitution-boxes to protect traffic between two P4 switches. An ONOS-based controller receives the first packet of each new flow and applies security policies to decide whether the protection should be enabled. Then, it installs the necessary data in registers and updates MATs to enable TCP sequence number substitution.

P4NIS [477] proposes a scheme to protect against eavesdropping attacks. It comprises three lines of defense. In the first line of defense, packets that belong to one traffic flow are disorderly transmitted via various links. In the second line of defense, source/destination ports and sequence/acknowledgment numbers are substituted via s-boxes similar to the approach of Qin et al. [476]. The third line of defense resembles existing encryption mechanisms that are not covered by P4NIS.

LANIM [479] presents a learning-based adaptive network immune mechanism to prevent against eavesdropping attacks. It targets the Smart Identifier Network (SINET) [481], a novel, three-layer Internet architecture. LANIM applies the minimum risk ML algorithm to respond to irregular conditions and applies a policy-based encryption strategy focusing on the intent and application.

13.6. Other Fields of Application

Chang et al. [462] present IP source address encryption. It accomplishes non-linkability of IP addresses as proactive defense mechanism. Network hosts are connected to trusted P4 switches at the network edges. In between, packets are exchanged via untrusted switches/routers. The P4 switch next to the sender encrypts the sender IP address by applying an XOR operation with a hash calculated by a random number and a shared key. The P4 switch next to the receiver decrypts the original sender IP address. The mechanism includes a dynamic key update mechanism so that transformations are random.

Clé [463] proposes to upgrade particular switches in a legacy network to P4 switches that implement security network functions (SNFs) such as rule-based firewalls or IDS on P4 switches. Clé comprises a smart device upgrade selection algorithm that selects switches to be upgraded and a controller that forwards traffic streams to the P4 switches that implement SNFs.

P4DAD [464] presents a novel approach to secure duplicate address detection (DAD) against spoofing attacks. Duplicate address detection is part of NDP in IPv6 where nodes check if an IPv6 address to be applied conflicts with another node. As the messages exchanged in duplicate address detection are not authenticated or encrypted, it is vulnerable to message spoofing. As simple alternative to authentication or encryption, P4DAD introduces a mechanism to filter spoofed NDP messages. The P4 switch maintains registers to create bindings between IPv6 addresses, port numbers, and address states. Thereby, it can detect and drop spoofed NDP messages.

Chen [465] shows how AES can be implemented on Tofino-based P4 targets in P4 using MATs as lookup tables. Expansion of the AES key is performed in the control plane. MAT entries specific to the encryption keys are generated by a controller.

Gondaliya et al. [467] implement six known mechanisms against IP address spoofing for the NetFPGA SUME P4 target. Those are Network Ingress Filtering, Reverse Path Forwarding (Loose, Strict and Feasible), Spoofing Prevention Method (SPM), and Source Address Validation Improvement (SAVI). The authors compare the different mechanisms with regard to resource usage on the FPGA and report that the implementations of all mechanisms achieve a throughput of about 8.5 Gbit/s and a processing latency of about 2 µs per packet.

Poise [468] introduces context-aware policies for securing P4-based networks in BYOD scenarios. Instead of relying on a remote controller or software-based solution, Poise implements context-aware policy enforcement directly on P4 targets. Network administrators define context-aware security policies in a declarative language based on Pyretic NetCore that are then compiled into P4 programs to be executed on P4 targets. BYOD clients run a context collection module that adds context information headers to network packets. The P4 program generated by Poise then parses and uses this information to enforce ACLs based on device runtime contexts. P4 targets in Poise are managed by a Poise controller that compiles the P4 programs, installs them on the P4 targets, and provides configuration data to the collection modules. The authors present a prototype including PoiseDroid, an implementation of the context collection module for Android devices.

13.7. Summary and Analysis

Several prototypes apply P4's *custom packet headers*, e.g., for building a GTP firewall for 5G networks, a DDoS attack mitigation mechanism for the SIP, or an IDS for the Modbus protocol in industrial networks. It is also used for in-band signaling, e.g., in cooperative DDoS attack detection. All prototypes rely on *flexible packet header processing*; outstanding for this section, many of them also rely on *target-specific packet header processing functions* offered by the P4 target. Some works require custom externs, e.g., for applying MACsec or IPsec on P4 data planes. As for prototypes from the research area *Monitoring* (Section 9), many prototypes rely on registers and counters for recording statistics, e.g., for detecting attacks in DDoS mitigation or in IDSs. While custom packet headers and basic packet header processing are supported by all P4 hardware targets, the portability of prototypes using these specific functions is very limited. Several prototypes also rely on *packet processing on the control plane* where information (e.g., from blocking lists, IDS rules) is translated into MAT rules for data plane control or data received from the data plane (e.g., statistical data or packet digests) is used for runtime control. *Flexible deployment* allows to re-deploy network security programs on P4 switches in large networks.

14. Miscellaneous Applied Research Domains

This section summarizes work that falls outside of the other application domains. We describe applied research on network coding, distributed algorithms, state migration, and application support. Table 11 shows an overview of all the work described. At the end of the section, we summarize the work and analyze it with regard to P4's core features described in Section 8.1.

14.1. Network Coding

In Network Coding (NC) [521], linear encoding and decoding operations are applied on packets to increase throughput, efficiency, scalability, and resilience. Network nodes apply primitive operations, e.g., splitting, encoding, or decoding packets, to implement NC mechanisms such as multicast, forward error correction, or rerouting (resilience).

Table 11: Overview of applied research on miscellaneous research domains (Section 14).

Research work	**Year**	**Targets**	**Code**
Network Coding (Section 14.1)			
Kumar et al. [482]	2018	bmv2	[483]
Gonçalves et al. [484]	2019	bmv2	
Distributed Algorithm (Section 14.2)			
P4CEP [485]	2018	bmv2, Netronome	
DAIET [486]	2017	-	
Sankaran et al. [487]	2020	-	
Zang et al. [488]	2017	bmv2	
Dang et al. [489, 490]	2016/20	Tofino	[491]
P4BFT [492, 493]	2019	bmv2, Netronome	
SwiShmem [494]	2020	-	
SC-BFT [495]	2020	bmv2	[496]
LODGE [497]	2018	bmv2	
LOADER [498]	2020		[499]
FLAIR [500]	2020	Tofino	
State Migration (Section 14.3)			
Swing State [501]	2017	bmv2	
P4Sync [502]	2020	bmv2	[503]
Xue et al. [504]	2020	bmv2	
Kurzniar et al. [505]	2020	bmv2	
Sankaran et al. [506]	2020	NetFPGA-SUME	
Application Support (Section 14.4)			
P4DNS [507]	2019	NetFPGA SUME	[508]
P4-BNG [509]	2019	bmv2, Tofino, Netronome, NetFPGA-SUME	[510]
ARP-P4 [511]	2018	bmv2	
Glebke et al. [512]	2019	Netronome	
COIN [513]	2019	-	
Lu et al. [514]	2019	Tofino	
Yazdinejad et al. [515]	2019	bmv2	
P4rt-OVS [516]	2020	-	[517]
SwitchML [518]	2021	Tofino	[519]
SwitchAgg [520]	2019	NetFPGA-SUME	

Kumar et al. [482] implement primitive NC operations such as splitting, encoding, and decoding for a PSA software switch. This is the first introduction of NC for SDN, as fixed-function data plane switches, e.g., as in OF, did not support such operations. The authors describe details of their implementation. The open-source implementation [483] relies on clone and recirculate operations to generate additional packets for encoding and decoding operations and packet processing loops. Temporary packet buffers for gathering operations are implemented with P4 registers. However, P4 hardware targets are not considered.

Gonçalves et al. [484] implement NC operations that may use information from multiple packets during processing. The authors implement their concept for PISA in P4$_{16}$. It features multiple complex NC operations that focus on multiplications in Galois fields used for encoding and decoding operations. NC operations are implemented in P4 externs that extend the capabilities of the software switch to store a specific amount of received packets. Again, hardware targets are not considered.

14.2. Distributed Algorithms

We describe related work on event processing and in-network consensus.

14.2.1. Event Processing

Data with stream characteristics often require specific processing. For example, sensor data may be analyzed to determine whether values are within certain thresholds, or chunks of data are aggregated and preprocessed.

P4CEP [485] shifts complex event processing from servers to P4 switches so that event stream data, e.g., from sensors, is directly processed on the data plane. The solution requires several workarounds to solve P4 limitations regarding stateful packet processing.

DAIET [486] introduces in-network data aggregation where the aggregation task is offloaded to the entire network. This reduces the amount of traffic and reliefs the destination of computational load. The authors provide a prototype implementation in P4$_{14}$ but only a few details are disclosed.

Sankaran et al. [487] increase the processing speed of packets by reducing the time that is required by forwarding nodes to parse the packet header. To that end, ingress routers parse the header stack to compute a so-called unique parser code (UPC) which they add to the packet header. Downstream nodes need to parse only the UPC to make forwarding decisions.

14.2.2. In-Network Consensus

Distributed algorithms or mechanisms may require consensus to determine the right solution or processing. This includes communication between participating entities and some ways to determine the right solution.

Zhang et al. [488] propose to offload parts of the Raft consensus algorithm to P4 switches. However, the mechanisms require an additional client to run on the switch. The authors implement their application for a P4 software switch, but details are not presented.

Dang et al. [489, 490] describe a P4 implementation of Paxos, a protocol that solves consensus for distributed algorithms in a network of unreliable processors based on information exchange between switches. This work contains a detailed description of a complex P4 implementation. The authors explain all components, provide code snippets, and discuss their design choices.

P4BFT [492, 493] introduces a consensus mechanism against buggy or malicious control plane instances. The controller responses are sent to trustworthy instances which compare the responses and establish consensus, e.g., by choosing the most common response. The authors propose to offload the comparison process to the data plane.

SwiShmem [494] is a distributed shared state management layer for the P4 data plane to implement stateful distributed network functions. In high-performance environments controllers are easily overloaded when consistency of write-intensive distributed network functions, like DDoS detection, or rate limiters, is required. Therefore, SwiShmem offloads consistency mechanisms from the control plane to the data plane. Then, consistency mechanisms operate at line rate because switches process traffic, and generate and forward state update messages without controller interaction.

Byzantine fault refers to a system where consensus between multiple entities has to be established where one or more entities are unreliable. Byzantine fault tolerance (BFT) describes mechanisms that handle such faults. However, BFTs often require significant time to reach consensus due to high computational overhead to reduce uncertainty. Switch-centric BFT (SC-BFT) [495] proposes to offload BFT functionalities, i.e., time synchronization and state synchronization, into the data plane. This significantly accelerates the consensus procedure since nodes process information at line rate.

LODGE [497] implements a mechanism for switches to make forwarding decisions based on global state without control of a central instance. Developers define global state variables which are stored by all stateful data plane devices. When such a node processes a packet that changes a global state variable, the switch generates and forwards an update packet to all other stateful switches on a predefined distribution tree. LOADER [498] introduces global state to the data plane. Consensus is maintained by the data plane devices through distributed algorithms, i.e., the switches send notification messages when global state changes. This increases scalability in comparison to mechanisms where consensus is managed by a central control entity.

FLAIR [500] accelerates read operations in leader-based consensus protocols by processing the read requests in the data plane. To that end, FLAIR devices in the core maintain persistent information about pending write operations on all objects in the system. When a client submits a read request, the FLAIR switch checks whether the requested object is stable, i.e., if it has pending write operations. If the object is stable, the FLAIR switch instructs another client with a stable version of the object, to send it to the requesting client. If the object is not stable, the FLAIR switch forwards the write request to the leader.

14.3. State Migration

In Swing State [501], switches maintain state in registers that should be migrated to other nodes. For migration, state information is carried by regular packets created by the P4 clone operation throughout the network.

P4Sync [502] is a protocol to migrate data plane state between switches. Thereby, it does not require controller interaction and provides guarantees on the authenticity of the transferred state. To that end, it leverages the switch's packet generator to transfer the content of registers between devices. Authenticity in a migration operation is guaranteed by a hash chain where each packet contains the hashed values of both the current payload and the payload of the previous packet.

Xue et al. [504] propose a hybrid approach for storing flow entries to address the issue of limited on-switch memory. While some flow entries are still stored in the internal memory of the switch, some flow entries may be stored on servers. Switches access them with only low latency via remote direct memory access (RDMA).

Kuzniar et al. [505] propose to leverage programmable switches to act as in-network cache to speed up queries over encrypted data stores. Encrypted key-value pairs are thereby stored in registers.

Sankaran et al. [506] describe a system to relieve switches from parsing headers. They propose to parse headers at an ingress switch only and add a *unique parser code* to the packet that identifies the set of headers of the packet. With this information, following switches can parse relevant information from the headers without having to parse the whole header stack.

14.4. Application Support

This subsection describes work that focuses on support or implementation of existing applications and protocols.

P4DNS [507] is an in-network DNS system. The authors propose a hybrid architecture with performance-critical components in the data plane and components with flexibility requirements in the control plane. The data plane responds to DNS requests and forwards regular traffic while cache management, recursive DNS requests, and uncached DNS responses are handled by the control plane.

P4-BNG [509] implements a carrier-grade broadband network gateway (BNG) in P4. The authors aim to provide an implementation for many different targets. To that end, they introduce a layer between data plane and control plane. This hardware-specific BNG data plane controller runs directly on the targets to provide a uniform interface to the control plane. It then configures the data plane according to the control commands from the control plane.

ARP-P4 [511] implements MAC address learning based on ARP solely on the P4 data plane. To substitute a control plane, the authors integrate MAC learning as an external function.

Glebke et al. [512] propose to offload computer vision functionalities, in particular, time-critical computations, to the data plane. To that end, the authors leverage convolution filters on a P4-programmable NIC. The necessary computations are distributed to various MATs.

COordinate-based INdexing (COIN) [513] is a mechanism to ensure efficient access to data on multiple distributed edge servers. To that end, the authors introduce a centralized instance that indexes data and its associated location. When an edge server requires data that it has not cached itself, it requests the data index at the centralized instance which provides a data location.

Lu et al. [514] propose intra-network inference (INI) and implement it in P4. It offloads neural network computations into the data plane. To that end, each P4 switch communicates via USB with a dedicated neural compute stick which performs computations.

Yazdinejad et al. [515] present a P4-based blockchain enabled packet parser. The proposed architecture focuses on FPGAs and aims to bring the security characteristics of blockchains into the data plane to greatly increase processing speed.

P4rt-OVS [516] is an extension for the OVS based on BPFs to combine the programmability of P4 and the well-known features of the OVS. P4rt-OVS enables runtime programming of the OVS, in particular, the deployment of new network features without recompilation of the OVS. It contains a P4-to-BPF compiler which allows developers to write data plane code for the OVS in P4.

SwitchML [518] proposes to accelerate distributed machine learning training with programmable switch data planes. Within distributed machine learning, so-called worker nodes compute model updates on a subset of the training data. Afterwards, these model updates are synchronized and merged on the worker nodes. The authors of SwitchML design a communication primitive to perform parts of the model aggregation within the network. They evaluate their algorithm on the Tofino platform and show an increase in training performance up to a factor of 5.5.

SwitchAgg [520] proposes a switch design for in-network aggregation that solves shortcomings of common reprogrammable switches. It processes packets at line rate and drastically reduces the required network traffic for distributed algorithms. The authors implement and evaluate their switch design in Verilog HDL on a NetFPGA-SUME.

14.5. Summary and Analysis

P4 facilitates the development of prototypes in the domain of network coding (see Subection 14.1) by providing *target-specific packet header processing functions.* The prototypes heavily rely on externs to implement complex packet processing behavior, i.e., encoding and decoding operations, packet splitting and packet merging. Such prototypes were mainly developed for the bmv2 and portability to hardware platforms depends on the properties of the used externs and the capabilities of the hardware targets. Distributed algorithms (see Section 14.2) leverage all sorts of P4's core features. Some prototypes *define and use custom packet headers* to transport information that are not available in standard protocols. Others rely on *flexible packet header processing* and *target-specific packet header processing functions* to implement unconventional and complex packet processing behavior. Some prototypes require *packet processing on the control plane* to resolve consistency issues or make network-wide configuration decisions. In the context of state migration (see Section 14.3) the prototypes mainly leverage externs to enable stateful processing. As a result, most projects were developed for the bmv2 with only limited portability to hardware platforms. Finally, some prototypes reimplement traditional network protocols or network elements, e.g., DNS, BNG, or ARP. Those projects mainly *define and use custom packet headers* for information transport, *flexible packet header processing* to implement the functionality of the specific protocol or network element, *target-specific packet header processing functions* for complex packet processing, and *packet processing on the control plane* for corner cases.

15. Discussion & Outlook

We discuss the findings of this survey and present an outlook.

15.1. P4 as a Language for Programmable Data Planes

From a variety of data plane programming approaches, P4 became the currently most widespread standard. Learning resources (Section 3.8) and the bmv2 P4 software target (Section 5.1) constitute low entry barriers for P4 technology. This is appealing for academia, and hardware support on high-speed platforms make P4 relevant for industry. The large body of literature that we surveyed in this work demonstrates that P4 has the right abstractions to build prototypes for many use cases in different application domains. Moreover, P4 allows simple and flexible definition of data plane APIs (Section 6) that can be used by simple control plane programs or complex, enterprise-grade SDN controllers. Thus, P4 allows practitioners and researchers to express their data plane and control plane algorithms in a simple way and thereby unleashes a great innovation potential. As P4 is supported by multiple platforms, there is a potentially large user group. In addition, P4 is an open programming language so that the source code can be published as open source. Therefore, public P4 code can profit from a large user community, both in quantity and quality, which is a benefit for software maintenance and security.

We consider P4 as a milestone technology. It offers great flexibility and an easy, generalized, yet powerful abstraction to describe data plane behavior. Its main objective is high-speed packet header processing. Its wide support by high-speed hardware targets enables prototype development for many different use cases.

15.2. P4 Targets Revisited

We have listed many available P4 targets in Section 5. However, our literature overview showed that mostly the bmv2 development and testing platform as well as P4 hardware targets based on the Tofino ASIC were applied in the reviewed papers.

The vast majority of prototypes runs on the software switch bmv2. One reason is that it is freely available for everyone. In addition, the complexity of the code is not constrained by hardware restrictions. And finally, any required extern can be customized. Therefore, there is no limit on algorithmic complexity so that bmv2 can serve as a platform for any use case – but only from a functional point of view. As it is a pure software-based prototyping solution, it cannot provide high throughput and is, therefore, not suitable for deployment in production environments.

The Intel Tofino family of Intelligent Fabric Processor (IFP) ASICs is currently the most popular hardware target and the only programmable data plane platform with throughput rates up to 25.6 Tbit/s and ports running at up to 400 Gbit/s, making it appropriate for production environments like data centers or core networks. Tofino uses P4 as native programming language. Therefore, comprehensive tools are offered to support the P4 development process on this platform. Moreover, P4 gives access to all features of the Tofino chip so that there is no penalty of using P4 as a programming language. Existing restrictions are due to the functional limitation of a high-speed platform. Thus, prototypes for Tofino are more challenging but prove the technical feasibility of a new concept at commercial scale. Probably for these reasons the Tofino turned out to be the mostly used hardware platform in our survey.

P4 can be also used on FPGA- or NPU-based targets. They come with only a few ports and lower throughput rates so that they may be used for special-purpose server applications but not for typical switching devices. They excel through the possibility to extend the target functionality with user-defined externs. These cards are typically programmed by vendor-specific languages. P4 support is achieved by trans-compilers that translate P4 programs into the vendor-specific format. P4 programmability might be limited to a restricted feature set while access to all features of a target is only possible through the vendor-specific programming language. Whether the application of P4 for such targets is beneficial compared to vendor-specific programming languages or interfaces, mainly depends on the use case, level of knowledge of the programmer, and if prospect target-independence is a goal.

15.3. Portability, Target- and Vendor-Independence

Portability is an important advantage of using a high-level programming language such as P4. While the subject of portability is explicitly discussed in the $P4_{16}$ specification (see Section 3), practical implications are frequently misunderstood. In general, P4 programs are not expected to be portable across different P4 architectures. P4 programs written for a given P4 architecture should be portable across all P4 targets that implement the corresponding model, provided there are sufficient resources on the P4 target. Even if two P4 targets support the same P4 architecture, a P4 program written for one P4 target might not compile to the other P4 target because of the differences in the available resources. The only portability guarantee that is made is that if the program can be successfully compiled on both P4 targets, it will exhibit the same behavior and produce the same results. This guarantee is somewhat weaker, compared to what portability means in the general purpose programming languages.

There have been several efforts to define portable P4 architectures. For network switches, it is mainly the Portable Switch Architecture (PSA). Their main challenge is not in the language, but the capabilities of existing high-speed hardware. While software P4 targets such as the bmv2 have no difficulties implementing any P4 architecture, it is almost impossible to emulate a non-existing capability or provide an adaptation layer on a high-speed hardware P4 target; simply because of the lack of sufficient resources. This is especially true for any differences that can be found in fixed-function components and externs. As a result, today's efforts tend to codify the "lowest common denominator" functionality that is guaranteed to be found on multiple P4 targets while carefully avoiding codifying any behavior that might differ. This severely limits the ability of P4 programs to fully use the capabilities of the chosen P4 targets and thus almost all P4 code surveyed today tends to use native P4 architectures instead.

Portable P4 architectures still do not provide target- or vendor-independence, i.e., the ability to simply recompile a P4 program without any changes on a different P4 target for either the same or a different vendor. This is due to the fact that the availability of specific resources differs among P4 targets.

We evaluated the specific P4 targets chosen by the authors of the surveyed works. Thereby, we noticed several important trends. First, the majority of works have been implemented either for the bmv2 P4 target, P4 targets with the Tofino ASIC, or both. When both implementations were present, the authors tend to keep their implementations for the bmv2 P4 target and Tofino P4 target separate as two independent P4 programs. Quite often, the implementations are highly different and many authors had dedicated sections in their works explaining the required major changes in porting a P4 program written for the bmv2 P4 target into a P4 program for Tofino P4 targets. A number of authors specifically mention that they could implement their P4 program only on some targets but not on others. Reasons are specific hardware resource limits, e.g., number of stages, and hardware constraints, e.g., available operations and number of operations per packet. They are naturally present on all high-speed targets. Additional reasons are special externs and fixed-function functionality that are only available on specific P4 targets.

15.4. A Business Perspective for P4-Programmable Data Planes

Today, the most prevalent hardware network appliances are proprietary devices for which customized hardware and software are jointly developed.

Data plane programming breaks with this process. Programmable packet processing ASICs such as the Tofino may be sold by specialized manufacturers and integrated by other vendors with a motherboard, CPU, memory, and connectors in white box switches. The accompanying software, i.e., data plane and control plane programs, might be provided by the same vendor, a third party, or implemented by the users themselves.

Because software is developed independently of hardware, the agility of the development process can be increased, which can reduce the time to market. Hardware platforms become reusable; they can be leveraged for multiple purposes with the help of appropriate P4 programs.

Network solution providers may leverage the lowered entry barrier for customized hardware appliances to develop and sell P4 software for various P4-capable targets, at least with moderate adaptation effort. A decade of implementation experience may no longer be a prerequisite for that business.

In addition, companies with large networks and particular use cases, e.g., special applications in data centers, may use customized algorithms to overcome inefficiencies of standardized protocols or mechanisms.

Large companies can avoid vendor lock-in by acquisition of programmable components instead of black boxes. The components are assembled possibly with open-source software leveraging data plane programming, SDN, and NFV. The ACCESS 4.0 architecture [522] and the O-RAN Alliance [523] are examples. This type of disaggregation also enables cost scaling effects where off-the-shelf components are bought at moderate cost instead of expensive specialized appliances.

15.5. Outlook

P4 is a programming language for a diverse set of programmable network targets. Currently, its main practical application are high-speed switches. It is supported by Intel's Tofino ASIC, but other manufacturers like Xilinx and Pensando recently also launched P4-based products.

The many prototypes surveyed in this paper showed that there is a need for more functionality on programmable switches, which may be provided by extern functions. While they reduce portability, they enable more use cases. Examples for such extern functions are features that have been used in some of the pure software-based P4 prototypes. They encrypt and decrypt packet payload, support floating-point operations, provide flexible hash functions, or allow more complex calculations. Those externs might be provided by the target manufacturers for common use cases or integrated by users.

Hardware with a vendor-specific programming language may benefit from offering interfaces and cross-compilers for P4 together with useful extern functions. Although this may not give access to the full functionality of the platform, users with P4 programming knowledge can customize such devices for their needs without worrying about hardware details.

The biggest driver for P4 is possibly disaggregation. While currently devices from different vendors can be orchestrated by a customized controller, P4 may have the potential to extend disaggregation towards specialized appliances based on off-the-shelf programmable hardware. Hardware without an open programming interface cannot profit from that market.

16. Conclusion

In this paper, we first gave a tutorial on data plane programming with P4. We delineated it from SDN and introduced programming models with a special focus on PISA which is most relevant for P4. We provided an overview of the current state of P4 with regard to programming language, architectures, compilers, targets, and data plane APIs. We reported research efforts to advance P4 that fall in the areas of optimization of development and deployment, research on P4 targets, and P4-specific approaches for control plane operation.

In the second part of the paper, we analyzed 245 papers on applied research that leverage P4 for implementation purposes. We categorized these publications into research domains, summarized their key points, and characterized them by prototype, target platform, and source code availability. For each research domain, we presented an analysis on how works benefit from P4. To that end, we identified a small set of core features that facilitate implementations. The survey proved a tremendous uptake of P4 for prototyping in academic research from 2018 to 2021. One reason is certainly the multitude of openly available resources on P4 and the bmv2 P4 software target. They are an ideal starting point for creating P4-based prototypes, even for beginners.

The many P4-based activities which emerged only within short time show that P4 technology can speed up the evolution of computer networking. While multiple hardware targets are available, most hardware-based prototypes leverage the Tofino ASIC that is optimized for high throughput on many ports and particularly suited for data center and WAN applications. However, the majority of P4-based prototypes was implemented with the bmv2 software switch. Many of them were not ported to hardware, probably due to the complexity of their data plane algorithms and lack of required extern functions on current hardware. This may change in the future if new P4 hardware targets are available. We expect P4 to become a base technology for multiple hardware appliances, in particular in the context of disaggregation and for small-scale markets.

17. Acknowledgement

This work was partly supported by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/1-2. The authors alone are responsible for the content of this paper.

References

[1] E. Kohler, R. Morris, B. Chen, J. Jannotti, M. F. Kaashoek, The Click Modular Router, ACM Transactions on Computer Systems (TOCS) 18 (2000) 217–231.

[2] VPP/What is VPP?, `https://bit.ly/2mrxVGE`, accessed 01-20-2021 (2021).

[3] GitHub: NPL-Spec, `https://github.com/nplang/NPL-Spec`, accessed 01-20-2021 (2021).

[4] Software Defined Specification Environment for Networking (SDNet), `https://www.xilinx.com/support/documentation/backgrounders/sdnet-backgrounder.pdf`, accessed 01-20-2021 (2021).

[5] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, D. Walker, P4: Programming Protocol-independent Packet Processors, ACM SIGCOMM Computer Communications Review (CCR) 44 (2014) 87–95.

[6] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, T. Turletti, A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks, IEEE Communications Surveys & Tutorials (COMST) 16 (2014) 1617–1634.

[7] Y. Jarraya, T. Madi, M. Debbabi, A Survey and a Layered Taxonomy of Software-Defined Networking, IEEE Communications Surveys & Tutorials (COMST) 16 (2014) 1955–1980.

[8] W. Xia, Y. Wen, C. H. Foh, D. Niyato, H. Xie, A Survey on Software-Defined Networking, IEEE Communications Surveys & Tutorials (COMST) 17 (2015) 27–51.

[9] D. F. Macedo, D. Guedes, L. F. M. Vieira, M. A. M. Vieira, M. Nogueira, Programmable Networks—From Software-Defined Radio to Software-Defined Networking, IEEE Communications Surveys & Tutorials (COMST) 17 (2015) 1102–1125.

[10] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, S. Uhlig, Software-Defined Networking: A Comprehensive Survey, Proceedings of the IEEE 103 (2015) 14–76.

[11] R. Masoudi, A. Ghaffari, Software defined networks: A survey, Journal of Network and Computer Applications (JNCA) 67 (2016) 1–25.

[12] C. Trois, M. D. Del Fabro, L. C. E. de Bona, M. Martinello, A Survey on SDN Programming Languages: Toward a Taxonomy, IEEE Communications Surveys & Tutorials (COMST) 18 (2016) 2687–2712.

[13] W. Braun, M. Menth, Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices, MDPI Future Internet Journal (FI) 6 (2014) 302–336.

[14] F. Hu, Q. Hao, K. Bao, A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation, IEEE Communications Surveys & Tutorials (COMST) 16 (2014) 2181–2206.

[15] A. Lara, A. Kolasani, B. Ramamurthy, Network Innovation using OpenFlow: A Survey, IEEE Communications Surveys & Tutorials (COMST) 16 (2014) 493–512.

[16] R. Bifulco, G. Rétvári, A Survey on the Programmable Data Plane: Abstractions, Architectures, and Open Problems, in: IEEE International Conference on High Performance Switching and Routing (HPSR), 2018, pp. 1–7.

[17] E. Kaljic, A. Maric, P. Njemcevic, M. Hadzialic, A Survey on Data Plane Flexibility and Programmability in Software-Defined Networking, IEEE ACCESS 7 (2019) 47804–47840.

[18] O. Michel, R. Bifulco, G. Rétvári, S. Schmid, The Programmable Data Plane: Abstractions, Architectures, Algorithms, and Applications, ACM Computing Surveys 1 (2021).

[19] S. Kaur, K. Kumar, N. Aggarwal, A review on p4-programmable data planes: Architecture, research efforts, and future directions, Computer Communications 170 (2021).

[20] E. F. Kfoury, J. Crichigno, E. Bou-Harb, An exhaustive survey on p4 programmable data plane switches: Taxonomy, applications, challenges, and future trends, ArXiv e-prints (2021).

[21] Y. Gao, Z. Wang, A Review of P4 Programmable Data Planes for Network Security, Mobile Information Systems (2021).

[22] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, OpenFlow: Enabling Innovation in Campus Networks, ACM SIGCOMM Computer Communications Review (CCR) 38 (2008) 69–74.

[23] BESS: Berkeley Extensible Software Switch, `http://span.cs.berkeley.edu/bess.html`, accessed 01-20-2021 (2021).

[24] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, M. Horowitz, Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN, ACM SIGCOMM Conference 43 (2013) 99–110.

[25] S. Chole, A. Fingerhut, S. Ma, A. Sivaraman, S. Vargaftik, A. Berger, G. Mendelson, M. Alizadeh, S.-T. Chuang, I. Keslassy, A. Orda, T. Edsall, DRMT: Disaggregated Programmable Switching, in: ACM SIGCOMM Conference, 2017, p. 1–14.

[26] Google Presentations: P4 Tutorial, `http://bit.ly/p4d2-2018-spring`, accessed 01-20-2021 (2018).

[27] Website of the P4 Language Consortium, https://p4.org/, accessed 01-20-2021 (2021).

[28] The P4 Language Specification, https://p4.org/p4-spec/p4-14/v1.0.5/tex/p4.pdf, accessed 01-20-2021 (2018).

[29] P4 16 Language Specification (v.1.2.1, https://p4.org/p4-spec/docs/P4-16-v1.2.1.html, accessed 01-20-2021 (2020).

[30] M. Moshref, A. Bhargava, A. Gupta, M. Yu, R. Govindan, Flow-level State Transition as a New Switch Primitive for SDN, in: ACM SIGCOMM Conference, 2014, p. 61–66.

[31] G. Bianchi, M. Bonola, A. Capone, C. Cascone, OpenState: Programming Platform-independent Stateful Openflow Applications Inside the Switch, ACM SIGCOMM Computer Communications Review (CCR) 44 (2014) 44–51.

[32] A. Sivaraman, A. Cheung, M. Budiu, C. Kim, M. Alizadeh, H. Balakrishnan, G. Varghese, N. McKeown, S. Licking, Packet Transactions: High-Level Programming for Line-Rate Switches, in: ACM SIGCOMM Conference, 2016, p. 15–28.

[33] S. Pontarelli, R. Bifulco, M. Bonola, C. Cascone, M. Spaziani, V. Bruschi, D. Sanvito, G. Siracusano, A. Capone, M. Honda, F. Huici, G. Siracusano, FlowBlaze: Stateful Packet Processing in Hardware, in: USENIX Symposium on Networked Systems Design & Implementation (NSDI), 2019, p. 531–547.

[34] H. Song, Protocol-Oblivious Forwarding: Unleash the Power of SDN Through a Future-proof Forwarding Plane, in: ACM Workshop on Hot Topics in Networks (HotNets), 2013, p. 127–132.

[35] C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, D. Walker, NetKAT: Semantic Foundations for Networks, in: ACM Symposium on Principles of Programming Languages (POPL), 2014, p. 113–126.

[36] P4 Tutorial, https://github.com/p4lang/tutorials, accessed 05-05-2021 (2021).

[37] P4 Guide, https://github.com/jafingerhut/p4-guide, accessed 05-05-2021 (2021).

[38] P4 Learning, https://github.com/nsg-ethz/p4-learning, accessed 05-05-2021 (2021).

[39] Charter of the P4 Architecture WG, https://github.com/p4lang/p4-spec/blob/master/p4-16/psa/charter/P4_Arch_Charter.mdk, accessed 01-20-2021 (2021).

[40] P4_16 PSA Specification (v1.1), https://p4lang.github.io/p4-spec/docs/PSA-v1.1.0.html, accessed 01-20-2021 (2018).

[41] P4-HLIR Specification v.0.9.30, https://github.com/p4lang/p4-hlir/blob/master/HLIRSpec.pdf, accessed 01-20-2021 (2016).

[42] GitHub: p4c, https://github.com/p4lang/p4c, accessed 01-20-2021 (2021).

[43] P. G. Patra, C. E. Rothenberg, G. Pongracz, MACSAD: High Performance Dataplane Applications on the Move, in: IEEE International Conference on High Performance Switching and Routing (HPSR), 2017, pp. 1–6.

[44] Open Data Plane, https://opendataplane.org/, accessed 01-20-2021 (2021).

[45] L. Jose, M. R. N. M. Lisa Yan, Stanford University; George Varghese, Compiling Packet Programs to Reconfigurable Switches, in: USENIX Symposium on Networked Systems Design & Implementation (NSDI), 2015, p. 103–115.

[46] P. Li, Y. Luo, P4GPU: Accelerate Packet Processing of a P4 Program with a CPU-GPU Heterogeneous Architecture, in: ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2016, pp. 125–126.

[47] GitHub: p4c-behavioural, https://github.com/p4lang/p4c-behavioral/tree/master/p4c_bm, accessed 01-20-2021 (2021).

[48] GitHub: Behavioural Model Version 2 (BMv2), https://github.com/p4lang/behavioral-model, accessed 01-20-2021 (2021).

[49] P4 Behaviour Model: Why did we need BMv2, https://github.com/p4lang/behavioral-model\#why-did-we-replace-p4c-behavioral-with-bmv2, accessed 01-20-2021 (2021).

[50] GitHub: Behavioral model targets, https://github.com/p4lang/behavioral-model/blob/master/targets/README.md, accessed 01-20-2021 (2021).

[51] BMv2 Performance, https://github.com/p4lang/behavioral-model/blob/master/docs/performance.md, accessed 01-20-2021 (2021).

[52] GitHub: eBPF Backend for p4c, https://github.com/p4lang/p4c/tree/master/backends/ebpf, accessed 01-20-2021 (2021).

[53] p4c-ubpf: a New Back-end for the P4 Compiler, https://p4.org/p4/p4c-ubpf.html, accessed 01-20-2021 (2021).

[54] GitHub: p4c-xdp, `https://github.com/vmware/p4c-xdp`, accessed 01-20-2021 (2021).

[55] P4@ELTE, `http://p4.elte.hu/`, accessed 01-20-2021 (2021).

[56] S. Laki, D. Horpácsi, P. Vörös, R. Kitlei, D. Leskó, M. Tejfel, High speed packet forwarding compiled from protocol independent data plane specifications, in: ACM SIGCOMM Conference, 2016, p. 629–630.

[57] Data Plane Development Kit (DPDK), `https://www.dpdk.org/`, accessed 01-20-2021 (2021).

[58] GitHub: T4P4S, `https://github.com/P4ELTE/t4p4s`, accessed 01-20-2021 (2021).

[59] A. Bhardwaj, A. Shree, V. B. Reddy, S. Bansal, A Preliminary Performance Model for Optimizing Software Packet Processing Pipelines, in: ACM SIGOPS Asia-Pacific Workshop on System (APSys), 2017, pp. 1–7.

[60] X. Wu, P. Li, T. Miskell, L. Wang, Y. Luo, X. Jiang, Ripple: An Efficient Runtime Reconfigurable P4 Data Plane for Multicore Systems, in: International Conference on Networking and Network Applications (NaNA), 2019, pp. 142–148.

[61] M. Shahbaz, S. Choi, B. Pfaff, C. Kim, N. Feamster, N. McKeown, J. Rexford, PISCES: A Programmable, Protocol-Independent Software Switch, in: ACM SIGCOMM Conference, 2016, p. 525–538.

[62] Open vSwitch, `https://www.openvswitch.org/`, accessed 01-20-2021 (2021).

[63] GitHub: PISCES, `https://github.com/P4-vSwitch`, accessed 01-20-2021 (2021).

[64] S. Choi, X. Long, M. Shahbaz, S. Booth, A. Keep, J. Marshall, C. Kim, The Case for a Flexible Low-Level Backend for Software Data Planes, in: Asia-Pacific Workshop on Networking (APnet), 2017, p. 71–77.

[65] S. Choi, X. Long, M. Shahbaz, S. Booth, A. Keep, J. Marshall, C. Kim, PVPP: A Programmable Vector Packet Processor, in: ACM Symposium on SDN Research (SOSR), 2017, p. 197–198.

[66] Northbound Networks - Who are You?, `https://northboundnetworks.com/pages/about-us`, accessed 01-20-2021 (2021).

[67] GitHub: ZodiacFX-P4, `https://github.com/NorthboundNetworks/ZodiacFX-P4`, accessed 01-20-2021 (2021).

[68] GitHub: p4c-zodiacfx, `https://github.com/NorthboundNetworks/p4c-zodiacfx`, accessed 01-20-2021 (2021).

[69] P. Zanna, P. Radcliffe, K. G. Chavez, A Method for Comparing OpenFlow and P4, in: International Telecommunication Networks and Applications Conference (ITNAC), 2019, pp. 1–3.

[70] GitHub: P4-NetFPGA, https://github.com/NetFPGA/P4-NetFPGA-public/wiki, accessed 01-20-2021 (2021).

[71] S. Ibanez, G. Brebner, N. McKeown, N. Zilberman, The P4-NetFPGA Workflow for Line-Rate Packet Processing, in: ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), 2019, p. 1–9.

[72] N. Zilberman, Y. Audzevich, G. A. Covington, A. W. Moore, NetFPGA SUME: Toward 100 Gbps as Research Commodity, IEEE Micro 34 (2014) 32–41.

[73] Netcope P4, https://www.netcope.com/Netcope/media/content/NetcopeP4_2019_web.pdf, accessed 01-20-2021 (2021).

[74] H. Wang, R. Soulé, H. T. Dang, K. S. Lee, V. Shrivastav, N. Foster, H. Weatherspoon, P4FPGA: A Rapid Prototyping Framework for P4, in: ACM Symposium on SDN Research (SOSR), 2017, p. 122–135.

[75] GitHub: P4FPGA, https://github.com/p4fpga/p4fpga, accessed 01-20-2021 (2021).

[76] P. Benácek, V. Pu, H. Kubátová, P4-to-VHDL: Automatic Generation of 100 Gbps Packet Parsers, in: IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2016, pp. 148–155.

[77] P. Benáček, V. Puš, J. Kořenek, M. Kekely, Line Rate Programmable Packet Processing in 100Gb Networks, in: International Conference on Field Programmable Logic and Applications (FPL), 2017, pp. 1–1.

[78] J. Cabal, P. Benáček, L. Kekely, M. Kekely, V. Puš, J. Kořenek, Configurable FPGA Packet Parser for Terabit Networks with Guaranteed Wire-Speed Throughput, in: ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), 2018, p. 249–258.

[79] S. da Silva, Jeferson, Boyer, François-Raymond, Langlois, J. Pierre, P4-Compatible High-Level Synthesis of Low Latency 100 Gb/s Streaming Packet Parsers in FPGAs, in: ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), 2018, p. 147–152.

[80] M. Kekely, J. Korenek, Mapping of P4 Match Action Tables to FPGA, in: International Conference on Field Programmable Logic and Applications (FPL), 2017, pp. 1–2.

[81] R. Iša, P. Benáček, V. Puš, Verification of Generated RTL from P4 Source Code, in: IEEE International Conference on Network Protocols (ICNP), 2018, pp. 444–445.

[82] Z. Cao, H. Su, Q. Yang, J. Shen, M. Wen, C. Zhang, P4 to FPGA-A Fast Approach for Generating Efficient Network Processors, IEEE ACCESS 8 (2020) 23440–23456.

[83] Z. Cao, H. Su, Q. Yang, M. Wen, C. Zhang, A Template-based Framework for Generating Network Processor in FPGA, in: IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2019, pp. 1057–1058.

[84] Open Tofino, https://github.com/barefootnetworks/open-tofino, accessed 01-22-2021 (2021).

[85] EdgeCore Wedge 100BF-32X, `https://www.edge-core.com/productsInfo.php?cls=1&cls2=180&cls3=181&id=335`, accessed 01-20-2021 (2021).

[86] APS Networks BF2556X-1T-A1F, `https://stordirect.com/shop/switches/25g-switches/aps-networks-bf2556x-1t-a1f/`, acessed 01-22-2021 (2021).

[87] APS Networks BF6064X-T-A2F, `https://stordirect.com/shop/switches/100g-switches/aps-networks-bf6064x-t-a2f/`, acessed 01-22-2021 (2021).

[88] Netberg Aurora 610, `https://netbergtw.com/products/aurora-610/`, accessed 01-20-2021 (2021).

[89] Arista Press Release: Arista Announces New Multi-function Platform for Cloud Networking, `https://www.arista.com/en/company/news/press-release/5148-pr-20180605`, accessed 01-20-2021 (2021).

[90] Cisco Blog: Increase Flexibility with Cisco's Programmable Cloud Infrastructure, `https://blogs.cisco.com/datacenter/increase-flexibility-with-ciscos-programmable-cloud-infrastructure`, accessed 01-20-2021 (2021).

[91] SONiC - Supported Platforms, `https://azure.github.io/SONiC/Supported-Devices-and-Platforms.html`, accessed 01-20-2021 (2021).

[92] A. Seibulescu, M. Baldi, Leveraging P4 Flexibility to Expose Target-Specific Features, in: P4 Workshop in Europe (EuroP4), 2020, p. 36–42.

[93] The Pensando Distributed Services Platform, `https://pensando.io/our-platform/`, accessed 01-20-2021 (2021).

[94] Netronome: P4 Data Plane Programming, https://netronome.com/media/documents/WP_P4_Data_Plane_Programming.pdf, accessed 01-20-2021 (2018).

[95] Netronome: Programming with P4 and C, https://www.netronome.com/media/documents/WP_Programming_with_P4_and_C.pdf, accessed 09-20-2019 (2018).

[96] H. Harkous, M. Jarschel, M. He, R. Pries, W. Kellerer, Towards Understanding the Performance of P4 Programmable Hardware, in: ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2019, pp. 1–6.

[97] Apache Thrift, https://thrift.apache.org/, accessed 01-20-2021 (2021).

[98] gRPC, https://grpc.io/, accessed 01-20-2021 (2021).

[99] Google Protocol Buffers, https://developers.google.com/protocol-buffers/, accessed 01-20-2021 (2021).

[100] Charter of the P4 API WG, https://github.com/p4lang/p4-spec/blob/master/api/charter/P4_API_WG_charter.mdk, accessed 01-20-2021 (2021).

[101] P4 Runtime API Specification v.1.3.0 (2019-12-01), https://p4.org/p4runtime/spec/v1.3.0/P4Runtime-Spec.html, accessed 01-20-2021 (2020).

[102] ONOS: P4 brigade, https://wiki.onosproject.org/display/ONOS/P4+brigade, accessed 01-20-2021 (2021).

[103] OpenDaylight: P4 brigade, P4PluginDeveloperGuide, accessed 09-23-2019 (2019).

[104] B. O'Connor, Y. Tseng, M. Pudelko, C. Cascone, A. Endurthi, Y. Wang, A. Ghaffarkhah, D. Gopalpur, T. Everman, T. Madejski, J. Wanderer, A. Vahdat, Using P4 on Fixed-Pipeline and Programmable Stratum Switches, in: P4 Workshop in Europe (EuroP4), 2010, pp. 1–2.

[105] GitHub: P4tutorial, https://github.com/p4lang/tutorials/tree/master/utils/p4runtime_lib, accessed 01-20-2021 (2021).

[106] GitHub: PI Library, https://github.com/p4lang/PI, accessed 01-20-2021 (2021).

[107] GitHub: Behavioural Model - simple_switch_grpc, https://github.com/p4lang/behavioral-model/tree/master/targets/simple_switch_grpc, accessed 01-20-2021 (2021).

[108] GitHub: bmv2 Runtime CLI, `https://github.com/p4lang/behavioral-model/blob/master/tools/runtime_CLI.py`, accessed 01-20-2021 (2021).

[109] E. O. Zaballa, Z. Zhou, Graph-to-P4: A P4 Boilerplate Code Generator for Parse Graphs, in: P4 Workshop in Europe (EuroP4), 2019, pp. 1–2.

[110] Y. Zhou, J. Bi, ClickP4: Towards Modular Programming of P4, in: ACM SIGCOMM Conference Posters and Demos, 2017, p. 100–102.

[111] M. Baldi, daPIPE A Data Plane Incremental Programming Environment, in: P4 Workshop in Europe (EuroP4), 2019, pp. 1–6.

[112] M. Eichholz, E. Campbell, N. Foster, G. Salvaneschi, M. Mezini, How to Avoid Making a Billion-Dollar Mistake: Type-Safe Data Plane Programming with SafeP4, in: European Conference on Object-Oriented Programming (ECOOP), 2019, pp. 1–28.

[113] M. Riftadi, F. Kuipers, P4I/O: Intent-Based Networking with P4, in: IEEE Conference on Network Softwarization (NetSoft), 2019, pp. 438–443.

[114] L. Yu, J. Sonchack, V. Liu, Mantis: Reactive Programmable Switches, in: ACM SIGCOMM Conference, 2020, p. 296–309.

[115] J. Gao, E. Zhai, H. H. Liu, R. Miao, Y. Zhou, B. Tian, C. Sun, D. Cai, M. Zhang, M. Yu, Lyra: A Cross-Platform Language and Compiler for Data PlaneProgramming on Heterogeneous ASICs, in: ACM SIGCOMM Conference, 2020, p. 435–450.

[116] M. Riftadi, J. Oostenbrink, F. Kuipers, GP4P4: Enabling Self-Programming Networks, ArXiv e-prints (2019).

[117] D. Moro, D. Sanvito, A. Capone, FlowBlaze.p4: a library for quick prototyping of stateful SDN applications in P4, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2020, pp. 95–99.

[118] D. Moro, D. Sanvito, A. Capone, Demonstrating FlowBlaze.p4: fast prototyping for EFSM-based data plane applications, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2020, pp. 116–117.

[119] D. Moro, D. Sanvito, A. Capone, Developing EFSM-Based Stateful Applications with FlowBlaze.P4 and ONOS, in: P4 Workshop in Europe (EuroP4), 2020, p. 52–53.

[120] N. Sultana, J. Sonchack, H. Giesen, I. Pedisich, Z. Han, N. Shyamkumar, S. Burad, A. DeHon, B. T. Loo, Flightplan: Dataplane disaggregation and placement for p4 programs, in: USENIX Symposium on Networked Systems Design & Implementation (NSDI), 2021, pp. 571–592.

[121] R. Shah, A. Shirke, A. Trehan, M. Vutukuru, P. Kulkarni, pcube: Primitives for Network Data Plane Programming, in: IEEE International Conference on Network Protocols (ICNP), 2018, pp. 430–435.

[122] Z. Ma, J. Bi, C. Zhang, Y. Zhou, A. B. Dogar, CacheP4: A Behavior-level Caching Mechanism for P4, in: ACM SIGCOMM Conference Posters and Demos, 2017, p. 108–110.

[123] A. Abhashkumar, J. Lee, J. Tourrilhes, S. Banerjee, W. Wu, J.-M. Kang, A. Akella, P5: Policy-driven Optimization of P4 Pipeline, in: ACM Symposium on SDN Research (SOSR), 2017, p. 136–142.

[124] P. Wintermeyer, M. Apostolaki, A. Dietmüller, L. Vanbever, P2GO: P4 Profile-Guided Optimizations, in: ACM Workshop on Hot Topics in Networks (HotNets), 2020, p. 146–152.

[125] S. Yang, L. Baia, L. Cui, Z. Ming, Y. Wu, S. Yu, H. Shen, Y. Pan, P4 Edge node enabling stateful traffic engineering and cyber security, Journal of Network and Computer Applications (JNCA) 171 (2020) A84–A95.

[126] B. Vass, E. Bérczi-Kovács, C. Raiciu, G. Rétvári, Compiling Packet Programs to Reconfigurable Switches: Theory and Algorithms, in: P4 Workshop in Europe (EuroP4), 2020, p. 28–35.

[127] S. Abdi, U. Aftab, G. Bailey, B. Boughzala, F. Dewal, S. Parsazad, E. Tremblay, PFPSim: A Programmable Forwarding Plane Simulator, in: ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2016, pp. 55–60.

[128] J. Bai, J. Bi, P. Kuang, C. Fan, Y. Zhou, C. Zhang, NS4: Enabling Programmable Data Plane Simulation, in: ACM Symposium on SDN Research (SOSR), 2018, pp. 1–7.

[129] C. Fan, J. Bi, Y. Zhou, C. Zhang, H. Yu, NS4: A P4-Driven Network Simulator, in: ACM SIGCOMM Conference Posters and Demos, 2017, p. 105–107.

[130] N. McKeown, D. Talayco, G. Varghese, N. P. Lopes, N. Bjørner, A. Rybalchenko, Automatically Verifying Reachability and Well-Formedness in P4 Networks, `https://www.microsoft.com/en-us/research/wp-content/uploads/2016/09/p4nod.pdf`, accessed 01-20-2021 (2016).

[131] A. Kheradmand, G. Rosu, P4K: A Formal Semantics of P4 and Applications, ArXiv e-prints (2018).

[132] J. Liu, W. Hallahan, C. Schlesinger, M. Sharif, J. Lee, R. Soulé, H. Wang, C. Caşcaval, N. McKeown, N. Foster, P4V: Practical Verification for Programmable Data Planes, in: ACM SIGCOMM Conference, 2018, p. 490–503.

[133] L. Freire, M. Neves, L. Leal, K. Levchenko, A. Schaeffer-Filho, M. Barcellos, Uncovering Bugs in P4 Programs with Assertion-based Verification, in: ACM Symposium on SDN Research (SOSR), 2018, p. 1–7.

[134] M. Neves, L. Freire, A. Schaeffer-Filho, M. Barcellos, Verification of P4 Programs in Feasible Time using Assertions, in: ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT), 2018, p. 73–85.

[135] R. Stoenescu, D. Dumitrescu, M. Popovici, L. Negreanu, C. Raiciu, Debugging P4 Programs with Vera, in: ACM SIGCOMM Conference, 2018, p. 518–532.

[136] M. A. Noureddine, A. Hsu, M. Caesar, F. A. Zaraket, W. H. Sanders, P4AIG: Circuit-Level Verification of P4 Programs, in: IEEE/IFIP International Conference on Dependable Systems and Networks – Supplemental Volume (DSN-S), 2019, pp. 21–22.

[137] D. Dumitrescu, R. Stoenescu, L. Negreanu, C. Raiciu, bf4: towards bug-free P4 programs, in: ACM SIGCOMM Conference, 2020, p. 571–585.

[138] D. Dumitrescu, R. Stoenescu, M. Popovici, L. Negreanu, C. Raiciu, Dataplane equivalence and its applications, in: USENIX Symposium on Networked Systems Design & Implementation (NSDI), 2019, pp. 683–698.

[139] F. Yousefi, A. Abhashkumar, K. Subramanian, K. Hans, S. Ghorbani, A. Akella, Liveness Verification of Stateful Network Functions, in: USENIX Symposium on Networked Systems Design & Implementation (NSDI), 2020, pp. 257–272.

[140] A. Nötzli, J. Khan, A. Fingerhut, C. Barrett, P. Athanas, P4Pktgen: Automated Test Case Generation for P4 Programs, in: ACM Symposium on SDN Research (SOSR), 2018, pp. 1–7.

[141] Y. Zhou, J. Bi, Y. Lin, Y. Wang, D. Zhang, Z. Xi, J. Cao, C. Sun, P4Tester: Efficient Runtime Rule Fault Detection for Programmable Data Planes, in: IEEE International Workshop on Quality of Service (IWQoS), 2019, pp. 1–10.

[142] GitHub: P4app, `https://github.com/p4lang/p4app`, accessed 01-20-2021 (2021).

[143] A. Shukla, K. N. Hudemann, A. Hecker, S. Schmid, Runtime Verification of P4 Switches with Reinforcement Learning, in: Workshop on Network Meets AI & ML, 2019, p. 1–7.

[144] D. Jindal, R. Joshi, B. Leong, P4TrafficTool: Automated Code Generation for P4 Traffic Generators and Analyzers, in: ACM Symposium on SDN Research (SOSR), 2019, p. 152–153.

[145] H. T. Dang, H. Wang, T. Jepsen, G. Brebner, C. Kim, J. Rexford, R. Soulé, H. Weatherspoon, Whippersnapper: A P4 Language Benchmark Suite, in: ACM Symposium on SDN Research (SOSR), 2017, p. 95–101.

[146] F. Rodriguez, P. G. K. Patra, L. Csikor, C. E. Rothenberg, P. Vörös, S. Laki, G. Pongrácz, BB-Gen: A Packet Crafter for P4 Target Evaluation, in: ACM SIGCOMM Conference Posters and Demos, 2018, p. 111–113.

[147] H. Harkous, M. Jarschel, M. He, R. Pries, W. Kellerer, P8: P4 with Predictable Packet Processing Performance, IEEE Transactions on Network and Service Management (TNSM) (2020) 1–1.

[148] S. Kodeswaran, M. T. Arashloo, P. Tammana, J. Rexford, Tracking P4 Program Execution in the Data Plane, in: ACM Symposium on SDN Research (SOSR), 2020, p. 117–122.

[149] K. Birnfeld, D. C. da Silva, W. Cordeiro, B. B. N. de França, P4 Switch Code Data Flow Analysis: Towards Stronger Verification of Forwarding Plane Software, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), 2020, pp. 1–8.

[150] M. Neves, B. Huffaker, K. Levchenko, M. Barcellos, Dynamic Property Enforcement in Programmable Data Planes, in: IFIP-TC6 Networking Conference (Networking), 2019, pp. 1–9.

[151] C. Zhang, J. Bi, Y. Zhou, J. Wu, B. Liu, Z. Li, A. B. Dogar, Y. Wang, P4DB: On-the-fly Debugging of the Programmable Data Plane, in: IEEE International Conference on Network Protocols (ICNP), 2017, pp. 1–10.

[152] Y. Zhou, J. Bi, C. Zhang, B. Liu, Z. Li, Y. Wang, M. Yu, P4DB: On-the-Fly Debugging for Programmable Data Planes, IEEE/ACM Transactions on Networking (ToN) 27 (2019) 1714–1727.

[153] M. Neves, K. Levchenko, M. Barcellos, Sandboxing Data Plane Programs for Fun and Profit, in: ACM SIGCOMM Conference Posters and Demos, 2017, p. 103–104.

[154] A. Shukla, S. Fathalli, T. Zinner, A. Hecker, S. Schmid, P4Consist: Toward Consistent P4 SDNs, IEEE Journal on Selected Areas in Communications (JSAC) 38 (2020) 1293–1307.

[155] Z. Xia, J. Bi, Y. Zhou, C. Zhang, KeySight: A Scalable Troubleshooting Platform Based on Network Telemetry, in: ACM Symposium on SDN Research (SOSR), 2018, pp. 1–2.

[156] F. Ruffy, T. Wang, A. Sivaraman, Gauntlet: Finding Bugs in Compilers for Programmable Packet Processing, in: USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2020, pp. 1–17.

[157] J. Krude, J. Hofmann, M. Eichholz, K. Wehrle, A. Koch, M. Mezini, Online Reprogrammable Multi Tenant Switches, in: ACM CoNEXT Workshop on Emerging In-Network Computing Paradigms, 2019, p. 1–8.

[158] D. Hancock, J. van der Merwe, HyPer4: Using P4 to Virtualize the Programmable Data Plane, in: ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT), 2016, p. 35–49.

[159] C. Zhang, J. Bi, Y. Zhou, A. B. Dogar, J. Wu, HyperV: A High Performance Hypervisor for Virtualization of the Programmable Data Plane, in: IEEE International Conference on Computer Communications and Networks (ICCCN), 2017, pp. 1–9.

[160] C. Zhang, J. Bi, Y. Zhou, A. B. Dogar, J. Wu, MPVisor: A Modular Programmable Data Plane Hypervisor, in: ACM Symposium on SDN Research (SOSR), 2017, p. 179–180.

[161] GitHub: HyperVDP, `https://github.com/HyperVDP`, accessed 01-20-2021 (2021).

[162] C. Zhang, J. Bi, Y. Zhou, J. Wu, HyperVDP: High-Performance Virtualization of the Programmable Data Plane, IEEE Journal on Selected Areas in Communications (JSAC) 37 (2019) 556–569.

[163] M. Saquetti, G. Bueno, W. Cordeiro, J. R. Azambuja, P4VBox: Enabling P4-Based Switch Virtualization, IEEE Communications Letters 24 (2020) 146–149.

[164] M. Saquetti, G. Bueno, W. Cordeiro, J. R. Azambuja, VirtP4: An Architecture for P4 Virtualization, in: IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2019, pp. 75–78.

[165] P. Zheng, T. Benson, C. Hu, P4Visor: Lightweight Virtualization and Composition Primitives for Building and Testing Modular Programs, in: ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT), 2018, p. 98–111.

[166] R. Parizotto, L. Castanheira, F. Bonetti, A. Santos, A. Schaeffer-Filho, PRIME: Programming In-Network Modular Extensions, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), 2020, pp. 1–9.

[167] E. O. Zaballa, D. Franco, M. S. Berger, M. Higuero, A Perspective on P4-Based Data and Control Plane Modularity for Network Automation, in: P4 Workshop in Europe (EuroP4), 2020, p. 59–61.

[168] R. Stoyanov, N. Zilberman, MTPSA: Multi-Tenant Programmable Switches, in: P4 Workshop in Europe (EuroP4), 2020, p. 43–48.

[169] GitHub: MTPSA, `https://github.com/mtpsa`, accessed 01-20-2021 (2021).

[170] S. Han, S. Jang, H. Choi, H. Lee, S. Pack, Virtualization in Programmable Data Plane: A Survey and Open Challenges, IEEE Open Journal of the Communications Society 1 (2020) 527–534.

[171] J. Santiago da Silva, T. Stimpfling, T. Luinaud, B. Fradj, B. Boughzala, One for All, All for One: A Heterogeneous Data Plane for Flexible P4 Processing, in: IEEE International Conference on Network Protocols (ICNP), 2018, pp. 440–441.

[172] C. Beckmann, R. Krishnamoorthy, H. Wang, A. Lam, C. Kim, Hurdles for a DRAM-based Match-Action Table, in: Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), 2020, pp. 13–16.

[173] A. Aghdai, Y. Xu, H. J. Chao, Design of a hybrid modular switch, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2017, pp. 1–6.

[174] S. Laki, D. Horpacsi, P. Voros, M. Tejfel, P. Hudoba, G. Pongracz, L. Molnar, The Price for Asynchronous Execution of Extern Functions in Programmable Software Data Planes, in: Workshop on Flexible Network Data Plane Processing (NETPROC@ICIN), 2020, pp. 23–28.

[175] D. Horpácsi, P. Vörös, M. Tejfel, S. Laki, G. Pongrácz, L. Molnár, Asynchronous Extern Functions in Programmable Software Data Planes, in: P4 Workshop in Europe (EuroP4), 2019, pp. 1–2.

[176] D. Scholz, A. Oeldemann, F. Geyer, S. Gallenmüller, H. Stubbe, T. Wild, A. Herkersdorf, G. Carle, Cryptographic Hashing in P4 Data Planes, in: P4 Workshop in Europe (EuroP4), 2019, pp. 1–6.

[177] J. S. da Silva, F.-R. Boyer, L.-O. Chiquette, J. P. Langlois, Extern Objects in P4: an ROHC Header Compression Scheme Case Study, in: IEEE Conference on Network Softwarization (NetSoft), 2018, pp. 517–522.

[178] N. Gray, A. Grigorjew, T. Hosssfeld, A. Shukla, T. Zinner, Highlighting the Gap Between Expected and Actual Behavior in P4-enabled Networks, in: IFIP/IEEE Symposium on Integrated Management (IM), 2019, pp. 731–732.

[179] M. V. Dumitru, D. Dumitrescu, C. Raiciu, Can We Exploit Buggy P4 Programs?, in: ACM Symposium on SDN Research (SOSR), 2020, p. 62–68.

[180] J. Mambretti, J. Chen, F. Yeh, S. Y. Yu, International P4 Networking Testbed, in: ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2019, pp. 1–2.

[181] B. Chung, C. Tseng, J. H. Chen, J. Mambretti, P4MT: Multi-Tenant Support Prototype for International P4 Testbed, in: ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2019, pp. 1–2.

[182] A national programmable infrastructure to experiment with next-generation networks, `https://www.2stic.nl/national-programmable-infrastructure.html`, accessed 01-20-2021 (2021).

[183] R. Sukapuram, G. Barua, PPCU: Proportional Per-packet Consistent Updates for SDNs using Data Plane Time Stamps, Computer Networks 155 (2019) 72–86.

[184] R. Sukapuram, G. Barua, ProFlow: Proportional Per-Bidirectional-Flow Consistent Updates, IEEE Transactions on Network and Service Management (TNSM) 16 (2019) 675–689.

[185] S. Liu, T. A. Benson, M. K. Reiter, Efficient and Safe Network Updates with Suffix Causal Consistency, in: European Conference on Computer Systems (EUROSYS), 2019, p. 1–15.

[186] T. D. Nguyen, M. Chiesa, M. Canini, Decentralized Consistent Network Updates in SDN with ez-Segway, ArXiv e-prints (2017).

[187] S. Geissler, S. Herrnleben, R. Bauer, A. Grigorjew, T. Zinner, M. Jarschel, The Power of Composition: Abstracting aMulti-Device SDN Data Path Through a Single API, IEEE Transactions on Network and Service Management (TNSM) (2019) 722–735.

[188] E. C. Molero, S. Vissicchio, L. Vanbever, Hardware-Accelerated Network Control Planes, in: ACM Workshop on Hot Topics in Networks (HotNets), 2018, p. 120–126.

[189] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, J. Rexford, Heavy-Hitter Detection Entirely in the Data Plane, in: ACM Symposium on SDN Research (SOSR), 2017, p. 164–176.

[190] GitHub: Hashpipe, `https://github.com/vibhaa/hashpipe`, accessed 01-20-2021 (2021).

[191] Y. Lin, C. Huang, S. Tsai, SDN Soft Computing Application for Detecting Heavy Hitters, IEEE Transactions on Industrial Informatics (ToII) 15 (2019) 5690–5699.

[192] D. A. Popescu, G. Antichi, A. W. Moore, Enabling Fast Hierarchical Heavy Hitter Detection using Programmable Data Planes, in: ACM Symposium on SDN Research (SOSR), 2017, p. 191–192.

[193] R. Harrison, Q. Cai, A. Gupta, J. Rexford, Network-Wide Heavy Hitter Detection with Commodity Switches, in: ACM Symposium on SDN Research (SOSR), 2018, pp. 1–7.

[194] J. Kučera, D. A. Popescu, H. Wang, A. Moore, J. Kořenek, G. Antichi, Enabling Event-Triggered Data Plane Monitoring, in: ACM Symposium on SDN Research (SOSR), 2020, p. 14–26.

[195] M. Silva, A. Jacobs, R. Pfitscher, L. Granville, IDEAFIX: Identifying Elephant Flows in P4-Based IXP Networks, in: IEEE Global Communications Conference (GLOBECOM), 2018, pp. 1–6.

[196] B. Turkovic, J. Oostenbrink, F. Kuipers, Detecting Heavy Hitters in the Data-plane, ArXiv e-prints (2019).

[197] D. Ding, M. Savi, G. Antichi, D. Siracusa, An Incrementally-Deployable P4-Enabled Architecture for Network-Wide Heavy-Hitter Detection, IEEE Transactions on Network and Service Management (TNSM) 17 (2020) 75–88.

[198] GitHub: Network-Wide Heavy-Hitter Detection Implementation in P4 Language, `https://github.com/DINGDAMU/Network-wide-heavy-hitter-detection`, accessed 01-20-2021 (2021).

[199] J. Sonchack, A. J. Aviv, E. Keller, J. M. Smith, Turboflow: Information Rich Flow Record Generation on Commodity Switches, in: European Conference on Computer Systems (EUROSYS), 2018, p. 1–16.

[200] GitHub: TurboFlow, `https://github.com/jsonch/TurboFlow`, accessed 01-20-2021 (2021).

[201] J. Sonchack, O. Michel, A. J. Aviv, E. Keller, J. M. Smith, Scaling Hardware Accelerated Network Monitoring to Concurrent and Dynamic Queries With *Flow, in: USENIX Annual Technical Conference (ATC), 2018, pp. 823–835.

[202] GitHub: StarFlow, `https://github.com/jsonch/starflow`, accessed 01-25-2021 (2021).

[203] J. Hill, M. Aloserij, P. Grosso, Tracking Network Flows with P4, in: IEEE/ACM Innovating the Network for Data-Intensive Science (INDIS), 2018, pp. 23–32.

[204] L. Castanheira, R. Parizotto, A. E. Schaeffer-Filho, FlowStalker: Comprehensive Traffic Flow Monitoring on the Data Plane using P4, in: IEEE International Conference on Communicaotions (ICC), 2019, pp. 1–6.

[205] R. Parizotto, L. Castanheira, R. H. Ribeiro, L. Zembruzki, A. S. Jacobs, L. Z. Granville, A. Schaeffer-Filho, ShadowFS: Speeding-up Data Plane Monitoring and Telemetry using P4, in: IEEE International Conference on Communicaotions (ICC), 2020, pp. 1–6.

[206] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. V. Ramos, A. Madeira, FlowLens: Enabling Efficient Flow Classification for ML-based Network Security Applications, in: Network and Distributed Systems Security Symposium (NDSS), 2021, pp. 1–18.

[207] GitHub: FlowLens, `https://github.com/dmbb/FlowLens`, accessed 04-14-2021 (2021).

[208] W. Wang, P. Tammana, A. Chen, T. S. E. Ng, Grasp the Root Causes in the Data Plane: Diagnosing Latency Problems with SpiderMon, in: ACM Symposium on SDN Research (SOSR), 2020, p. 55–61.

[209] X. Chen, S. Landau-Feibish, Y. Koral, J. Rexford, O. Rottenstreich, S. A. Monetti, T.-Y. Wang, Fine-Grained Queue Measurement in the Data Plane, in: ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT), 2019, p. 15–29.

[210] Z. Zhao, X. Shi, X. Yin, Z. Wang, Q. Li, HashFlow for Better Flow Record Collection, in: IEEE International Conference on Distributed Computing Systems (ICDCS), 2019, pp. 1416–1425.

[211] Q. Huang, P. P. C. Lee, Y. Bao, Sketchlearn: Relieving User Burdens in Approximate Measurement with Automated Statistical Inference, in: ACM SIGCOMM Conference, 2018, p. 576–590.

[212] GitHub: SketchLearn, `https://github.com/huangqundl/SketchLearn`, accessed 01-20-2021 (2021).

[213] L. Tang, Q. Huang, P. C. Lee, A Fast and Compact Invertible Sketch for Network-Wide Heavy Flow Detection, IEEE/ACM Transactions on Networking (ToN) 28 (2020) 2350–2363.

[214] GitHub: MV-Sketch, `https://github.com/Grace-TL/MV-Sketch`, accessed 01-20-2021 (2021).

[215] Z. Hang, M. Wen, Y. Shi, C. Zhang, Interleaved Sketch: Toward Consistent Network Telemetry for Commodity Programmable Switches, IEEE ACCESS 7 (2019) 146745–146758.

[216] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, V. Braverman, One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon, in: ACM SIGCOMM Conference, 2016, p. 101–114.

[217] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, S. Uhlig, Elastic Sketch: Adaptive and Fast Network-wide Measurements, in: ACM SIGCOMM Conference, 2018, p. 561–575.

[218] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, S. Uhlig, Adaptive Measurements Using One Elastic Sketch, IEEE/ACM Transactions on Networking (ToN) 27 (2019) 2236–2251.

[219] GitHub: ElasticSketch, https://github.com/BlockLiu/ElasticSketchCode, accessed 01-20-2021 (2021).

[220] F. Pereira, N. Neves, F. M. V. Ramos, Secure network monitoring using programmable data planes, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2017, pp. 286–291.

[221] R. F. T. Martins, F. L. Verdi, R. Villaça, L. F. U. Garcia, Using Probabilistic Data Structures for Monitoring of Multi-tenant P4-based Networks, in: IEEE Symposium on Computers and Communications (ISCC), 2018, pp. 204–207.

[222] Y.-K. Lai, K.-Y. Shih, P.-Y. Huang, H.-P. Lee, Y.-J. Lin, T.-L. Liu, J. H. Chen, Sketch-based Entropy Estimation for Network Traffic Analysis using Programmable Data Plane ASICs, in: ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2019, pp. 1–2.

[223] Z. Liu, S. Zhou, O. Rottenstreich, V. Braverman, J. Rexford, Memory-Efficient Performance Monitoring on Programmable Switches with Lean Algorithms, in: SIAM Symposium on Algorithmic Principles of Computer Systems (APOCS), 2020, pp. 31–44.

[224] L. Tang, Q. Huang, P. P. C. Lee, SpreadSketch: Toward Invertible and Network-Wide Detection of Superspreaders, in: IEEE International Conference on Computer Communications (INFOCOM), 2020, pp. 1608–1617.

[225] GitHub: SpreadSketch, http://adslab.cse.cuhk.edu.hk/software/spreadsketch/, accessed 01-20-2021 (2021).

[226] J. Vestin, A. Kassler, D. Bhamare, K. Grinnemo, J. Andersson, G. Pongracz, Programmable Event Detection for In-Band Network Telemetry, in: IEEE International Conference on Cloud Networking (IEEE CloudNet), 2019, pp. 1–6.

[227] S. Wang, Y. Chen, J. Li, H. Hu, J. Tsai, Y. Lin, A Bandwidth-Efficient INT System for Tracking the Rules Matched by the Packets of a Flow, in: IEEE Global Communications Conference (GLOBECOM), 2019, pp. 1–6.

[228] D. Bhamare, A. Kassler, J. Vestin, M. A. Khoshkholghi, J. Taheri, IntOpt: In-Band Network Telemetry Optimization for NFV Service Chain Monitoring, in: IEEE International Conference on Communicaotions (ICC), 2019, pp. 1–7.

[229] C. Jia, T. Pan, Z. Bian, X. Lin, E. Song, C. Xu, T. Huang, Y. Liu, Rapid Detection and Localization of Gray Failures in Data Centers via In-band Network Telemetry, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), 2020, pp. 1–9.

[230] GitHub: Gray Failures Detection and Localization, `https://github.com/graytower/INT_DETECT`, accessed 01-20-2021 (2021).

[231] B. Niu, J. Kong, S. Tang, Y. Li, Z. Zhu, Visualize Your IP-Over-Optical Network in Realtime: A P4-Based Flexible Multilayer In-Band Network Telemetry (ML-INT) System, IEEE ACCESS 7 (2019) 82413–82423.

[232] N. S. Kagami, R. I. T. da Costa Filho, L. P. Gaspary, CAPEST: Offloading Network Capacity and Available Bandwidth Estimation to Programmable Data Planes, IEEE Transactions on Network and Service Management (TNSM) 17 (2020) 175–189.

[233] GitHub: Capest, `https://github.com/nicolaskagami/capest`, accessed 01-20-2021 (2021).

[234] N. Choi, L. Jagadeesan, Y. Jin, N. N. Mohanasamy, M. R. Rahman, K. Sabnani, M. Thottan, Run-time Performance Monitoring, Verification, and Healing of End-to-End Services, in: IEEE Conference on Network Softwarization (NetSoft), 2019, pp. 30–35.

[235] A. Sgambelluri, F. Paolucci, A. Giorgetti, D. Scano, F. Cugini, Exploiting Telemetry in Multi-Layer Networks, in: International Conference on Transparent Optical Networks (ICTON), 2020, pp. 1–4.

[236] Y. Feng, S. Panda, S. G. Kulkarni, K. K. Ramakrishnan, N. Duffield, A SmartNIC-Accelerated Monitoring Platform for In-band Network Telemetry, in: IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), 2020, pp. 1–6.

[237] J. Marques, K. Levchenko, L. Gaspary, IntSight: Diagnosing SLO Violations with in-Band Network Telemetry, in: ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT), 2020, p. 421–434.

[238] GitHub: IntSight, `https://github.com/jonadmark/intsight-conext`, accessed 01-20-2021 (2021).

[239] D. Suh, S. Jang, S. Han, S. Pack, X. Wang, Flexible sampling-based in-band network telemetry in programmable data plane, ICT Express 6 (2020) 62–65.

[240] S. Narayana, A. Sivaraman, V. Nathan, P. Goyal, V. Arun, M. Alizadeh, V. Jeyakumar, C. Kim, Language-Directed Hardware Design for Network Performance Monitoring, in: ACM SIGCOMM Conference, 2017, p. 85–98.

[241] V. Nathan, S. Narayana, A. Sivaraman, P. Goyal, V. Arun, M. Alizadeh, V. Jeyakumar, C. Kim, Demonstration of the Marple System for Network Performance Monitoring, in: ACM SIGCOMM Conference Posters and Demos, 2017, p. 57–59.

[242] GitHub: Marple, `https://github.com/performance-queries/marple`, accessed 01-20-2021 (2021).

[243] P. Laffranchini, L. Rodrigues, M. Canini, B. Krishnamurthy, Measurements As First-class Artifacts, in: IEEE International Conference on Computer Communications (INFOCOM), 2019, pp. 415–423.

[244] GitHub: Mafia, `https://github.com/paololaff/mafia-sdn`, accessed 01-20-2021 (2021).

[245] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, W. Willinger, Sonata: Query-Driven Streaming Network Telemetry, in: ACM Symposium on SDN Research (SOSR), 2018, p. 357–371.

[246] GitHub: SONATA, `https://github.com/Sonata-Princeton/SONATA-DEV`, accessed 01-20-2021 (2021).

[247] R. Teixeira, R. Harrison, A. Gupta, J. Rexford, PacketScope: Monitoring the Packet Lifecycle Inside a Switch, in: ACM Symposium on SDN Research (SOSR), 2020, p. 76–82.

[248] Y. Gao, Y. Jing, W. Dong, UniROPE: Universal and Robust Packet Trajectory Tracing for Software-Defined Networks, IEEE/ACM Transactions on Networking (ToN) 26 (2018) 2515–2527.

[249] S. Knossen, J. Hill, P. Grosso, Hop Recording and Forwarding State Logging: Two Implementations for Path Tracking in P4, in: IEEE/ACM Innovating the Network for Data-Intensive Science (INDIS), 2019, pp. 36–47.

[250] A. Indra Basuki, D. Rosiyadi, I. Setiawan, Preserving Network Privacy on Fine-grain Path-tracking Using P4-based SDN, in: International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET), 2020, pp. 129–134.

[251] R. Joshi, T. Qu, M. C. Chan, B. Leong, B. T. Loo, BurstRadar: Practical Real-time Microburst Monitoring for Datacenter Networks, in: ACM SIGOPS Asia-Pacific Workshop on System (APSys), 2018, pp. 1–8.

[252] GitHub: BurstRadar, `https://github.com/harshgondaliya/burstradar`, accessed 01-20-2021 (2021).

[253] M. Ghasemi, T. Benson, J. Rexford, Dapper: Data Plane Performance Diagnosis of TCP, in: ACM Symposium on SDN Research (SOSR), 2017, p. 61–74.

[254] C.-H. He, B. Y. Chang, S. Chakraborty, C. Chen, L. C. Wang, A Zero Flow Entry Expiration Timeout P4 Switch, in: ACM Symposium on SDN Research (SOSR), 2018, pp. 1–2.

[255] A. Riesenberg, Y. Kirzon, M. Bunin, E. Galili, G. Navon, T. Mizrahi, Time-Multiplexed Parsing in Marking-Based Network Telemetry, in: ACM International Conference on Systems and Storage (SYSTOR), 2019, p. 80–85.

[256] GitHub: P4 Alternate Marking Algorithm, `https://github.com/AlternateMarkingP4/FlaseClase`, accessed 01-20-2021 (2021).

[257] S. Y. Wang, H. W. Hu, Y. B. Lin, Design and Implementation of TCP-Friendly Meters in P4 Switches, IEEE/ACM Transactions on Networking (ToN) 28 (2020) 1885–1898.

[258] R. Kundel, F. Siegmund, J. Blendin, A. Rizk, B. Koldehofe, P4STA: High Performance Packet Timestamping with Programmable Packet Processors, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), 2020, p. 1–9.

[259] GitHub: P4STA, `https://github.com/ralfkundel/P4STA`, accessed 01-20-2021 (2021).

[260] R. Hark, D. Bhat, M. Zink, R. Steinmetz, A. Rizk, Preprocessing Monitoring Information on the SDN Data-Plane using P4, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2019, pp. 1–6.

[261] D. Ding, M. Savi, D. Siracusa, Estimating Logarithmic and Exponential Functions to Track Network Traffic Entropy in P4, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), 2020, pp. 1–9.

[262] GitHub: P4Entropy, `https://github.com/DINGDAMU/P4Entropy`, accessed 01-20-2021 (2021).

[263] P. Taffet, J. Mellor-Crummey, Lightweight, Packet-Centric Monitoring of Network Traffic and Congestion Implemented in P4, in: IEEE Symposium on High-Performance Interconnects (HOTI), 2019, pp. 54–58.

[264] Y. Lin, Y. Zhou, Z. Liu, K. Liu, Y. Wang, M. Xu, J. Bi, Y. Liu, J. Wu, NetView: Towards On-Demand Network-Wide Telemetry in the Data Center, in: IEEE International Conference on Communicaotions (ICC), 2020, pp. 1–6.

[265] J. Bai, M. Zhang, G. Li, C. Liu, M. Xu, H. Hu, FastFE: Accelerating ML-Based Traffic Analysis with Programmable Switches, in: Workshop on Secure Programmable Network Infrastructure (SPIN), 2020, p. 1–7.

[266] J. Kučera, R. B. Basat, M. Kuka, G. Antichi, M. Yu, M. Mitzenmacher, Detecting Routing Loops in the Data Plane, in: ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT), 2020, p. 466–473.

[267] Z. Hang, Y. Shi, M. Wen, C. Zhang, TBSW: Time-Based Sliding Window Algorithm for Network Traffic Measurement, in: IEEE International Conference on High Performance Computing and Communications; IEEE International Conference on Smart City; IEEE International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2019, pp. 1305–1310.

[268] B. Guan, S. Shen, FlowSpy: An Efficient Network Monitoring Framework Using P4 in Software-Defined Networks, in: IEEE Semiannual Vehicular Technology Conference (VTC), 2019, pp. 1–5.

[269] Heavy Hitter Detection: Guest lecture for CS344 at Stanford, `https://cs344-stanford.github.io/lectures/Lecture-4-HHD.pdf`, accessed 01-20-2021 (2018).

[270] B. Claise, Cisco Systems NetFlow Services Export Version 9, RFC 3954, RFC Editor (10 2004).
URL `http://www.rfc-editor.org/rfc/rfc3954.txt`

[271] P. Phaal, S. Panchen, N. McKee, InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks, RFC 3176, RFC Editor (09 2001).
URL `http://www.rfc-editor.org/rfc/rfc3176.txt`

[272] B. Claise, B. Trammell, P. Aitken, Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information, STD 77, RFC Editor (09 2013).
URL `http://www.rfc-editor.org/rfc/rfc7011.txt`

[273] In-band Network Telemetry (INT), `https://p4.org/assets/INT-current-spec.pdf`, accessed 01-20-2021 (2021).

[274] Charter of the P4 Applications WG, `https://github.com/p4lang/p4-applications/blob/master/docs/charter.pdf`, accessed 01-20-2021 (2021).

[275] C. Kim, A. Sivaraman, N. P. Katta, A. Bas, A. Dixit, L. J. Wobker, In-band Network Telemetry via Programmable Dataplanes, `https://nkatta.github.io/papers/int-demo.pdf` (2015).

[276] F. Cugini, P. Gunning, F. Paolucci, P. Castoldi, A. Lord, P4 In-Band Telemetry (INT) for Latency-Aware VNF in Metro Networks, in: Optical Fiber Communication Conference (OFC), 2019, pp. 1–3.

[277] Open Networking Foundation: Trellis, `https://www.opennetworking.org/trellis/`, accessed 01-20-2021 (2021).

[278] Google Presentations: Trellis & P4 Tutorial, `http://bit.ly/trellis-p4-slides`, accessed 01-20-2021 (2018).

[279] GitHub: ONF Trellis, https://github.com/opennetworkinglab/routing/tree/master/trellis, accessed 01-20-2021 (2021).

[280] A. Sivaraman, C. Kim, R. Krishnamoorthy, A. Dixit, M. Budiu, DC.P4: Programming the Forwarding Plane of a Data-center Switch, in: ACM SIGCOMM Conference, 2015, p. 1–8.

[281] GitHub: DC.p4, https://github.com/p4lang/papers/tree/master/sosr15, accessed 01-20-2021 (2021).

[282] Open Network Foundation: P4 apps at ONF, https://github.com/p4lang/p4-applications/blob/master/meeting_slides/2018_04_19_ONF.pdf, accessed 01-20-2021 (2018).

[283] GitHub: fabric.p4, https://github.com/opennetworkinglab/onos/blob/master/pipelines/fabric/impl/src/main/resources/fabric.p4, accessed 01-20-2021 (2021).

[284] RARE (Router for Academia, Research & Education), https://wiki.geant.org/display/RARE/Home, accessed 04-16-2021 (2021).

[285] GitHub: RARE, https://github.com/frederic-loui/RARE, accessed 04-16-2021 (2021).

[286] B. Pit-Claudel, Y. Desmouceaux, P. Pfister, M. Townsley, T. Clausen, Stateless Load-Aware Load Balancing in P4, in: IEEE International Conference on Network Protocols (ICNP), 2018, pp. 418–423.

[287] R. Miao, H. Zeng, C. Kim, J. Lee, M. Yu, SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap using Switching ASICs, in: ACM SIGCOMM Conference, 2017, p. 15–28.

[288] N. Katta, M. Hira, C. Kim, A. Sivaraman, J. Rexford, HULA: Scalable Load Balancing using Programmable Data Planes, in: ACM Symposium on SDN Research (SOSR), 2016, p. 1–12.

[289] C. H. Benet, A. J. Kassler, T. Benson, G. Pongracz, MP-HULA: Multipath Transport Aware Load Balancing using Programmable Data Planes, in: Morning Workshop on In-Network Computing, 2018, p. 7–13.

[290] B. T. Chiang, K. Wang, Cost-effective Congestion-aware Load Balancing for Datacenters, in: International Conference on Electronics, Information, and Communication (ICEIC), 2019, pp. 1–6.

[291] J.-L. Ye, C. Chen, Y. H. Chu, A Weighted ECMP Load Balancing Scheme for Data Centers using P4 Switches, in: IEEE International Conference on Cloud Networking (IEEE CloudNet), 2018, pp. 1–4.

[292] K.-F. Hsu, P. Tammana, R. Beckett, A. Chen, J. Rexford, D. Walker, Adaptive Weighted Traffic Splitting in Programmable Data Planes, in: ACM Symposium on SDN Research (SOSR), 2020, p. 103–109.

[293] M. Pizzutti, A. Schaeffer-Filho, An Efficient Multipath Mechanism Based on the Flowlet Abstraction and P4, in: IEEE Global Communications Conference (GLOBECOM), 2018, pp. 1–6.

[294] M. Pizzutti, A. Schaeffer-Filho, Adaptive Multipath Routing based on Hybrid Data and Control Plane Operation, in: IEEE International Conference on Computer Communications (INFOCOM), 2020, pp. 730–738.

[295] J. Zhang, S. Wen, J. Zhang, H. Chai, T. Pan, T. Huang, L. Zhang, Y. Liu, F. R. Yu, Fast Switch-Based Load Balancer Considering Application Server States, IEEE/ACM Transactions on Networking (ToN) 28 (2020) 1391–1404.

[296] Q. Li, J. Zhang, T. Pan, T. Huang, Y. Liu, Data-driven Routing Optimization based on Programmable Data Plane, in: IEEE International Conference on Computer Communications and Networks (ICCCN), 2020, pp. 1–9.

[297] E. Kawaguchi, H. Kasuga, N. Shinomiya, Unsplittable flow Edge Load factor Balancing in SDN using P4 Runtime, in: International Telecommunication Networks and Applications Conference (ITNAC), 2019, pp. 1–6.

[298] E. Cidon, S. Choi, S. Katti, N. McKeown, AppSwitch: Application-layer Load Balancing withina Software Switch, in: Asia-Pacific Workshop on Networking (APnet), 2017, p. 64–70.

[299] V. Olteanu, A. Agache, A. Voinescu, C. Raiciu, Stateless Datacenter Load-balancing with Beamer, in: USENIX Symposium on Networked Systems Design & Implementation (NSDI), 2018, pp. 125–139.

[300] GitHub: Beamer, `https://github.com/Beamer-LB`, accessed 01-25-2021 (2021).

[301] J. Geng, J. Yan, Y. Zhang, P4QCN: Congestion Control using P4-Capable Device in Data Center Networks, Electronics Journal 8 (2019) 280.

[302] J. Jiang, Y. Zhang, An Accurate Congestion Control Mechanism in Programmable Network, in: IEEE Annual Computing and Communication Workshop and Conference (CCWC), 2019, pp. 673–677.

[303] S. Shahzad, E. Jung, J. Chung, R. Kettimuthu, Enhanced Explicit Congestion Notification (EECN) in TCP with P4 Programming, in: International Conference on Green and Human Information Technology (ICGHIT), 2020, pp. 35–40.

[304] C. Chen, H. Fang, M. S. Iqbal, QoSTCP: Provide Consistent Rate Guarantees to TCP flows in Software Defined Networks, in: IEEE International Conference on Communicaotions (ICC), 2020, pp. 1–6.

[305] A. Laraba, J. François, I. Chrisment, S. R. Chowdhury, R. Boutaba, Defeating Protocol Abuse with P4: Application to Explicit Congestion Notification, in: IFIP-TC6 Networking Conference (Networking), 2020, pp. 431–439.

[306] N. K. Sharma, M. Liu, K. Atreya, A. Krishnamurthy, Approximating Fair Queueing on Reconfigurable Switches, in: USENIX Symposium on Networked Systems Design & Implementation (NSDI), 2018, p. 1–16.

[307] C. Cascone, N. Bonelli, L. Bianchi, A. Capone, B. Sansò, Towards Approximate Fair Bandwidth Sharing via Dynamic Priority Queuing, in: IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), 2017, pp. 1–6.

[308] D. Bhat, J. Anderson, P. Ruth, M. Zink, K. Keahey, Application-based QoE support with P4 and OpenFlow, in: IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2019, pp. 817–823.

[309] E. F. Kfoury, J. Crichigno, E. Bou-Harb, D. Khoury, G. Srivastava, Enabling TCP Pacing using Programmable Data Plane Switches, in: International Conference on Telecommunications and Signal Processing (TSP), 2019, pp. 273–277.

[310] Y. Chen, L. Yen, W. Wang, C. Chuang, Y. Liu, C. Tseng, P4-Enabled Bandwidth Management, in: Asia-Pacific Network Operations and Management Symposium (APNOMS), 2019, pp. 1–5.

[311] S. S. W. Lee, K. Chan, A Traffic Meter Based on a Multicolor Marker for Bandwidth Guarantee and Priority Differentiation in SDN Virtual Networks, IEEE Transactions on Network and Service Management (TNSM) 16 (2019) 1046–1058.

[312] S.-Y. Wang, J.-Y. Li, Y.-B. Lin, Aggregating and disaggregating packets with various sizes of payload in P4 switches at 100 Gbps line rate, Journal of Network and Computer Applications (JNCA) 165 (2020) 102676.

[313] K. Tokmakov, M. Sarker, J. Domaschka, S. Wesner, A Case for Data Centre Traffic Management on Software Programmable Ethernet Switches, in: IEEE International Conference on Cloud Networking (IEEE CloudNet), 2019, pp. 1–6.

[314] B. Turkovic, F. Kuipers, N. van Adrichem, K. Langendoen, Fast Network Congestion Detection and Avoidance using P4, in: Workshop on Networking for Emerging Applications and Technologies (NEAT), 2018, p. 45–51.

[315] R. Kundel, J. Blendin, T. Viernickel, B. Koldehofe, R. Steinmetz, P4-CoDel: Active Queue Management in Programmable Data Planes, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2018, pp. 1–4.

[316] GitHub: P4-CoDel, `https://github.com/ralfkundel/p4-codel`, accessed 01-20-2021 (2021).

[317] M. Menth, H. Mostafaei, D. Merling, M. Häberle, Implementation and Evaluation of Activity-Based Congestion Management using P4 (P4-ABC), MDPI Future Internet Journal (FI) 11 (2019) 159.

[318] B. Turkovic, F. Kuipers, P4air: Increasing Fairness among Competing Congestion Control Algorithms, in: IEEE International Conference on Network Protocols (ICNP), 2020, pp. 1–12.

[319] L. B. Fernandes, L. Camargos, Bandwidth throttling in a P4 switch, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2020, pp. 91–94.

[320] G. Wang, C. Chen, C. Chen, L. Pan, Y. Wang, C. Fan, C. Hsu, Streaming Scalable Video Sequences with Media-Aware Network Elements Implemented in P4 Programming Language, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), 2018, pp. 1–2.

[321] A. G. Alcoz, A. Dietmüller, L. Vanbever, SP-PIFO: Approximating Push-In First-Out Behaviors using Strict-Priority Queues, in: USENIX Symposium on Networked Systems Design & Implementation (NSDI), 2020, pp. 59–76.

[322] I. Kunze, M. Gunz, D. Saam, K. Wehrle, J. Rüth, Tofino + P4: A Strong Compound for AQM on High-Speed Networks?, in: IFIP/IEEE International Symposium on Integrated Network Management, 2021, pp. 72–80.

[323] GitHub: PIE for Tofino, `https://github.com/COMSYS/pie-for-tofino`, accessed 04-15-2021 (2021).

[324] H. Harkous, C. Papagianni, K. De Schepper, M. Jarschel, M. Dimolianis, R. Preis, Virtual queues for p4: A poor man's programmable traffic manager, IEEE Transactions on Network and Service Management (TNSM) (2021) 1–1.

[325] B. Andrus, S. A. Sasu, T. Szyrkowiec, A. Autenrieth, M. Chamania, J. K. Fischer, S. Rasp, Zero-Touch Provisioning of Distributed Video Analytics in a Software-Defined Metro-Haul Network with P4 Processing, in: Optical Fiber Communication Conference (OFC), 2019, pp. 1–3.

[326] S. Ibanez, G. Antichi, G. Brebner, N. McKeown, Event-Driven Packet Processing, in: ACM Workshop on Hot Topics in Networks (HotNets), 2019, p. 133–140.

[327] E. F. Kfoury, J. Crichigno, E. Bou-Harb, Offloading Media Traffic to Programmable Data Plane Switches, in: IEEE International Conference on Communicaotions (ICC), 2020, pp. 1–7.

[328] I. Kettaneh, S. Udayashankar, A. Abdel-hadi, R. Grosman, S. Al-Kiswany, Falcon: Low Latency, Network-Accelerated Scheduling, in: P4 Workshop in Europe (EuroP4), 2020, p. 7–12.

[329] T. Osiński, M. Kossakowski, M. Pawlik, J. Palimąka, M. Sala, H. Tarasiuk, Unleashing the Performance of Virtual BNG by Offloading Data Plane to a Programmable ASIC, in: P4 Workshop in Europe (EuroP4), 2020, p. 54–55.

[330] J. Lee, R. Miao, C. Kim, M. Yu, H. Zeng, Stateful Layer-4 Load Balancing in Switching ASICs, in: ACM SIGCOMM Conference Posters and Demos, 2017, p. 133–135.

[331] K. Nichols, V. Jacobson, A. McGregor, J. Iyengar, Controlled Delay Active Queue Management, RFC 8289, RFC Editor (01 2018).
URL `https://tools.ietf.org/rfc/rfc8289.txt`

[332] B. Lewis, L. Fawcett, M. Broadbent, N. Race, Using P4 to Enable Scalable Intents in Software Defined Networks, in: IEEE International Conference on Network Protocols (ICNP), 2018, pp. 442–443.

[333] GitHub: P4 Source Routing, `https://github.com/BenRLewis/P4-Source-Routing`, accessed 01-20-2021 (2021).

[334] L. Luo, H. Yu, S. Luo, Z. Ye, X. Du, M. Guizani, Scalable Explicit Path Control in Software-Defined Networks, Journal of Network and Computer Applications (JNCA) 141 (2019) 86–103.

[335] GitHub: P4 Paco, `https://github.com/an15m/paco`, accessed 01-20-2021 (2021).

[336] A. Kushwaha, S. Sharma, N. Bazard, A. Gumaste, B. Mukherjee, Design, Analysis, and a Terabit Implementation of a Source-Routing-Based SDN Data Plane, IEEE Systems Journal (2020).

[337] A. Abdelsalam, A. Tulumello, M. Bonola, S. Salsano, C. Filsfils, Pushing Network Programmability to the limits with SRv6 uSIDs and P4, in: P4 Workshop in Europe (EuroP4), 2020, p. 62–64.

[338] W. Braun, J. Hartmann, M. Menth, Demo: Scalable and Reliable Software-Defined Multicast with BIER and P4, in: IFIP/IEEE Symposium on Integrated Management (IM), 2017, pp. 905–906.

[339] Bitbucket: p4-bfr), `https://bitbucket.org/wb-ut/p4-bfr`, accessed 01-20-2021 (2021).

[340] D. Merling, S. Lindner, M. Menth, P4-Based Implementation of BIER and BIER-FRR for Scalable and Resilient Multicast, Journal of Network and Computer Applications (JNCA) 169 (2020) 102764.

[341] D. Merling, S. Lindner, M. Menth, Hardware-based evaluation of scalable and resilient multicast with bier in p4, IEEE ACCESS 9 (2021) 34500–34514.

[342] GitHub: P4-BIER, `https://github.com/uni-tue-kn/p4-bier`, accessed 01-20-2021 (2021).

[343] GitHub: P4-BIER for Tofino, `https://github.com/uni-tue-kn/p4-bier-tofino`, accessed 04-26-2021 (2021).

[344] M. Shahbaz, L. Suresh, J. Rexford, N. Feamster, O. Rottenstreich, M. Hira, Elmo: Source Routed Multicast for Public Clouds, in: ACM Special Interest Group on Data Communication, 2019, pp. 2587–2600.

[345] GitHub: Elmo MCast, `https://github.com/Elmo-MCast/p4-programs`, accessed 01-20-2021 (2021).

[346] S. Luo, H. Yu, K. Li, H. Xing, Efficient File Dissemination in Data Center Networks with Priority-based Adaptive Multicast, IEEE Journal on Selected Areas in Communications (JSAC) 38 (2020) 1161–1175.

[347] C. Wernecke, H. Parzyjegla, G. Mühl, P. Danielis, D. Timmermann, Realizing Content-Based Publish/Subscribe with P4, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2018, pp. 1–7.

[348] C. Wernecke, H. Parzyjegla, G. Mühl, E. Schweissguth, D. Timmermann, Flexible Notification Forwarding for Content-Based Publish/Subscribe Using P4, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2020, pp. 1–5.

[349] C. Wernecke, H. Parzyjegla, G. Mühl, Implementing Content-based Publish/Subscribe on the Network Layer with P4, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2020, pp. 144–149.

[350] C. Wernecke, H. Parzyjegla, G. Mühl, P. Danielis, E. Schweissguth, D. Timmermann, Stitching Notification Distribution Trees for Content-based Publish/Subscribe with P4, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2020, pp. 100–104.

[351] T. Jepsen, M. Moshref, A. Carzaniga, N. Foster, R. Soulé, Packet Subscriptions for Programmable ASICs, in: ACM Workshop on Hot Topics in Networks (HotNets), 2018, p. 176–183.

[352] R. Kundel, C. Gaertner, M. Luthra, S. Bhowmik, B. Koldehofe, Flexible Content-based Publish/Subscribe over Programmable Data Planes, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), 2020, pp. 1–5.

[353] GitHub: p4bsub, `https://github.com/ralfkundel/p4bsub/`, accessed 01-20-2021 (2021).

[354] J. Vestin, A. Kassler, S. Laki, G. Pongrácz, Towards In-Network Event Detection and Filtering for Publish/Subscribe Communication using Programmable Data Planes, IEEE Transactions on Network and Service Management (TNSM) (2020) 415–428.

[355] S. Signorello, R. State, J. François, O. Festor, NDN.p4: Programming Information-Centric Data-Planes, in: IEEE Conference on Network Softwarization (NetSoft), 2016, pp. 384–389.

[356] R. Miguel, S. Signorello, F. M. V. Ramos, Named Data Networking with Programmable Switches, in: IEEE International Conference on Network Protocols (ICNP), 2018, pp. 400–405.

[357] GitHub: NDN.p4, `https://github.com/signorello/NDN.p4`, accessed 01-20-2021 (2021).

[358] GitHub: NDN.p4-16, `https://github.com/netx-ulx/NDN.p4-16`, accessed 01-20-2021 (2021).

[359] O. Karrakchou, N. Samaan, A. Karmouch, ENDN: An Enhanced NDN Architecture with a P4-programmable Data Plane, in: International Conference on Networking (ICN), 2020, p. 1–11.

[360] R. Sedar, M. Borokhovich, M. Chiesa, G. Antichi, S. Schmid, Supporting Emerging Applications With Low-Latency Failover in P4, in: Workshop on Networking for Emerging Applications and Technologies (NEAT), 2018, p. 52–57.

[361] GitHub: P4-FRR, `https://bitbucket.org/roshanms/p4-frr/src/master/`, accessed 01-20-2021 (2021).

[362] H. Giesen, L. Shi, J. Sonchack, A. Chelluri, N. Prabhu, N. Sultana, L. Kant, A. J. McAuley, A. Poylisher, A. DeHon, B. T. Loo, In-Network Computing to the Rescue of Faulty Links, in: Morning Workshop on In-Network Computing, 2018, pp. 1–6.

[363] T. Qu, R. Joshi, M. Chan, B. Leong, D. Guo, Z. Liu, SQR: In-network Packet Loss Recovery from Link Failures for Highly Reliable Datacenter Networks, in: IEEE International Conference on Network Protocols (ICNP), 2019, pp. 1–12.

[364] GitHub: P4 SQR, `https://git.io/fjbnV`, accessed 01-20-2021 (2021).

[365] S. Lindner, D. Merling, M. Häberle, M. Menth, P4-Protect: 1+1 Path Protection for P4, in: P4 Workshop in Europe (EuroP4), 2020, p. 21–27.

[366] GitHub: P4-Protect BMv2, `https://github.com/uni-tue-kn/p4-protect`, accessed 01-20-2021 (2021).

[367] GitHub: P4-Protect Tofino, `https://github.com/uni-tue-kn/p4-protect-tofino`, accessed 01-20-2021 (2021).

[368] K. Hirata, , T. Tachibana, Implementation of Multiple Routing Configurations on Software-Defined Networks with P4, in: Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), 2019, pp. 13–16.

[369] S. Lindner, M. Häberle, F. Heimgaertner, N. Nayak, S. Schildt, D. Grewe, H.Loehr, M. Ment, P4 In-Network Source Protection for Sensor Failover, in: IFIP-TC6 Networking Conference (Networking), 2020, pp. 791–796.

[370] GitHub: P4 Source Protection BMv2, `https://github.com/uni-tue-kn/p4-source-protection`, accessed 01-20-2021 (2021).

[371] GitHub: P4 Source Protection Tofino, `https://github.com/uni-tue-kn/p4-source-protection-tofino`, accessed 01-20-2021 (2021).

[372] K. Subramanian, A. Abhashkumar, L. D'Antoni, A. Akella, D2R: Dataplane-Only Policy-Compliant Routing Under Failures (2019).

[373] M. Chiesa, R. Sedar, G. Antichi, M. Borokhovich, A. Kamisiński, G. Nikolaidis, S. Schmid, PURR: A Primitive for Reconfigurable Fast Reroute, in: ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT), 2019, p. 1–14.

[374] T. Holterbach, E. C. Molero, M. Apostolaki, A. Dainotti, S. Vissicchio, L. Vanbever, Blink: Fast Connectivity Recovery Entirely in the Data Plane, in: USENIX Symposium on Networked Systems Design & Implementation (NSDI), 2019, pp. 161–176.

[375] GitHub: Blink, `https://github.com/nsg-ethz/Blink`, accessed 01-20-2021 (2021).

[376] K.-F. Hsu, R. Beckett, A. Chen, J. Rexford, D. Walker, Contra: A Programmable System for Performance-aware Routing, in: USENIX Symposium on Networked Systems Design & Implementation (NSDI), 2020, pp. 701–721.

[377] O. Michel, E. Keller, Policy Routing using Process-Level Identifiers, in: IEEE International Conference on Cloud Engineering Workshop (IC2EW), 2016, pp. 7–12.

[378] A. C. Baktir, A. Ozgovde, C. Ersoy, Implementing Service-Centric Model with P4: A Fully-Programmable Approach, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), 2018, pp. 1–6.

[379] W. Froes, L. Santos, L. N. Sampaio, M. Martinello, A. Liberato, R. S. Villaca, ProgLab: Programmable Labels for QoS Provisioning on Software Defined Networks, Computer Communications 161 (2020) 99–108.

[380] N. VARYANI, Z.-L. ZHANG, D. DAI, QROUTE: An Efficient Quality of Service (QoS) Routing Scheme for Software-Defined Overlay Networks, IEEE ACCESS 8 (2020) 104109–104126.

[381] S. Gimenez, E. Grasa, S. Bunch, A Proof of Concept Implementation of a RINA Interior Router using P4-enabled Software Targets, in: Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), 2020, pp. 57–62.

[382] W. Feng, X. Tan, Y. Jin, Implementing ICN over P4 in HTTP Scenario, in: IEEE International Conference on Hot Information-Centric Networking (HotICN), 2019, pp. 37–43.

[383] G. Grigoryan, Y. Liu, M. Kwon, PFCA: A Programmable FIB Caching Architecture, IEEE/ACM Transactions on Networking (ToN) 28 (2020) 1872–1884.

[384] A. McAuley, Y. M. Gottlieb, L. Kant, J. Lee, A. Poylisher, P4-Based Hybrid Error Control Booster Providing New Design Tradeoffs in Wireless Networks, in: IEEE Military Communications Conference (MILCOM), 2019, pp. 731–736.

[385] M. Kogias, G. Prekas, A. Ghosn, J. Fietz, E. Bugnion, R2P2: Making RPCs first-class datacenter citizens, in: USENIX Annual Technical Conference (ATC), 2019, pp. 863–880.

[386] GitHub: R2P2 - Request Response Pair Protocol, `https://github.com/epfl-dcsl/r2p2`, accessed 01-25-2021 (2021).

[387] D. Merling, M. Menth, N. Warnke, T. Eckert, An Overview of Bit Index Explicit Replication (BIER), IETF Journal (2018).

[388] M. Hollingsworth, J. Lee, Z. Liu, J. Lee, S. Ha, D. Grunwald, P4EC: Enabling Terabit Edge Computing in Enterprise 4G LTE, in: USENIX Workshop on Hot Topics in Edge Computing (HotEdge), 2020, pp. 1–7.

[389] GitHub: spgw.p4, `https://github.com/opennetworkinglab/onos/blob/master/pipelines/fabric/impl/src/main/resources/include/control/spgw.p4`, accessed 01-20-2021 (2021).

[390] P. Palagummi, K. M. Sivalingam, SMARTHO: A Network Initiated Handover in NG-RAN using P4-based Switches, in: International Conference on Network and Services Management (CNSM), 2018, pp. 338–342.

[391] A. Aghdai, M. Huang, D. Dai, Y. Xu, J. Chao, Transparent Edge Gateway for Mobile Networks, in: IEEE International Conference on Network Protocols (ICNP), 2018, pp. 412–417.

[392] A. Aghdai, Y. Xu, M. Huang, D. H. Dai, H. J. Chao, Enabling Mobility in LTE-Compatible Mobile-edge Computing with Programmable Switches, ArXiv e-prints (2019).

[393] J. Xie, C. Qian, D. Guo, X. Li, S. Shi, H. Chen, Efficient Data Placement and Retrieval Services in Edge Computing, in: IEEE International Conference on Distributed Computing Systems (ICDCS), 2019, pp. 1029–1039.

[394] J. Xie, D. Guo, X. Shi, H. Cai, C. Qian, H. Chen, A Fast Hybrid Data Sharing Framework for Hierarchical Mobile Edge Computing, in: IEEE International Conference on Computer Communications (INFOCOM), 2020, pp. 2609–2618.

[395] C. Shen, D. Lee, C. Ku, M. Lin, K. Lu, S. Tan, A Programmable and FPGA-accelerated GTP Offloading Engine for Mobile Edge Computing in 5G Networks, in: IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2019, pp. 1021–1022.

[396] C. Lee, K. Ebisawa, H. Kuwata, M. Kohno, S. Matsushima, Performance Evaluation of GTP-U and SRv6 Stateless Translation, in: International Conference on Network and Services Management (CNSM), 2019, pp. 1–6.

[397] R. Ricart-Sanchez, P. Malagon, J. M. Alcaraz-Calero, Q. Wang, P4-NetFPGA-based network slicing solution for 5G MEC architectures, in: ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2019, pp. 1–2.

[398] S. K. Singh, C. E. Rothenberg, G. Patra, G. Pongracz, Offloading Virtual Evolved Packet Gateway User Plane Functions to a Programmable ASIC, in: ACM CoNEXT Workshop on Emerging In-Network Computing Paradigms, 2019, p. 9–14.

[399] R. Shah, V. Kumar, M. Vutukuru, P. Kulkarni, TurboEPC: Leveraging Dataplane Programmability to Accelerate the Mobile Packet Core, in: ACM Symposium on SDN Research (SOSR), 2020, p. 83–95.

[400] P. Vörös, G. Pongrácz, S. Laki, Towards a Hybrid Next Generation NodeB, in: P4 Workshop in Europe (EuroP4), 2020, p. 56–58.

[401] Y. Lin, T. Huang, S. Tsai, Enhancing 5G/IoT Transport Security Through Content Permutation, IEEE ACCESS 7 (2019) 94293–94299.

[402] M. Uddin, S. Mukherjee, H. Chang, T. V. Lakshman, SDN-Based Service Automation for IoT, in: IEEE International Conference on Network Protocols (ICNP), 2017, pp. 1–10.

[403] M. Uddin, S. Mukherjee, H. Chang, T. V. Lakshman, SDN-Based Multi-Protocol Edge Switching for IoT Service Automation, IEEE Journal on Selected Areas in Communications (JSAC) 36 (2018) 2775–2786.

[404] S.-Y. Wang, C.-M. Wu, Y.-B. Linm, C.-C. Huang, High-Speed Data-Plane Packet Aggregation and Disaggregation by P4 Switches, Journal of Network and Computer Applications (JNCA) 142 (2019) 98–110.

[405] A. L. R. Madureira, F. R. C. Araújo, L. N. Sampaio, On supporting IoT data aggregation through programmable data planes, Computer Networks 177 (2020) 107330.

[406] P. Engelhard, A. Zachlod, J. Schulz-Zander, S. Du, Toward scalable and virtualized massive wireless sensor networks, in: International Conference on Networked Systems (NetSys), 2019, pp. 1–6.

[407] J. Vestin, A. Kassler, J. Åkerberg, FastReact: In-Network Control and Caching for Industrial Control Networks using Programmable Data Planes, in: IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2018, pp. 219–226.

[408] F. E. R. Cesen, L. Csikor, C. Recalde, C. E. Rothenberg, G. Pongrácz, Towards Low Latency Industrial Robot Control in Programmable Data Planes, in: IEEE Conference on Network Softwarization (NetSoft), 2020, pp. 165–169.

[409] I. Kunze, R. Glebke, J. Scheiper, M. Bodenbenner, R. H. Schmitt, K. Wehrle, Investigating the Applicability of In-Network Computing to Industrial Scenarios, in: International Conference on Industrial Cyber-Physical Systems (ICPS), 2021, pp. 334–340.

[410] J. Rüth, R. Glebke, K. Wehrle, V. Causevic, S. Hirche, Towards In-Network Industrial Feedback Control, in: Morning Workshop on In-Network Computing, 2018, p. 14–19.

[411] P. G. Kannan, R. Joshi, M. C. Chan, Precise Time-Synchronization in the Data-Plane using Programmable Switching ASICs, in: ACM Symposium on SDN Research (SOSR), 2019, p. 8–20.

[412] R. Kundel, F. Siegmund, B. Koldehofe, How to Measure the Speed of Light with Programmable Data Plane Hardware?, in: P4 Workshop in Europe (EuroP4), 2019, pp. 1–2.

[413] G. Bonofiglio, V. Iovinella, G. Lospoto, G. D. Battista, Kathará: A Container-Based Framework for Implementing Network Function Virtualization and Software Defined Networks, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), 2018, pp. 1–9.

[414] M. He, A. Basta, A. Blenk, N. Deric, W. Kellerer, P4NFV: An NFV Architecture with Flexible Data Plane Reconfiguration, in: International Conference on Network and Services Management (CNSM), 2018, pp. 90–98.

[415] T. Osiński, H. Tarasiuk, M. Kossakowski, R. Picard, Offloading Data Plane Functions to the Multi-Tenant Cloud Infrastructure using P4, in: P4 Workshop in Europe (EuroP4), 2019, pp. 1–6.

[416] D. Moro, G. Verticale, A. Capone, A Framework for Network Function Decomposition and Deployment, in: International Workshop on the Design of Reliable Communication Networks (DRCN), 2020, pp. 1–6.

[417] T. Osiński, H. Tarasiuk, L. Rajewski, E. Kowalczyk, DPPx: A P4-based Data Plane Programmability and Exposure framework to enhance NFV services, in: IEEE Conference on Network Softwarization (NetSoft), 2019, pp. 296–300.

[418] A. Mohammadkhan, S. Panda, S. G. Kulkarni, K. K. Ramakrishnan, L. N. Bhuyan, P4NFV: P4 Enabled NFV Systems with SmartNICs, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2019, pp. 1–7.

[419] D. Moro, M. Peuster, H. Karl, A. Capone, FOP4: Function Offloading Prototyping in Heterogeneous and Programmable Network Scenarios, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2019, pp. 1–6.

[420] D. Moro, M. Peuster, H. Karl, A. Capone, Demonstrating FOP4: A Flexible Platform to Prototype NFV Offloading Scenarios, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2019, pp. 1–2.

[421] D. R. Mafioletti, C. K. Dominicini, M. Martinello, M. R. N. Ribeiro, R. d. S. Villaça, Piaffe: A place-as-you-go in-network framework for flexible embedding of vnfs, in: IEEE International Conference on Communicaotions (ICC), 2020, pp. 1–6.

[422] X. Chen, D. Zhang, X. Wang, K. Zhu, H. Zhou, P4SC: Towards High-Performance Service Function Chain Implementation on the P4-Capable Device, in: IFIP/IEEE Symposium on Integrated Management (IM), 2019, pp. 1–9.

[423] D. Zhang, X. Chen, Q. Huang, X. Hong, C. Wu, H. Zhou, Y. Yang, H. Liu, Y. Chen, P4SC: A High Performance and Flexible Framework for Service Function Chain, IEEE ACCESS 7 (2019) 160982–160997.

[424] GitHub: P4SC, `https://github.com/P4SC/p4sc`, accessed 01-20-2021 (2021).

[425] H. Lee, J. Lee, H. Ko, S. Pack, Resource-Efficient Service Function Chaining in Programmable Data Plane, in: P4 Workshop in Europe (EuroP4), 2019.

[426] Y. Zhou, J. Bi, C. Zhang, M. Xu, J. Wu, FlexMesh: Flexibly Chaining Network Functions on Programmable Data Planes at Runtime, in: IFIP-TC6 Networking Conference (Networking), 2020, pp. 73–81.

[427] A. Stockmayer, S. Hinselmann, M. Häberle, M. Menth, Service Function Chaining Based on Segment Routing Using P4 and SR-IOV (P4-SFC), in: Workshop on Virtualization in High-Performance Cloud Computing (VHPC), 2020, pp. 297–309.

[428] GitHub: P4-SFC, `https://github.com/uni-tue-kn/p4-sfc-faas`, accessed 01-20-2021 (2021).

[429] R. Ricart-Sanchez, P. Malagon, J. M. Alcaraz-Calero, Q. Wang, Hardware-Accelerated Firewall for 5G Mobile Networks, in: IEEE International Conference on Network Protocols (ICNP), 2018, pp. 446–447.

[430] Ruben Ricart-Sanchez and Pedro Malagon and Jose M. Alcaraz-Calero and Qi Wang, NetFPGA-Based Firewall Solution for 5G Multi-Tenant Architectures, in: IEEE International Conference on Edge Computing (EDGE), 2019, pp. 132–136.

[431] J. Cao, J. Bi, Y. Zhou, C. Zhang, CoFilter: A High-Performance Switch-Assisted Stateful Packet Filter, in: ACM SIGCOMM Conference Posters and Demos, 2018, p. 9–11.

[432] R. Datta, S. Choi, A. Chowdhary, Y. Park, P4Guard: Designing P4 Based Firewall, in: IEEE Military Communications Conference (MILCOM), 2018, pp. 1–6.

[433] P. Vörös, A. Kiss, Security Middleware Programming Using P4, in: International Conference on Human Aspects of Information Security, Privacy, and Trust (HAS), 2016, pp. 277–287.

[434] E. O. Zaballa, D. Franco, Z. Zhou, M. S. Berger, P4Knocking: Offloading host-based firewall functionalities to the network, in: Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), 2020, pp. 7–12.

[435] A. Almaini, A. Al-Dubai, I. Romdhani, M. Schramm, Delegation of Authentication to the Data Plane in Software-Defined Networks, in: IEEE International Conferences on Smart Computing, Networking and Services (SmartCNS), 2019, pp. 58–65.

[436] G. Grigoryan, Y. Liu, LAMP: Prompt Layer 7 Attack Mitigation with Programmable Data Planes, in: ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2018, pp. 1–4.

[437] A. Febro, H. Xiao, J. Spring, Telephony Denial of Service Defense at Data Plane (TDoSD@DP), in: IEEE/IFIP Network Operations and Management Symposium (NOMS), 2018, pp. 1–6.

[438] A. Febro, H. Xiao, J. Spring, Distributed SIP DDoS Defense with P4, in: IEEE Wireless Communications and Networking Conference (WCNC), 2019, pp. 1–8.

[439] M. Kuka, K. Vojanec, J. Kučera, P. Benáček, Accelerated DDoS Attacks Mitigation using Programmable Data Plane, in: ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2019, pp. 1–3.

[440] F. Paolucci, F. Cugini, P. Castoldi, P4-based Multi-Layer Traffic Engineering Encompassing Cyber Security, in: Optical Fiber Communication Conference (OFC), 2018, pp. 1–3.

[441] F. Paolucci, F. Civerchia, A. Sgambelluri, A. Giorgetti, F. Cugini, P. Castoldi, An efficient pipeline processing scheme for programming Protocol-independent Packet Processors, IEEE/OSA Journal of Optical Communications and Networking 11 (2019) 88–95.

[442] Y. Mi, A. Wang, ML-Pushback: Machine Learning Based Pushback Defense Against DDoS, in: ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT), 2019, p. 80–81.

[443] Y. Afek, A. Bremler-Barr, L. Shafir, Network Anti-Spoofing with SDN Data Plane, in: IEEE International Conference on Computer Communications (INFOCOM), 2017, pp. 1–9.

[444] A. C. Lapolli, J. A. Marques, L. P. Gaspary, Offloading Real-time DDoS Attack Detection to Programmable Data Planes, in: IFIP/IEEE Symposium on Integrated Management (IM), 2019, pp. 19–27.

[445] GitHub: ddosd-p4, `https://github.com/aclapolli/ddosd-p4`, accessed 01-20-2021 (2021).

[446] Y.-Z. Cai, C.-H. Lai, Y.-T. Wang, M.-H. Tsai, Improving Scanner Data Collection in P4-based SDN, in: Asia-Pacific Network Operations and Management Symposium (APNOMS), 2020, pp. 126–131.

[447] T.-Y. Lin, J.-P. Wu, P.-H. Hung, C.-H. Shao, Y.-T. Wang, Y.-Z. Cai, M.-H. Tsai, Mitigating SYN flooding Attack and ARP Spoofing in SDN Data Plane, in: Asia-Pacific Network Operations and Management Symposium (APNOMS), 2020, pp. 114–119.

[448] F. Musumeci, V. Ionata, F. Paolucci, M. Cugini, Filippo Tornatore, Machine-learning-assisted DDoS attack detection with P4 language, in: IEEE International Conference on Communicaotions (ICC), 2020, pp. 1–6.

[449] X. Z. Khooi, L. Csikor, D. M. Divakaran, M. S. Kang, DIDA: Distributed In-Network Defense Architecture Against Amplified Reflection DDoS Attacks, in: IEEE Conference on Network Softwarization (NetSoft), 2020, pp. 277–281.

[450] M. Dimolianis, A. Pavlidis, V. Maglaris, A Multi-Feature DDoS Detection Schema on P4 Network Hardware, in: Workshop on Flexible Network Data Plane Processing (NETPROC@ICIN), 2020, pp. 1–6.

[451] D. Scholz, S. Gallenmüller, H. Stubbe, G. Carle, SYN Flood Defense in Programmable Data Planes, in: P4 Workshop in Europe (EuroP4), 2020, p. 13–20.

[452] GitHub: syn-proxy, `https://github.com/syn-proxy`, accessed 01-20-2021 (2021).

[453] K. Friday, E. Kfoury, E. Bou-Harb, J. Crichigno, Towards a Unified In-Network DDoS Detection and Mitigation Strategy, in: IEEE Conference on Network Softwarization (NetSoft), 2020, pp. 218–226.

[454] R. Meier, P. Tsankov, V. Lenders, L. Vanbever, M. Vechev, NetHide: Secure and Practical Network Topology Obfuscation, in: USENIX Security Symposium, 2018, pp. 693–709.

[455] Benjamin Lewis and Matthew Broadbent and Nicholas Race, P4ID: P4 Enhanced Intrusion Detection, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2019, pp. 1–4.

[456] Gorby Kabasele Ndonda and Ramin Sadre, A Two-level Intrusion Detection System for Industrial Control System Networks using P4, in: International Symposium for ICS & SCADA Cyber Security Research (ICS-CSR), 2018, pp. 1–10.

[457] J. Hypolite, J. Sonchack, S. Hershkop, N. Dautenhahn, A. DeHon, J. M. Smith, DeepMatch: Practical Deep Packet Inspection in the Data Plane Using Network Processors, in: ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT), 2020, p. 336–350.

[458] GitHub: DeepMatch, `https://github.com/jhypolite/DeepMatch`, accessed 01-20-2021 (2021).

[459] Q. Qin, K. Poularakis, K. K. Leung, L. Tassiulas, Line-Speed and Scalable Intrusion Detection at the Network Edge via Federated Learning, in: IFIP-TC6 Networking Conference (Networking), 2020, pp. 352–360.

[460] GitHub: syn-proxy, `https://github.com/vxxx03/IFIPNetworking20`, accessed 01-20-2021 (2021).

[461] J. Amado, S. Signorello, M. Correia, F. Ramos, Poster: Speeding up network intrusion detection, in: IEEE International Conference on Network Protocols (ICNP), 2020, pp. 1–2.

[462] D. Chang, W. Sun, Y. Yang, A SDN Proactive Defense Mechanism Based on IP Transformation, in: International Conference on Safety Produce Informatization (IICSPI), 2019, pp. 248–251.

[463] W. Feng, Z.-L. Zhang, C. Liu, J. Chen, Clé: Enhancing Security with Programmable Dataplane Enabled Hybrid SDN, in: ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT), 2019, p. 76–77.

[464] P. Kuang, Y. Liu, L. He, P4DAD: Securing Duplicate Address Detection Using P4, in: IEEE International Conference on Communicaotions (ICC), 2020, pp. 1–7.

[465] X. Chen, Implementing aes encryption on programmable switches via scrambled lookup tables, in: Workshop on Secure Programmable Network Infrastructure (SPIN), 2020, p. 8–14.

[466] GitHub: Tofino AES encryption, `https://github.com/Princeton-Cabernet/p4-projects/tree/master/AES-tofino`, accessed 01-20-2021 (2021).

[467] H. Gondaliya, G. C. Sankaran, K. M. Sivalingam, Comparative Evaluation of IP Address Anti-Spoofing Mechanisms Using a P4/NetFPGA-Based Switch, in: P4 Workshop in Europe (EuroP4), 2020, p. 1–6.

[468] Q. Kang, L. Xue, A. Morrison, Y. Tang, A. Chen, X. Luo, Programmable In-Network Security for Context-aware BYOD Policies, in: USENIX Security Symposium, 2020, pp. 595–612.

[469] GitHub: Poise, `https://github.com/qiaokang92/poise`, accessed 01-20-2021 (2021).

[470] F. Hauser, M. Schmidt, M. Häberle, M. Menth, P4-MACsec: Dynamic Topology Monitoring and Data Layer Protection With MACsec in P4-Based SDN, IEEE ACCESS 8 (2020) 58845–58858.

[471] GitHub: P4-MACsec, `https://github.com/uni-tue-kn/p4-macsec`, accessed 01-20-2021 (2021).

[472] F. Hauser, M. Häberle, M. Schmidt, M. Menth, P4-IPsec: Site-to-Site and Host-to-Site VPN With IPsec in P4-Based SDN, IEEE ACCESS 8 (2020) 139567–139586.

[473] GitHub: P4-IPsec, `https://github.com/uni-tue-kn/p4-ipsec`, accessed 01-20-2021 (2021).

[474] T. Datta, N. Feamster, J. Rexford, L. Wang, SPINE: Surveillance Protection in the Network Elements, in: USENIX Workshop on Free and Open Communications on the Internet (FOCI), 2019, pp. 1–7.

[475] GitHub: SPINE, `https://github.com/SPINE-P4/spine-code`, accessed 01-20-2021 (2021).

[476] Y. Qin, W. Quan, F. Song, L. Zhang, G. Liu, M. Liu, C. Yu, Flexible Encryption for Reliable Transmission Based on the P4 Programmable Platform, in: Information Communication Technologies Conference (ICTC), 2020, pp. 147–152.

[477] G. Liu, W. Quan, N. Cheng, N. Lu, H. Zhang, X. Shen, P4NIS: Improving network immunity against eavesdropping with programmable data planes, in: IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2020, pp. 91–96.

[478] GitHub: P4NIS, `https://github.com/KB00100100/P4NIS`, accessed 01-20-2021 (2021).

[479] M. Liu, D. Gao, G. Liu, J. He, L. Jin, C. Zhou, F. Yang, Learning based adaptive network immune mechanism to defense eavesdropping attacks, IEEE ACCESS 7 (2019) 182814–182826.

[480] J. Deng, H. Hu, H. Li, Z. Pan, K. Wang, G. Ahn, J. Bi, Y. Park, VNGuard: An NFV/SDN Combination Framework for Provisioning and Managing Virtual Firewalls, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2015, pp. 107–114.

[481] H. Zhang, W. Quan, H.-c. Chao, C. Qiao, Smart identifier network: A collaborative architecture for the future internet, Networks Magazine 30 (3) (2016) 46–51.

[482] R. Kumar, V. Babu, D. Nicol, Network Coding for Critical Infrastructure Networks, in: IEEE International Conference on Network Protocols (ICNP), 2018, pp. 436–437.

[483] GitHub: AquaFlow, `https://github.com/gopchandani/AquaFlow`, accessed 01-20-2021 (2021).

[484] D. Goncalves, S. Signorello, F. M. V. Ramos, M. Medard, Random Linear Network Coding on Programmable Switches, in: ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2019, pp. 1–6.

[485] T. Kohler, R. Mayer, F. Dürr, M. Maaß, S. Bhowmik, K. Rothermel, P4CEP: Towards In-Network Complex Event Processing, in: Morning Workshop on In-Network Computing, 2018, p. 33–38.

[486] A. Sapio, I. Abdelaziz, M. Canini, P. Kalnis, DAIET: A System for Data Aggregation Inside the Network, in: ACM Symposium on Cloud Computing (SoCC), 2017, p. 1.

[487] G. C. Sankaran, K. M. Sivalingam, Design and Analysis of Fast IP Address-Lookup Schemes based on Cooperation among Routers, in: International Conference on COMmunication Systems and NETworks (COMSNETS), 2020, pp. 330–339.

[488] Y. Zhang, B. Han, Z.-L. Zhang, V. Gopalakrishnan, Network-Assisted Raft Consensus Algorithm, in: ACM SIGCOMM Conference Posters and Demos, 2017, p. 94–96.

[489] H. T. Dang, M. Canini, F. Pedone, R. Soulé, Paxos Made Switch-y, ACM SIGCOMM Computer Communications Review (CCR) 46 (2016) 18–24.

[490] H. T. Dang, P. Bressana, H. Wang, K. S. Lee, N. Zilbermanand, H. Weatherspoon, M. Canini, F. Pedone, R. Soulé, P4xos: Consensus as a Network Service, IEEE/ACM Transactions on Networking (ToN) 28 (2020) 1726–1738.

[491] GitHub: P4xos, `https://github.com/P4xos/P4xos`, accessed 01-20-2021 (2021).

[492] E. Sakic, N. Deric, E. Goshi, W. Kellerer, P4BFT: Hardware-Accelerated Byzantine-Resilient Network Control Plane, in: IEEE Global Communications Conference (GLOBECOM), 2019, pp. 1–7.

[493] E. Sakic, N. Deric, C. B. Serna, E. Goshi, W. Kellerer, P4BFT: A Demonstration of Hardware-Accelerated BFT in Fault-Tolerant Network Control Plane, in: ACM SIGCOMM Conference Posters and Demos, 2019, p. 6–8.

[494] L. Zeno, D. R. K. Ports, J. Nelson, M. Silberstein, SwiShmem: Distributed Shared State Abstractions for Programmable Switches, in: ACM Workshop on Hot Topics in Networks (HotNets), 2020, p. 160–167.

[495] S. Han, S. Jang, H. Lee, S. Pack, Switch-Centric Byzantine Fault Tolerance Mechanism in Distributed Software Defined Networks, IEEE Communications Letters 24 (2020) 2236–2239.

[496] GitHub: SC-BFT, `https://github.com/MNC-KOR/SC-BFT`, accessed 01-20-2021 (2021).

[497] G. Sviridov, M. Bonola, A. Tulumello, P. Giaccone, A. Bianco, G. Bianchi, LODGE: LOcal Decisions on Global statEs in Progrananaable Data Planes, in: IEEE Conference on Network Softwarization (NetSoft), 2018, pp. 257–261.

[498] G. Sviridov, M. Bonola, A. Tulumello, P. Giaccone, A. Bianco, G. Bianchi, LOcAl DEcisions on Replicated States (LOADER) in programmable dataplanes: Programming abstraction and experimental evaluation, Computer Networks 181 (2020) 107637.

[499] GitHub: LOADER, `https://github.com/german-sv/loader`, accessed 01-20-2021 (2021).

[500] H. Takruri, I. Kettaneh, A. Alquraan, S. Al-Kiswany, FLAIR: Accelerating Reads with Consistency-Aware Network Routing, in: USENIX Symposium on Networked Systems Design & Implementation (NSDI), 2020, pp. 723–737.

[501] S. Luo, H. Yu, L. Vanbever, Swing State: Consistent Updates for Stateful and Programmable Data Planes, in: ACM Symposium on SDN Research (SOSR), 2017, p. 115–121.

[502] J. Xing, A. Chen, T. E. Ng, Secure State Migration in the Data Plane, in: Workshop on Secure Programmable Network Infrastructure (SPIN), 2020, p. 28–34.

[503] GitHub: P4Sync, `https://github.com/jiarong0907/P4Sync`, accessed 01-20-2021 (2021).

[504] Y. Xue, Z. Zhu, Hybrid Flow Table Installation: Optimizing Remote Placements of Flow Tables on Servers to Enhance PDP Switches for In-Network Computing, IEEE Transactions on Network and Service Management (TNSM) (2020) 429–440.

[505] C. Kuzniar, M. Neves, I. Haque, POSTER: Accelerating Encrypted Data Stores Using Programmable Switches, in: IEEE International Conference on Network Protocols (ICNP), 2020, pp. 1–2.

[506] G. C. Sankaran, K. M. Sivalingam, Collaborative Packet Header Parsing in NetFPGA-Based High Speed Switches, IEEE Networking Letters 2 (2020) 124–127.

[507] J. Woodruff, M. Ramanujam, N. Zilberman, P4DNS: In-Network DNS, in: P4 Workshop in Europe (EuroP4), 2019, pp. 1–6.

[508] GitHub: P4DNS, `https://github.com/cucl-srg/P4DNS`, accessed 01-20-2021 (2021).

[509] R. Kundel, L. Nobach, J. Blendin, H.-J. Kolbe, G. Schyguda, V. Gurevich, B. Koldehofe, R. Steinmetz, P4-BNG: Central Office Network Functions on Programmable Packet Pipelines, in: International Conference on Network and Services Management (CNSM), 2019, pp. 1–9.

[510] GitHub: p4se, `https://github.com/opencord/p4se`, accessed 01-20-2021 (2021).

[511] I. Martinez-Yelmo, J. Alvarez-Horcajo, M. Briso-Montiano, D. Lopez-Pajares, E. Rojas, ARP-P4: A Hybrid ARP-Path/P4Runtime Switch, in: IEEE International Conference on Network Protocols (ICNP), 2018, pp. 438–439.

[512] R. Glebke, J. Krude, I. Kunze, J. Rüth, F. Senger, K. Wehrle, Towards Executing Computer Vision Functionality on Programmable Network Devices, in: ACM CoNEXT Workshop on Emerging In-Network Computing Paradigms, 2019, p. 15–20.

[513] J. Xie, C. Qian, D. Guo, M. Wang, S. Shi, H. Chen, Efficient Indexing Mechanism for Unstructured Data Sharing Systems in Edge Computing, in: IEEE International Conference on Computer Communications (INFOCOM), 2019, pp. 820–828.

[514] Y.-S. Lu, K. C.-J. Lin, Enabling Inference Inside Software Switches, in: Asia-Pacific Network Operations and Management Symposium (APNOMS), 2020, pp. 1–4.

[515] A. Yazdinejad, R. M. Parizi, A. Dehghantanha, K.-K. R. Choo, P4-to-blockchain: A secure blockchain-enabled packet parser for software defined networking, Computers & Security Journal 88 (2019) 101629.

[516] T. Osiński, H. Tarasiuk, P. Chaignon, M. Kossakowski, P4rt-OVS: Programming Protocol-Independent,Runtime Extensions for Open vSwitch with P4, in: IFIP-TC6 Networking Conference (Networking), 2020, pp. 413–421.

[517] GitHub: P4rt-OVS, `https://github.com/Orange-OpenSource/p4rt-ovs`, accessed 01-20-2021 (2021).

[518] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. Ports, P. Richtarik, Scaling Distributed Machine Learning with In-Network Aggregation, in: USENIX Symposium on Networked Systems Design & Implementation (NSDI), 2021.

[519] SwitchML, `https://github.com/p4lang/p4app-switchML`, accessed 15-02-2022 (2021).

[520] F. Yang, Z. Wang, X. Ma, G. Yuan, X. An, SwitchAgg: A Further Step Towards In-Network Computing, in: IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom), 2019.

[521] S. R. Li, R. W. Yeung, N. Cai, Linear Network Coding, IEEE Transactions on Information Theory 49 (2003) 371–381.

[522] Deutsche Telekom AG: Deutsche Telekom's Access 4.0 platform goes live, `https://www.telekom.com/en/media/media-information/archive/deutsche-telekom-s-access-4-0-platform-goes-live-615974`, accessed 05-17-2021 (2021).

[523] O-RAN Alliance, `https://www.o-ran.org/`, accessed 05-17-2021 (2021).

3 Accepted Manuscripts (Additional Content)

3.1 LoCoSDN: A local controller for operation of of switches in non-SDN networks

LoCoSDN: A Local Controller for Operation of OF Switches in non-SDN Networks

Mark Schmidt, Frederik Hauser, Bastian Germann, and Michael Menth
Chair of Communication Networks, University of Tuebingen, Tuebingen, Germany
Email: {mark-thomas.schmidt,frederik.hauser,menth}@uni-tuebingen.de, bastian.germann@student.uni-tuebingen.de

***Abstract*—Hybrid OpenFlow (OF) switches can be operated as legacy switches with a local control plane or as OF switches that are controlled by an SDN controller. As a first thought, hybrid switches are the first choice for network administrators as they facilitate the transition from a traditionally operated towards a software-defined network. However, we encountered several limitations as most of these switches are based on a legacy hardware platform that is augmented with a limited set of OF features. As a consequence, some packet processing pipelines of hybrid switches filter out packets before they pass the OF part. To avoid these issues, we developed LoCoSDN, an SDN architecture that imitates the hybrid switch behavior on OF-only devices, i.e., they can be operated in traditional networks and without any special knowledge about SDN. LoCoSDN contains a Cisco-like CLI and configuration format so that configurations from existing devices can be imported. It differentiates between a startup and running configuration, and provides reconfigurability during operation and support for an additional SDN controller. We implemented LoCoSDN as a lightweight SDN controller for the Ryu SDN controller framework that can be run on a single-board computer.**

I. Introduction

The current landscape of hardware-based network switches includes three different architectures that are depicted in Figure 1. Traditional network switches (1) consist of a data plane and a control plane. The data plane is responsible for fast packet processing and programmed by the control plane. It contains algorithms for path computations and implements various network mechanisms and protocols along with user interfaces that allow network administrators to configure the device. Software-defined networking (SDN) splits the strong binding between data and control plane. OpenFlow (OF) is the most widespread standard for SDN that defines an architecture and a communication protocol between SDN switches and the SDN controller. OF-only switches (2) only contain a data plane and require an SDN controller that provides all functionality of the former control plane. Protocols and mechanisms need to be implemented in software running on the SDN controller. Hybrid OF switches (3) contain the control plane of a legacy switch with an interface to an OF controller. Forwarding decisions are either made by the switch's legacy control plane or by an external SDN controller. Current hybrid OF switches are mostly legacy hardware switches that are augmented with a limited set of OF features. Therefore, hardware parts of the switch such as TCAM, ASIC, and switch memory are used to implement OF support through firmware updates.

This work was supported by the bwNET100G+ project which is funded by the Ministry of Science, Research and the Arts Baden-Württemberg (MWK). The authors alone are responsible for the content of this paper.

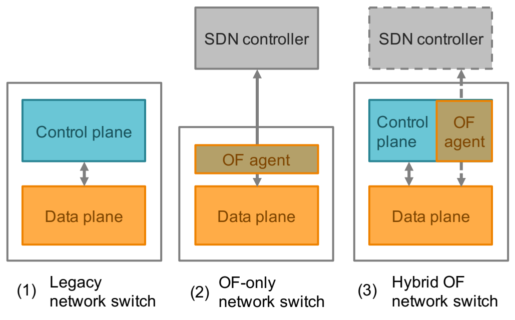


Fig. 1: Three different architectures for network switches. The legacy switch combines a control and data plane on a single device. The OF-only switch requires an SDN controller as substitute for the local data plane. The hybrid OpenFlow switch combines a local control plane with an external SDN controller.

Nowadays, only a few companies are ready to apply SDN control to their networks. Others want to go this step only in the future. When they invest now into new hardware, they have two choices so that they can reuse the equipment in the SDN network: hybrid OF switches or OF-only switches. In comparison to OF-only switches, hybrid OF switches bring many benefits: legacy control plane functions such as L2 switching, L3 routing, and VLAN tagging as well as legacy user interfaces are further on usable. However, we encountered several limitations that are caused by the difficulties in distinguishing between a local control plane and OF control. In particular, we observed packet processing pipelines of hybrid OF switches that filter out packets so that they did not reach the SDN controller.

As an alternative for current hybrid OF switch architectures, we propose LoCoSDN. It imitates the hybrid switch behavior by coupling an OF-only switch with a local SDN controller that can be run on cheap commodity hardware platforms, e.g., a Raspberry Pi. In contrast to hybrid OF architectures, all parts and interfaces of LoCoSDN are open-source. LoCoSDN offers the legacy functions and user interfaces of a legacy switch so that network administrators do not have to know SDN specifics as long as the hybrid switch is operated in the legacy mode. If LoCoSDN should be used within an SDN infrastructure, it may be configured to connect to an external SDN controller and pass-through the OF communication to the switch.

The remainder of the paper is structured as follows. Section II provides an overview of related work. In Section III, we outline the basic idea of our concept. The internal design of the local SDN controller as central part of LoCoSDN is described in Section IV. The prototypical implementation and a functional validation is described in Section V. Section VI concludes this work.

II. Related Work

In the following, we describe the characteristics and architecture of legacy network switches, introduce OF-only, hybrid, and whitebox OF switches, and discuss related approaches for bringing hybrid-like behavior to OF-only switches.

A. Legacy Network Switches

Legacy network switches offer many control plane functions on different layers. Protocol-related functionalities include ARP, MPLS, Q-in-Q, and ICMP, additional features include access control lists (ACLs), mechanisms for authentication and authorization, and vendor-specific interfaces to external systems.

This large variety of control plane functions requires extensive and complex configuration. Therefore, legacy network switches offer user interfaces and network management protocols. Most legacy network switches include two types of user interfaces: web interfaces and command line interfaces (CLIs). Web interfaces are graphical user interfaces that can be accessed using a browser. CLIs represent simpler interfaces where network administrators connect to a switch either over a serial, Telnet, or secure shell (SSH) connection. Then, ASCII-based human-to-machine languages are either used in an interactive console prompt or for manually typing scripts or configurations. Although syntax can be vendor-specific, Cisco-like CLIs emerged as de-facto standard for legacy network devices from various manufacturers. They are even used in software tools [1] and operating systems [2] for interactive configuration. Network management protocols such as SNMP [3] or NETCONF [4] aim at network infrastructures with many devices that need to be managed and operated.

B. OF Switches

The current landscape of OF switches can be divided into OF-only and hybrid switch architectures. We classify whitebox switches as either OF-only or hybrid OF switches, depending on the particular form.

1) OF-Only Switches: The minimal architecture for an OF switch strictly follows the OF switch specification [5] and does not contain a legacy control plane. Instead, it has a microprocessor running a firmware that includes an OF agent as interface between the OF-only switch and an SDN controller. OF-only switch architectures are mostly found in software switches: Lagopus [6] or LINC-Switch [7] are examples for OF-only switches. However, there is only a limited number of OF-only hardware switches such as the NEC ProgrammableFlow series [8], the NoviFlow NOVISWITCHes [9], or the experimental SDN switch platform Zodiac FX [10].

2) Hybrid OF Switches: This type of OF switch architecture combines a legacy control plane with an additional option to use an external SDN controller. Figure 2 depicts the internal architecture of a hybrid OF switch. The data plane is programmed by a local control plane (1). It includes an on-board logic (2) implementing basic network functionalities such as L2 switching, L3 routing, or VLAN tagging, and contains interfaces to external systems (3). The OF agent (4) as additional component on the control plane represents the interface to the SDN controller (5).

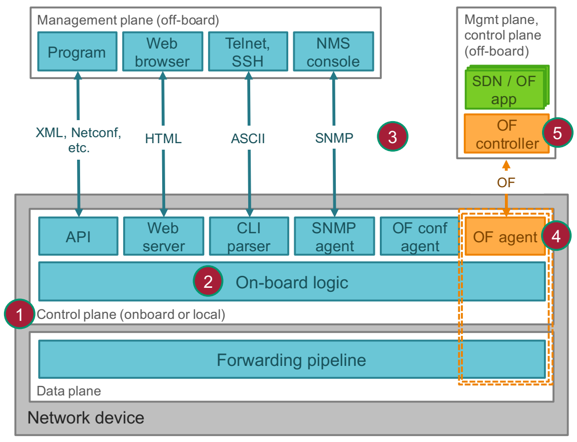


Fig. 2: Internal architecture of a hybrid OF switch consisting of a data and control plane (according to [11]).

The control plane firmware of hardware hybrid OF switches is closed-source and mostly tailored to a particular hardware model. It comes with a basic set of features, while additional features can be activated by purchasing upgrade licenses. In addition, the interface between the control and data plane is based on a proprietary protocol. As a result, neither the operating system nor the functionalities can be extended or exchanged. For virtualized environments, also software hybrid switches like Open vSwitch (OVS) [12] exist.

Still, this concept offers many benefits over OF-only switches. First, control plane functions such as L2 switching or L3 routing need to be implemented on the SDN controller first when OF-only switches are rolled out. Hybrid OF switches allow a step-by-step transition: legacy control plane functions are still available on the local control plane, an SDN controller is not required at this point. However, an SDN controller may introduce new functionalities at a later time. Second, hybrid OF switches enable parallel operation of local control plane functions and of an SDN controller. As an example, ARP functionality for L2 switching may be provided by the local control plane whereas L3 routing is achieved by an SDN controller. This reduces OF control plane traffic and load on the SDN controller as some forwarding decisions are made locally. It also reduces the latency for installing forwarding rules resulting from the communication between SDN switches and the SDN controller which may be physically distant. Last, the hybrid architecture lowers the barrier for network administrators to adopt OF-based SDN because features and well-known user interfaces such as CLIs of legacy switches are still present.

As a consequence of the hybrid architecture, forwarding decisions for incoming data are either made by the local control plane or the SDN controller. That requires a separation mechanism. One approach defines the separation on a per port or per VLAN base. Another approach defines the separation on a per function base. As an example, an SDN controller is used for L3 routing while L2 switching based on MAC addresses still remains on the legacy control plane.

Typical hybrid switches are mostly based on a traditional switch design which are extended with OF functionality. This architecture may lead to several problems. E.g., in a previous work [13], we experienced problems in forwarding EAPoL packets in an 802.1X test infrastructure from a hybrid switch to an SDN controller. The manufacturer's OF documentation [14] for that particular switch model shows a list of incompatibilities when using OF.

3) Whitebox Switches: The architecture is mainly driven by large network operators and enterprises. To lower CAPEX, low-cost original device manufacturers use generic, off-the-shelf switching hardware to build SDN hardware switches as an alternative to legacy, OF-only, or hybrid OF network switches from big vendors. Whitebox switches like the OPEN NETWORKING series from Edgecore [15] contain merchant silicon switching chips, a commodity CPU, and memory. They rely on an either proprietary or open-source network operating system (NOS) [16] which can be installed via the Open Network Install Environment (ONIE) [17]. Two widely used examples are Cumulus Linux [18] and the Facebook Open Switching System (FBOSS) [19]. Cumulus Linux is an example for a heavyweight NOS that requires a powerful computing platform on the network hardware, i.e., bare metal switches that include multi-core CPUs, RAM, and a SSD. It is based on a modified Debian distribution with interfaces to hardware components such as routing tables and switching ASICs. Besides providing an OF interface for complete external control, the switch can be operated in a legacy manner as well. Linux tools such as iproute2 [20] or Quagga [1] can be used to control the data plane just like for hybrid OF switches as discussed before. FBOSS is an example for an open-source and rather lightweight NOS. API libraries such as OpenNSL [21] and OF-DPA [22] run on a microprocessing unit to interface specific switch ASICs.

Figure 3 depicts the difference between the hybrid and OF-only whitebox switch architectures. Hybrid whitebox switch architectures for running NOS such as Cumulus Linux contain legacy network functions as part of their control plane. OF-only whitebox switch architectures solely contain an OF agent as interface to an external SDN controller.

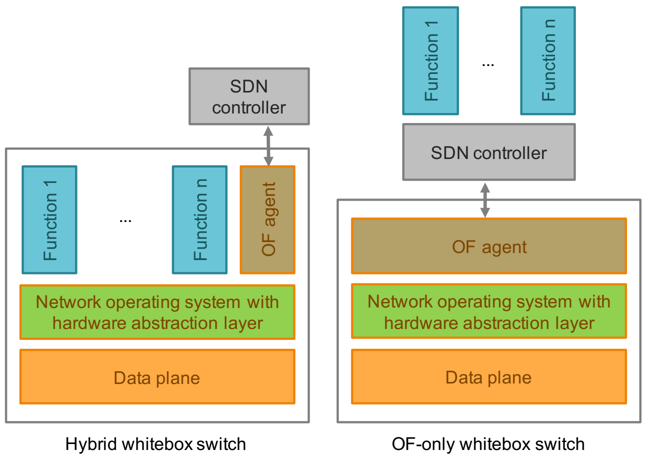


Fig. 3: Comparison of hybrid and OF-only whitebox switch architectures.

C. Alternative Hybrid OF Switch Architectures

In contrast to OF-only switches, hybrid OF switches contain a local control plane that primarily consists of two functional parts: control plane functionalities and interfaces for monitoring and configuration. In the following, we present related approaches that try to introduce hybrid-like behavior to infrastructures that are built of OF-only switches and SDN controllers by implementing the two functional parts.

1) Legacy Control Plane Functionalities: Valve [23] is an SDN application that runs on top of the Ryu SDN controller framework. It introduces various functionalities from legacy control planes such as VLAN tagging, IPv4/IPv6 ACLs, auto-configuration of ports, and port statistics. Valve is configured within a static YAML file and can be run in parallel with other SDN controllers. Therefore, flow entries installed by Valve are identified by a special and unique OF cookie. Faucet [24], [25] is an improvement of Valve and currently used in various enterprise networks, including the administration network of the Open Networking Foundation. Faucet supports VLAN switching, IPv4/IPv6 routing, ACLs, port mirroring, and policy-based forwarding. In addition, it pushes statistical information to an external database and offers to use Grafana to construct visualization views. Identical to Valve, Faucet uses individual YAML files for configuration. Dragonflow [26] is an SDN controller for the OpenStack Neutron [27] platform. It is also based on the Ryu SDN controller framework and provides several applications that implement legacy control plane functionalities such as L2 switching, L3 routing, and VLAN tagging. However, due to the fixed integration into the OpenStack platform, the configuration of DragonFlow is fully automated and based on the platform's orchestrator.

2) User Interfaces to Control Plane Functions: CLIs represent the common way to configure traditional switches. The following approaches introduce CLIs on SDN controllers to ease the transition for network administrators. SDNsh [28] is an advanced CLI for the OpenDaylight SDN controller platform and the Open Network Operating System. It offers a Python-based CLI depending on a running instance of sdncon, a northbound interface of the two platforms. OpenMUL [29] is an SDN controller that provides a CLI to specify the entries of the flow tables, specify actions, and define OF pipelines. However, both approaches do not come with built-in commands for traditional networking.

III. Architecture of LoCoSDN

In this section, we describe the physical setup of LoCoSDN.

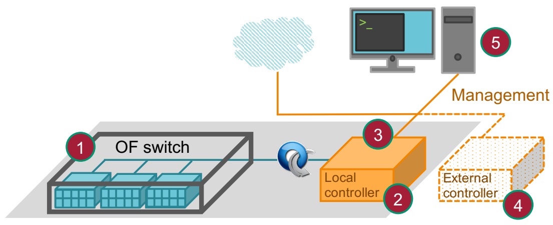


Fig. 4: Physical setup of LoCoSDN consisting of an OF switch, a local controller, a management station, and an optional external controller.

Figure 4 depicts the physical architecture of LoCoSDN. An OF-only switch (1) is directly connected via an Ethernet cable to a cheap single-board computer (SBC) that runs a local SDN controller (2). OF is used as open and extensible southbound protocol on the link between the data (1) and control plane (2). The SBC provides another physical interface (3) to connect to a management network that can consist of additional external SDN controllers (4) or management computers (5).

We chose to use an *OF-only switch* (1) as datapath element. As introduced in Section II, it represents a minimal architecture for an OF switch which only forwards packets on physical ports based on entries in the OF flow table. It has an OF agent on a microprocessor that holds an OF connection with an SDN controller over an Ethernet link. *The LoCoSDN's controller* (2) is the key element of the architecture. It runs on an SBC and realizes the entire control plane functionality of the LoCoSDN architecture. It implements all control plane functionalities such as L2 switching, L3 routing, or VLAN tagging and includes known user interfaces such as a CLI and web interface. LoCoSDN's SBC running the local SDN controller contains another physical Ethernet port. It is used to connect a management network that can host *additional SDN controllers* (4) and *management computers* (5). Additional SDN controllers may be either used as a substitute or in conjunction with LoCoSDN's local SDN controller. External SDN controllers may be responsible for multiple switches and therefore provide a more global view of the network which enables global optimizations and more complex traffic steering. Management computers configure switches and routers either through network management protocols such as SNMP or user interfaces. LoCoSDN's SDN controller offers a Cisco-like CLI and web-based GUI for administration. The size of the management network can vary greatly, ranging from a single management computer up to an infrastructure with numerous distributed SDN controller instances.

IV. The LoCoSDN Controller

In the following, we present the modes, the module system, the storage model, and the configuration model of the LoCoSDN controller in detail.

A. Modes

LoCoSDN supports three modes of operation: local controller, remote controller, and hybrid mode. In *local controller mode*, no external SDN controller is used at all. Instead, all desired network functionalities are either provided by basic SDN applications that are part of the LoCoSDN controller or by additional modules. In *remote controller mode*, LoCoSDN's local SDN controller is not used at all. Instead, all control plane functionalities are provided by an external SDN controller that is attached to the management network. In use cases where a remote controller manages the entire switch, the local SDN controller is not needed, i.e., the switch has to be reconfigured to connect to an external controller and after that, the local controller can be switched off. In *hybrid mode*, data plane control can be either done by the local controller or by an external SDN controller on a per-port base. This mode is useful for scenarios where some ports of the switch are managed by the local controller and some ports are configured by a global controller. Therefore, the network administrator needs to define which physical port of the OF switch belongs to which controller. In this mode, LoCoSDN's local controller acts as a proxy to external SDN controllers and ensures that the SDN controllers can only install flow rules lying in their respective range of authority. Figure 5 depicts an exemplary scenario of the hybrid mode. Several physical ports are controlled by LoCoSDN's local SDN controller (1) where other physical ports are controlled by the external SDN controller (2).

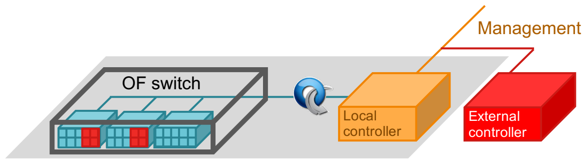


Fig. 5: Multiple physical ports of a LoCoSDN switch are controlled by the local SDN controller, other physical ports are controlled by an external SDN controller.

B. Module System

When operated in local controller or hybrid mode, LoCoSDN's SDN controller provides the control plane functionalities of a legacy network switch. L2 switching, L3 routing, and VLAN tagging are examples for legacy control plane network functions, the AAM as presented in [13] is a novel functionality. LoCoSDN splits those different functions into dedicated modules. Figure 6 depicts the mechanism of LoCoSDN's module system. It consists of three layers: the SDN controller layer (1), the module layer (2), and the configuration layer (3). Each module consists of controller logic that realizes the actual functionality and a parser for the configuration data.

As depicted in Figure 6, LoCoSDN's SDN controller already includes basic modules such as L2 switching, L3 routing, and VLAN tagging depicted in (2)(a). Missing features can be implemented in add-on modules like shown in (2)(b). An example add-on module is the AAM [13].

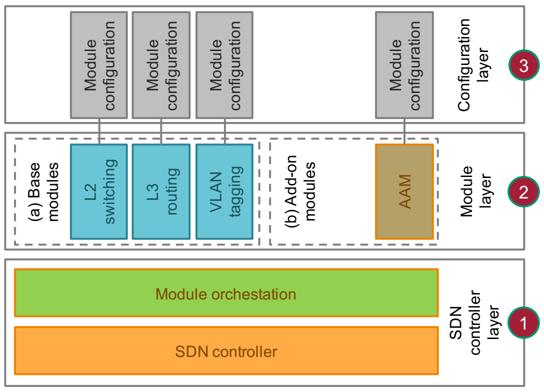


Fig. 6: Simple module mechanism for the LoCoSDN switch consisting of a base module that registers add-on modules and configuration data.

C. Storage Model

Figure 7 shows the storage model of LoCoSDN that is split up into two partitions. The *firmware partition* (1) holds a minimal Linux operating system with an SDN controller. Basic control plane functionalities as discussed before are encapsulated in base modules. The firmware partition is bundled as a downloadable image so that upgrading the whole firmware consisting of operating system and SDN controller is an easy task. The *persistent storage partition* (2) contains the module configurations and add-on modules. This partition is not affected by changes in the firmware partition so that the operating system and the SDN controller can be upgraded without losing configuration data or base modules.

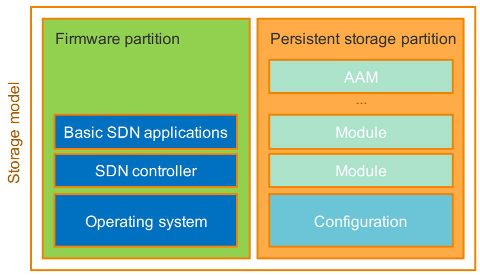


Fig. 7: Storage layout of the LoCoSDN consisting of a firmware partition and a partition for configuration data and additional modules.

D. Configuration Model

Figure 8 depicts the internal structure of the local controller. It consists of three main blocks: the configuration front end (1), the configuration back end (2), and the actual SDN controller (3).

The *configuration front end* includes two different user interfaces: a CLI and a web-based GUI. A Cisco-like CLI offers administrators full access to all features of LoCoSDN's local SDN controller, i.e., all functionalities can be configured exactly like on legacy network switches. To enable a soft transition from existing deployments that are based on traditional switches, it allows importing already existing network configuration data in the Cisco format. An adopted Cisco format is used as native configuration format. The web-interface provides a simplified interface for end users.

Both user interfaces communicate via API calls with the *configuration back end*. Like in typical traditional switches or routers, the configuration distinguishes between a startup config and a running config. The startup config represents the persistent configuration that is loaded at system start and then copied to a temporary running config. Changes to the configuration are only applied to the running config and require an explicit write command to be made persistent. The advantage of this mechanism is that the startup config should always be in a working state and in case of a misconfiguration, a reboot returns into a working state.

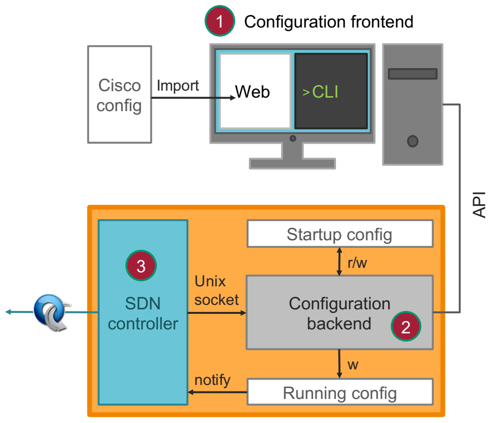


Fig. 8: The local controller consists of a configuration front and back end and an SDN controller communicating with each other.

The actual *SDN controller* realizes the functionality of a traditional L3 switch and configures the SDN switch via OF accordingly. To that end, the controller needs to implement parsers for the different parts of the running config and the corresponding functionality to translate the configuration data into flow rules.

In case of changes to the running config, the controller gets notified to re-read the data. After that, the modified flow rules are applied at the switch. Information from the switch like, e.g., port-up or port-down messages, are forwarded to the configuration back end via a custom API and visualized in the configuration front end.

V. Prototypical Implementation & Functional Validation

In the following, we describe the hardware and software that we used to implement the prototype and describe our prototypical validation.

A. Prototypical Implementation

Figure 9 depicts the hardware architecture of the prototypical implementation. The prototype runs on an inexpensive experimental hardware platform and focuses on features rather than performance. We decided to use a Zodiac FX [10] as OF-only switch. It is the result of a crowdfunding campaign [30] initiated in 2015 by Northbound Networks, the current seller of the switch. The Zodiac FX is an open hardware platform that is based on the Atmel ATSAM4E Cortex M4 microprocessor. It offers four 100 Mb/s links and support for OF in version 1.0 and 1.3. The firmware of the Zodiac FX is open source [31] which allowed us to fix a firmware bug in VLAN tagging. We further decided to use the Raspberry Pi 3 [32] microcomputing platform with a quad-core ARM CPU and 1GB RAM as SBC running LoCoSDN's local SDN controller. The first USB port of the Raspberry Pi is used as power supply to the Zodiac FX switch, the other USB port connects a USB NIC for the management network. This platform provides enough hardware resources to run LoCoSDN's local SDN controller with typical network functions such as L2 switching, L3 routing, or VLAN tagging. If more complex and CPU-intense network functions should be provided, the hardware platform might be substituted by a more powerful platform.

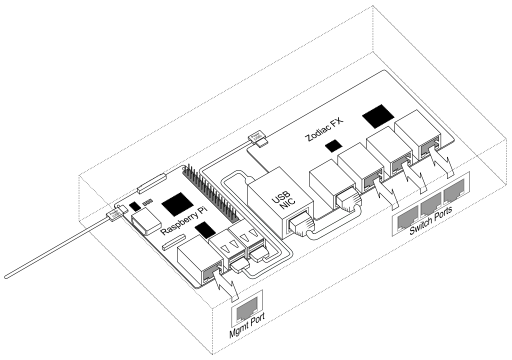
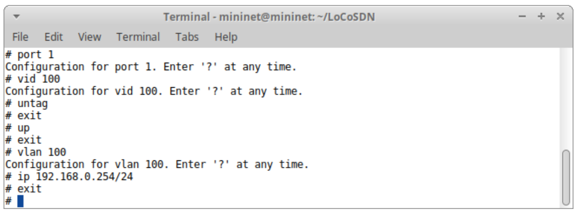


Fig. 9: Assembled hardware platform of LoCoSDN consisting of a Zodiac FX OF-only switch, a Raspberry Pi 3, and a USB NIC.

We chose RASPBIAN [33] as Linux-based operating system for the SBC. We implemented LoCoSDN's local SDN controller using the Ryu SDN controller framework [34]. It is implemented in Python 3 and optimized for rapid prototyping using a lightweight architecture. The implementation of LoCoSDN's local SDN controller encompasses two parts: the network functions and the user interfaces of legacy control planes. On the network function side, we implemented L2 switching, L3 routing, and VLAN tagging. On the user interface side, we implemented a specific northbound interface to communicate with the configuration back end that uses two different methods. The SDN controller reads its configuration data from a file and regularly polls it for updates. A Unix socket signals events from the switch to the configuration back end. The configuration back end and front end are also implemented in Python and communicate with each other directly via API calls. For evaluation purposes, we implemented a fully functional CLI as shown in Figure 10 and a read-only web-interface to display the current configuration and state of the LoCoSDN switch.

B. Functional Validation

To validate the prototypical implementation, we built a simple test setup consisting of three LoCoSDN switches that are connected by Ethernet links like shown in Figure 11. Each


Terminal - mininet@mininet: ~/LoCoSDN
File Edit View Terminal Tabs Help

```
# port 1
Configuration for port 1. Enter '?' at any time.
# vid 100
Configuration for vid 100. Enter '?' at any time.
# untag
# exit
# up
# exit
# vlan 100
Configuration for vlan 100. Enter '?' at any time.
# ip 192.168.0.254/24
# exit
#
```

Fig. 10: Screenshot of LoCoSDN's CLI presenting the VLAN configuration.

LoCoSDN switch has a Raspberry Pi 3 acting as network client attached to it via an Ethernet link. The management port of each LoCoSDN switch is connected to a 5-port legacy switch that also includes a notebook for management. For an initial test, we configure the Client1 and Client2 with an IP address in the subnet 10.0.1.0/24. The two clients can reach each other via the switch. Next, we configure the port towards Client1 with VLAN 1 and the port towards Client2 with VLAN 2. As expected, the two clients cannot reach each other any longer. As a final configuration test, we configure for Client1, Client2, and Client3 an IP address in the IP subnet 10.0.2.0/24 and set their access ports on the switches to VLAN 2. The ports connecting both switches are configured with tagged in VLAN 2. Initially only the clients in VLAN 2 can communicate. After enabling IP routing in the CLI, Client1 can reach the other Clients. To test the communication in the other direction from the switch to the configuration back end, we keep the setup like in the previous experiment. The CLI shows that all ports are up. When we unplug Client2, we receive a port-down event in the CLI and the port status is set accordingly.

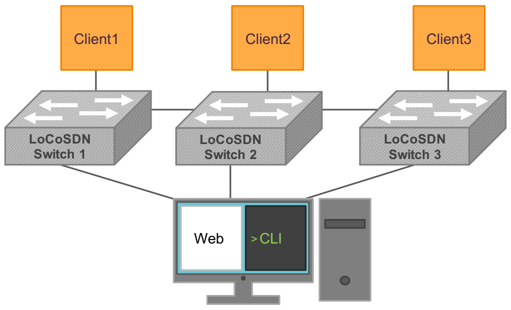


Fig. 11: The evaluation setup.

VI. Conclusion

In this paper, we presented LoCoSDN as an alternative architecture to hybrid switches. It consists of an OpenFlow-only switch and a local SDN controller running on a cheap single board computer. As with typical hybrid switches, LoCoSDN supports traditional network operation and SDN-based operation. To that end, the local SDN controller provides the basic features of a typical L3 switch which can be extended via a simple module system. In SDN mode, the SDN applications can either run within the local SDN controller or an external SDN controller with a global network view can be attached. In the latter case, the local SDN controller acts as a proxy. We evaluated LoCoSDN with the help of a prototype in a simple testbed.

References

[1] B. Gurudoss, P. Jakma, T. Teräs, G. Troxel, and Quagga developer team, "Quagga Routing Suite," http://www.nongnu.org/quagga/.

[2] Graeme McKerrell, "CLISH Documentation," http://clish.sourceforge.net/clish-0.7.3/.

[3] J. Case, R. Mundy, D. Partain, and B. Stewart, "Introduction and Applicability Statements for Internet Standard Management Framework," RFC 3410 (Proposed Standard), Internet Engineering Task Force, Dec. 2002.

[4] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network Configuration Protocol (NETCONF)," RFC 6241 (Proposed Standard), Internet Engineering Task Force, Jun. 2011.

[5] Open Networking Foundation members, "OpenFlow Switch Specification," https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf, The Open Networking Foundation, 2017.

[6] Y. Nakajima, K. Kaplita, and Lagopus developer team, "Lagopus switch," http://www.lagopus.org/.

[7] K. Rutka and FlowForwarding.org community, "LINC - OpenFlow software switch," https://github.com/FlowForwarding/LINC-Switch.

[8] NEC Corporation, "NEC ProgrammableFlow," http://www.nec.com/en/global/prod/pflow/, 2017.

[9] NoviFlow Inc., "NoviFlow NoviSwitch," https://noviflow.com/products/noviswitch/, 2017.

[10] Northbound Networks, "Zodiac FX," https://northboundnetworks.com/products/zodiac-fx, 2016.

[11] E. Verizhnikova, "SDN – Hybrid architecture," https://www.slideshare.net/verizhnikova/larch-network-hybridization-jan2013.

[12] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of open vswitch," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, 2015, pp. 117–130. [Online]. Available: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff

[13] F. Hauser, M. Schmidt, and M. Menth, "Establishing a session database for sdn using 802.1x and multiple authentication resources," in *2017 IEEE International Conference on Communications (ICC)*, May 2017, pp. 1–7.

[14] Hewlett Packard Enterprise, "HP Switch Software OpenFlow v1.3 Administrator Guide nl K/KA/KB/WB 15.18," http://h20566.www2.hpe.com/hpsc/doc/public/display?sp4ts.oid=7074783&docLocale=en_US&docId=emr_na-c04777809.

[15] EdgeCore Networks Corporation, "EdgeCore OPEN NETWORKING," https://noviflow.com/products/noviswitch/, 2017.

[16] Edge Core Networks, "Open Networking Solutions for Data Center, Telecom, and Enterprise," http://www.edge-core.com/solution-inquiry.php?cls=5\&id=7.

[17] Cumulus Networks Inc. and Open Compute Project, "Open Network Install Environment (ONIE)," http://onie.org/about/, 2015.

[18] Cumulus Networks Inc., "Cumulus Linux - The world's most flexible open network operating system for bare metal switches," https://cumulusnetworks.com/products/cumulus-linux/, 2013.

[19] Facebook Inc., "Facebook Open Switching System (FBOSS)," https://github.com/facebook/fboss, 2015.

[20] A. Kuznetsov and S. Hemminger, "iproute2: Utilities for Controlling TCP/IP Networking and Traffic," http://www.linuxfoundation.org/collaborate/workgroups/networking/iproute2, 2012.

[21] Broadcom Limited, "OpenNSL," https://github.com/Broadcom-Switch/OpenNSL, 2017.

[22] ——, "OF-DPA," https://github.com/Broadcom-Switch/of-dpa, 2017.

[23] B. Cowie, C. Lorier, and J. Stringer, "valve," https://github.com/wandsdn/valve, 2016.

[24] J. Bailey and S. Stuart, "Faucet: Deploying sdn in the enterprise," *Queue*, vol. 14, no. 5, pp. 30:54–30:68, Oct. 2016.

[25] J. Bailey, "faucet," https://github.com/REANNZ/faucet, 2017.

[26] E. Gampel and DragonFlow drivers team, "dragonflow," https://launchpad.net/dragonflow.

[27] OpenStack Foundation, "Openstack," https://www.openstack.org/, 2017.

[28] opendaylight, "archived-net-virt-platform/cli," https://github.com/opendaylight/archived-net-virt-platform/tree/master/cli, 2013.

[29] OpenMUL Foundation, "OpenMUL – High Performance SDN," http://www.openmul.org/.

[30] Northbound Networks, "Zodiac FX: The world's smallest OpenFlow SDN switch," https://www.kickstarter.com/projects/northboundnetworks/zodiac-fx-the-worlds-smallest-openflow-sdn-switch, 2016.

[31] ——, "Zodiac FX Firmware," https://github.com/NorthboundNetworks/ZodiacFX, 2016.

[32] RASPBERRY PI FOUNDATION, "RASPBERRY PI 3 MODEL B," https://www.raspberrypi.org/products/raspberry-pi-3-model-b/, 2015.

[33] M. Thompson and P. Green, "Raspbian," https://www.raspbian.org/.

[34] Ryu SDN Framework Community, "Ryu," http://osrg.github.io/ryu/.

3.2 A Master Course on Network Softwarization: Lectures and Practical Assignments

A Master Course on Network Softwarization: Lectures and Practical Assignments

Frederik Hauser, Mark Schmidt, Michael Menth
Chair of Communication Networks, University of Tuebingen, Tuebingen, Germany
Email: {frederik.hauser,mark-thomas.schmidt,menth}@uni-tuebingen.de,

I. Motivation

The bwNET100G+ research project focuses on innovating campus and scientific wide area networks using software-defined networking (SDN) and network function virtualization (NFV), two concepts that are also referred to as network softwarization. In that context, Master students at the University of Tuebingen have the opportunity to work on prototypical implementations within thesis projects. Examples are [1]–[3].

As such projects are demanding, students need a solid understanding of network softwarization and ideally have some practical experience in that area. In the past, they were instructed to read scientific papers from the ONF reading list [4] and to program applications for SDN controllers using tutorials such as [5] or [6]. However, this was rather inefficient as the process was difficult for the students and still required lots of individual supervision. Therefore, we established a lecture course including practical assignments at Master level as preparation for a thesis project on network softwarization.

The course prerequisites basic knowledge in communication networks. It provides an overview of legacy communication network management and introduces network softwarization as novel and relevant development. It teaches both theoretical fundamentals and implementation approaches, and discusses new challenges in that field. Moreover, students get first practical experiences with virtualized SDN using Mininet and program SDN applications for the Ryu SDN controller. While time-intense courses on network softwarization already exist [7], [8] at other institutions, our new course is tailored to a workload of only 3 ECTS including 90 hours of weekly lectures over 10 weeks during the summer term plus work on practical assignments and exam preparation.

II. Concept

The concept of the course includes three parts: lectures, programming assignments, and a final exam.

A. Lectures

The first part of the course focuses on theoretical education in form of a lecture's presentation that is based on slides. The contents are split up in seven chapters as following.

1) Introduction to Network Softwarization: The first chapter covers the transition from legacy to softwarized networks. Legacy management concepts, e.g., SNMP and NETCONF, more active network concepts, e.g., ForCES and 4D, and more recent concepts, e.g., OpenFlow SDN and P4, are discussed.

2) OpenFlow: The second chapter focuses on covering both, the architecture and protocol of OpenFlow. The lecture introduces OpenFlow in version 1.0 and 1.5.1. and shows the transition from the first to the latest feature set while describing the respective novelties such as flow table pipelines or meters.

3) SDN Controller: The third chapter describes the SDN application and control layer. It covers the architecture, design principles, and control plane functions of SDN controllers and their interfaces along concrete implementations, e.g., REST for northbound interfaces. The chapter closes with an overview of pupular SDN controllers, e.g., Ryu, OpenDaylight, and ONOS.

4) SDN Switches: The fourth chapter revises the hardware architecture of legacy routers and switches in the beginning. SDN software switches, Whitebox switches, OpenFlow-only switches, and SDN hybrid switches and their respective deployment scenarios are described in detail afterwards. The chapter closes with an in-depth view on the status quo.

5) SDN Use Cases: The fifth chapter introduces datacenter, enterprise, campus and wide area networks as predestined environments for the application of network softwarization. Especially benefits, but also limitation in comparison to legacy technologies are discussed in detail. The chapter closes with an in-depth description of particular concepts from our current and past research activities.

6) Virtualization Techniques: The sixth chapter focuses on the strongly related concept of computer virtualization. Hypervisor-based and OS-level virtualization technologies along with concrete implementations are presented. Concepts for the orchestration of virtualized infrastructures and particular concepts from our current and past research form the end.

7) Network Function Virtualization: The seventh chapter introduces the specifics of telecommunication provider networks as motivation for the emergence of NFV. The main part of the lecture covers the ETSI NFV architecture with all relevant details and discusses use cases. OPNFV is introduced as exemplary implementation of a NFV environment.

B. Practical Assignments

The second part of the course focuses on practical experiences. Students are required to meet a total score of 60% in two assignments to get an admission for the final exam. Scores above 60% are transformed into at most 10% bonus for the final exam grading. Teamwork skills were encouraged while the grading effort is limited by the requirement to work in groups of two. Sessions for the practical assignments were part of the weekly course schedule, i.e., two lecture slots within the semester were reserved. We published one assignment sheet at the beginning of the course, another in the middle that needs to be finished within a time period of four weeks. It contains two parts, a short group test and programming tasks.

1) Group Test: The assignment sheets include a catalogue of 15 to 20 questions that are either related to contents of the lecture or to aspects of the assignment. Each student is required to give answers to five questions of the catalogue a week after the assignment sheet was handed out. The group test verifies that both group members dealt with the related theoretical questions prior to programming task work.

2) Programming Tasks: In contrast to other courses, we limited the programming tasks on SDN application implementation for Ryu. Therefore, the students learn to implement known networking concepts, e.g., L2 switching or IP routing, using the capabilities of Ryu and OpenFlow-based SDN.

The first programming task focuses on familiarization with Mininet and Ryu by extending a L2 switch sample application to support port-based segmentation. The second programming assignment extends the L2 switch implementation with IPv4 routing. Four hosts residing in two different IP subnets are attached to a SDN switch which routes packets between the subnets. Figure 1 shows the topology for the third programming task. Eight hosts are conntected to two SDN switches that act as IPv4 edge routers. Two additional SDN switches acting as core routers represent the Internet between both networks. The edge and core routers are related to two different SDN controllers. The fourth programming task furthermore extends the routing functionality by introducing longest prefix matching in a more complex routing scenario and routing of IPv6 packets. In the last programming task, students implement packet- and flow-based IP anycast.

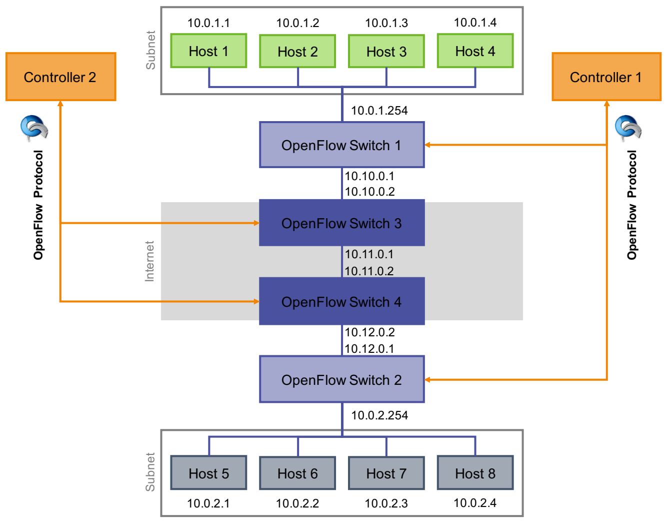


Fig. 1: Network topology for an IPv4 routing scenario with 8 hosts, 4 OpenFlow switches, and 2 SDN controllers.

To simplify the work on the programming assignments, we provided the NetSoft-VM, a virtual machine image as uniform programming and execution environment. We chose VirtualBox as virtualization platform as it can be installed on Windows, macOS, and Linux. The NetSoft-VM is based on Ubuntu 17.04 with XFCE and contains Mininet, MiniEdit as GUI for Mininet, Python 3, the Ryu SDN controller framework, and the Atom editor. Figure 2 shows the NetSoft-VM running in VirtualBox with MiniEdit as GUI for the Mininet network simulator.

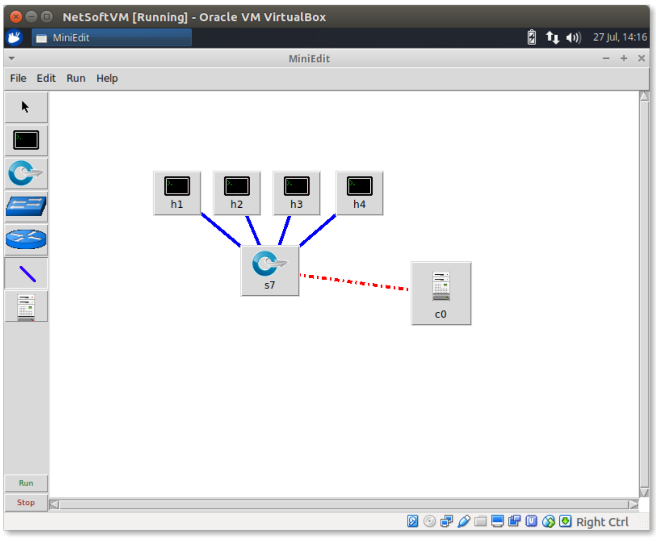


Fig. 2: Netsoft VM in VirtualBox with MiniEdit as GUI for the Mininet network simulation environment.

Successfully passing the group test was set as requirement for the grading of the programming tasks. The grading focused on source code functionality and quality.

C. *Final Exam*

Due to a limited number of participating students, we decided not to offer a written final exam. Instead, we chose the format of an oral exam with 25 minutes per student.

References

[1] M. Menth, M. Schmidt, D. Reutter, R. Finze, S. Neuner, and T. Kleefass, “Resilient Integration of Distributed High-Performance Zones into the BelWue Network Using OpenFlow,” *IEEE Communications Magazine (Commag)*, vol. 55, no. 4, Apr. 2017.
[2] F. Hauser, M. Schmidt, and M. Menth, “Establishing a Session Database for SDN Using 802.1X and Multiple Authentication Resources,” in *IEEE International Conference on Communications (ICC)*, Paris, France, Jun. 2017.
[3] M. Schmidt, R. Finze, D. Reutter, and M. Menth, “Demo: Resilient Integration of Distributed High-Performance Zones into the BelWue Network Using OpenFlow,” in *International Teletraffic Congress (ITC)*, Wuerzburg, Germany, Sep. 2016.
[4] “Open Networking Foundation: SDN Reading List,” https://www.opennetworking.org/sdn-resources/sdn-reading-list, accessed: 2017-08-06.
[5] “Mininet Walkthrough,” http://mininet.org/walkthrough/, accessed: 2017-08-06.
[6] “Ryu Documentation: Getting Started,” http://ryu.readthedocs.io/en/latest/getting_started.html, accessed: 2017-08-06.
[7] “Coursera: Software Defined Networking,” https://www.coursera.org/learn/sdn, accessed: 2017-08-06.
[8] “Princeton University: COS-597E, Fall 2013: Software Defined Networking,” http://www.cs.princeton.edu/courses/archive/fall13/cos597E/index.html, accessed: 2017-08-06.

3.3 On Moving Averages, Histograms and Time-Dependent Rates for Online Measurement

On Moving Averages, Histograms and Time-Dependent Rates for Online Measurement

Michael Menth and Frederik Hauser
Department of Computer Science
University of Tuebingen, Germany
{menth, frederik.hauser}@uni-tuebingen.de

ABSTRACT

Moving averages (MAs) are often used in adaptive systems to monitor the state during operation. Their output is used as input for control purposes. There are multiple methods with different ability, complexity, and parameters. We propose a framework for the definition of MAs and develop performance criteria, e.g., the concept of memory, that allow to parameterize different methods in a comparable way. Moreover, we identify deficiencies of frequently used methods and propose corrections. We extend MAs to moving histograms which facilitate the approximation of time-dependent quantiles. We further extend the framework to rate measurement, discuss various approaches, and propose a novel method which reveals excellent properties. The proposed concepts help to visualize time-dependent data and to simplify design, parametrization, and evaluation of technical control systems.

1. INTRODUCTION

Moving averages (MAs), moving histograms, and time-dependent rates calculate time-dependent statistics from time series while giving more importance to recent than to old samples. The exponential MA (EMA) is a simple example:

$$A_i = a \cdot A_{i-1} + (1-a) \cdot X_i. \qquad (1)$$

X_i is the size of an observed sample at time i and A_i is the time-dependent average at that time. The smoothing parameter a is often set to $a = 0.9$, but there is only little insight in literature about appropriate configuration.

On the one hand, the above mentioned methods are helpful to track and visualize the behavior of technical systems over time. On the other hand, they are used for control purposes by adaptive systems to observe their state and react appropriately. A prominent example is TCP [1] which estimates the current roundtrip time (RTT) by exponential smoothing of individual RTT samples. As another example, we currently develop an automatic bypass for a rate-constraint firewall using software-defined networking (SDN). The firewall is attached to a border switch which steers all in- and outgoing traffic through the firewall. Traffic leaving the firewall is monitored by exporting every n^{th} packet to the SDN controller using sFlow. The controller calculates a time-dependent rate over a relatively short time scale to detect congestion when a rate threshold is exceeded. Then, the controller bypasses sampled flows around the firewall through installation of appropriate flow rules on the switch. As the switch can support only a limited number of rules, the installation rate may be high if congestion occurs rather seldom, and should be low otherwise. To that end, we track congestion using a MA over a relatively long time scale and use its value for computation of a suitable offloading rate.

In this paper, we consider multiple MA methods with different parameters. We define some metrics for MAs that relate to time scale. The memory is most important and helps to set the parameters of the different methods such that their output behaves similarly. MAs are mostly applied under the assumption that samples are equidistantly spaced in time (evenly spaced time series). However, in practice unevenly spaced time series may also occur. Therefore, we adapt well-known MA methods such that the time between samples has influence on calculated average values. We show that EMA has a strong bias towards the first observed sample X_0 and propose a method for an unbiased EMA (UEMA). We demonstrate that EMA's existing extension for unevenly spaced time series (TEMA) is even persistently biased and suggest an unbiased TEMA (UTEMA). A challenge for the application of MAs is the tradeoff between accuracy and timeliness: MAs require a large memory to produce accurate estimates for mean values which is desired under the assumption of a stationary process. However, they need a short memory to quickly reveal changed behavior of a non-stationary process. We illustrate this tradeoff and give recommendations for appropriate parametrization. We propose moving histograms (MHs) as a straightforward extension of MAs including an efficient implementation. They allow approximation of time-dependent quantiles. Time-dependent rates should reflect the recent intensity of a sample process. We define time-dependent rate measurement (TDRM) as an extension of MAs. The time scale of TDRM can be controlled by the memory of the underlying MA. We review and compare existing TDRM methods and suggest TDRM-UTEMA as a novel method which exhibits desired properties.

The authors acknowledge the funding by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/2-1.

ICPE'17, April 22-26, 2017, L'Aquila, Italy
© 2017 ACM. ISBN 978-1-4503-4404-3/17/04…$15.00
DOI: http://dx.doi.org/10.1145/3030207.3030212

We believe that this rather elementary work contributes to a more informed usage of MAs, MHs, and TDRM. It focuses on online measurement, i.e., only the past of the process is known at measurement time. In contrast, offline measurement can consider a complete time series as input and smooth values or determine rates by leveraging both past and future samples around an observation point. While this work considers time series in the time domain, there is a large body of related work for time series analysis in the frequency domain [2].

The paper is structured as follows. Section 2 shows that MAs are used in various fields with different applications. Section 3 provides a novel framework to define MAs, suggests novel metrics to characterize MA properties, in particular the memory, reviews well-known MA approaches, and proposes new methods. Section 4 studies the impact of memory on the accuracy and timeliness of UEMA. MAs are extended towards MHs in Section 5. An extension for TDRM is proposed in Section 6: existing methods and a novel one are presented and compared. Section 7 concludes this work.

2. RELATED WORK

Though exponential smoothing is a standard technique for scientific work, it is hard to track back its roots in scientific literature. Books about fundamental statistics like [3] present only unweighted (simple) and weighted moving averages. However, we did not find an overview of general MA concepts and their properties. MAs are known under a broad range of different terms, e.g., they are also referred to as smoothing or filtering methods. Eckner describes simple and exponential moving averages as well as rolling operators [4] for the purpose of smoothing. The work in [5] presents similar concepts for the specific requirements of unevenly spaced time series. Autoregressive MAs and variants are discussed to model stochastic processes for the purpose of forecasting which is different from our application which is the calculation of an average value for the current observation point.

MAs are often applied in networking. Three different modifications of EMA (low pass, gradient adaptive, and retrospective) for bandwidth estimation are presented in [6]. Conga [7] is a distributed congestion-aware load balancing mechanism for datacenters. It leverages a discounting rate estimator which is similar to EMA. CSFQ [8] aims at providing fair bandwidth allocation. Exponential averaging with variable weights is used to estimate flow arrival rates. The active queue management PIE [9] is designed to control latency and jitter in the Internet. Its departure rate estimation uses exponential smoothing. TCP's smoothed roundtrip time (SRTT) mechanism computes the retransmission timeout in data communication with exponential averaging [1]. In [10] we presented a rate measurement method based on exponential smoothing. It was leveraged by the authors in [11] for time-decaying Bloom filters. Furthermore, we introduced the concept of moving histograms in [12] to calculate time-dependent quantiles.

MAs also have application in other areas. A simple window-based moving average is used in [13] for detection of the end of the transient phase of a stochastic process. To that end, a symmetric moving window is applied as low-pass filter to a time series, extracting its long-term trend. Exponential smoothing is widely used in the context of operations research and financial analyses for trend forecasting (e.g. [14]). The Holt-Winters forecasting procedure [15] uses three degrees of smoothing to extract level, trend, and seasonal components in time series. In quality control, exponential smoothing with optimized weights is used for the generation of control charts which detect exceedance or shortfall of critical boundaries [16, 17].

3. MOVING AVERAGES (MA)

We consider MAs for samples observed with evenly and unevenly spaced time series. For both cases, we propose a general definition of MAs. We introduce several performance metrics to characterize properties of MAs. We consider specific MAs and express their properties depending on their parameters. We point out differences among presented MAs, show that widely used exponential moving averages for evenly and unevenly spaced time series have an initial or even persistent bias, and propose unbiased variants.

3.1 MAs for Evenly Spaced Time Series

Let $(X_i)_{0\leq i<\infty}$ be an evenly spaced time series with samples of size X_i and Δt time between consecutive samples (inter-sample time). We define a MA A_j for observation point j by

$$S_j = \sum_{0\leq i\leq j} g_i(i-j)\cdot X_i \tag{2}$$

$$N_j = \sum_{0\leq i\leq j} g_i(i-j) \tag{3}$$

$$A_j = \begin{cases} \frac{S_j}{N_j} & N_j > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

The sample-specific discrete weight functions $g_i(.)$ are characteristic for special types of MAs. They are used to compute a weighted sample sum S_j and weighted sample number N_j. Thereby, the weight functions $g_i(.)$ may reduce the impact of samples X_i on the MA that are distant from the observation point j. For most MA types, the weight $g_i(0)$ for the most recent sample takes the maximum $g_i^{max} = \max_k(g_i(k))$. The function decreases monotonously towards negative and positive arguments whereby only negative arguments are considered for online measurement. The fraction of the weighted sample sum and the weighted sample number yields the MA A_j if the number N_j is positive. Otherwise, we define the MA to be zero.

3.1.1 Metrics

If the weight functions $g_i(.)$ for samples X_i do not differ, their subscript as well as those of the following metrics may be omitted.

The contribution C_i quantifies how much a sample X_i contributes to all average values A_j. It can be calculated by

$$C_i = \sum_{-\infty<k\leq 0} g_i(k)\cdot\Delta t. \tag{5}$$

If contributions C_i differ, the MA has a bias to towards samples with larger contributions.

The memory M_i quantifies the average duration over which a sample X_i contributes to average values A_j. It considers fractional contributions relative to the maximum sample weight g_i^{max} and can be derived as

$$M_i = \frac{C_i}{g_i^{max}}. \tag{6}$$

Essentially, the memory reflects the time scale over which samples are averaged. If weight functions $g_i(.)$ differ among samples, they may – but do not need to – yield different contributions C_i and memory M_i.

The memory depends on the inter-sample time Δt which must be taken into account when applying MAs in practice. A MA may be used to track the transmission delay of packets on a communication line. Transmission of packets with 1500 bytes yields inter-sample times of $\Delta t = 12$ ms and $\Delta t = 0.012$ ms on a 1 Mb/s and 1 Gb/s link, respectively. Irrespective of the specific type of MA, the resulting memory differs by three orders of magnitude if MAs are applied with identical parametrization. If the timely dynamics of the measured averages should be comparable, the parameters of the MAs need to be adapted to the specific inter-sample time Δt so that the MAs exhibit the same memory for all considered processes.

The delay D_i quantifies the average age of all contributions of a sample X_i to all average values A_j. It is computed by

$$D_i = \sum_{-\infty<k\leq 0} (|k| \cdot \Delta t) \cdot \frac{(g_i(k) \cdot \Delta t)}{C_i}. \tag{7}$$

Shifting a MA's weight function by j to $g_i^*(k) = g_i(k+j)$ for $k \leq -j$ and $g_i^*(k) = 0$ for $k > -j$ leads to another MA with the same memory but a larger delay.

3.1.2 Cumulative Mean (CumMean)

CumMean is defined by

$$A_j = \frac{1}{j+1} \cdot \sum_{0\leq i\leq j} X_i. \tag{8}$$

It fits the Definitions (2) – (4) for homogeneous weights $g(k) = 1$, $-\infty < k \leq 0$. Thus, each sample exhibits a contribution, delay, and memory of $M = D = C = \infty \cdot \Delta t$. Figure 1(a) visualizes the evolution of the CumMean for the evenly spaced time series $(X_0, ..., X_{11}) = (1,1,0,1,1,1,0,1,0,0,0,0)$. The impact of recent samples diminishes with increasing time index because all past samples equally contribute to CumMean's average. Therefore, CumMean does not focus on the recent evolution of the observed process and cannot provide appropriate feedback for self-adapting systems.

3.1.3 Window Moving Average (WMA)

WMA, also known as simple moving average (SMA) [4], essentially computes the arithmetic mean of the last w samples whereby w is an integer. The weighted sample sum is

$$S_j = \sum_{max(0,j-w+1)\leq i\leq j} X_i \tag{9}$$

and the weighted sample number is $N_j = min(w, j+1)$. This fits the Definitions (2) – (4) for weights

$$g(k) = \begin{cases} 1 & -w < k \leq 0 \\ 0 & \text{otherwise} \end{cases}. \tag{10}$$

Contribution and memory are $C = M = w \cdot \Delta t$ and delay is $D = \frac{(w-1)\cdot\Delta t}{2} = \frac{M}{2} - \frac{\Delta t}{2}$.

Figure 1(a) visualizes the evolution of WMA-based averages for the above introduced sample series. It yields an average value of $A_{11} = 0$ because it disregards past samples outside its window. Furthermore, it yields $A_5 = 0.75$ and $A_6 = 0.75$ although the observed '1' are younger in A_5 than in A_6. This is due to the fact that all samples within a WMA's window are equally weighted.

3.1.4 Disjoint Windows Moving Average (DWMA)

DWMA partitions a time series into sets of consecutive and disjoint windows $\mathcal{W}_k = \{k \cdot w \leq i < (k+1) \cdot w\}$ with w time indices each. At the end of such a window, the arithmetic mean of the samples falling in this window is taken as current average until the next measurement value is available. This can be denoted by

$$S_j = \begin{cases} 0 & j < w-1 \\ \sum_{i\in\mathcal{W}_{(\lfloor\frac{j+1}{w}\rfloor-1)}} X_i & \text{otherwise} \end{cases} \tag{11}$$

and $N_j = w$. It fits the Definitions (2) – (4) for sample-specific weights

$$g_i(k) = \begin{cases} 1 & \begin{cases} (i \mod w) - (2\cdot w - 1) \\ < k \leq \\ (i \mod w) - (w-1) \end{cases} \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

Thus, contribution and memory are $C = M = w \cdot \Delta t$. The sample-specific delay is $D_i = \frac{w-1}{2} \cdot \Delta t + ((w-1) - (i \mod w)) \cdot \Delta t$ and the average delay is $\overline{D} = (w-1) \cdot \Delta t = M - \Delta t$.

Figure 1(a) shows the evolution of DWMA. For the first $w-1$ observation points, there is no actual average value available. Due to its increased delay, DWMA clearly lags behind WMA and it is coarser.

3.1.5 Unbiased Exponential Moving Average (UEMA)

We propose UEMA as a novel algorithm using the weight function

$$g(k) = a^{-k} \tag{13}$$

and the Definitions (2) – (4) for the computation of the average values. The underlying geometric model enables an elegant, recursive calculation:

$$S_j = \begin{cases} X_j & j = 0 \\ a \cdot S_{j-1} + X_j & j > 0 \end{cases} \tag{14}$$

$$N_j = \begin{cases} 1 & j = 0 \\ a \cdot N_{j-1} + 1 & j > 0 \end{cases}. \tag{15}$$

UEMA has a contribution and memory of $C = M = \frac{\Delta t}{1-a}$, and a delay of $D = \frac{a\cdot\Delta t}{1-a}$ for all samples, i.e., $D = a \cdot M$. Figure 1(a) compares the evolution of UEMA with the one of WMA for the same memory $M = 4 \cdot \Delta t$. While WMA disregards samples that lie outside its window and yields $A_{11} = 0$, UEMA respects the full past of the process and yields $A_{11} = 0.21$. WMA also disregards the position of samples within its window. In contrast, UEMA is sensitive to that and yields $A_5 = 0.87$ (observed 0 in WMA's window is old) and $A_6 = 0.62$ (observed 0 in WMA's window is young).

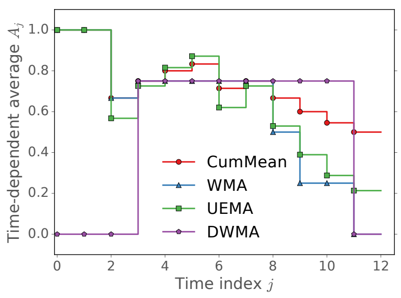
Figure 1(b) illustrates the impact of UEMA's memory on the evolution of the MA. The MA is more inert for larger memory, i.e., for a larger smoothing factor a.

3.1.6 Exponential Moving Average (EMA)

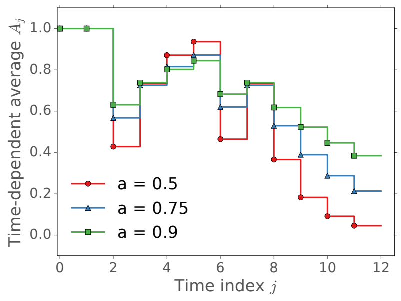
EMA, also known as Exponentially Weighted MA (EWMA) or exponential smoothing, calculates its weighted sum S_j as

$$S_j = \begin{cases} X_0 & j = 0 \\ a \cdot S_{i-1} + (1-a) \cdot X_j & j > 0 \end{cases} \tag{16}$$

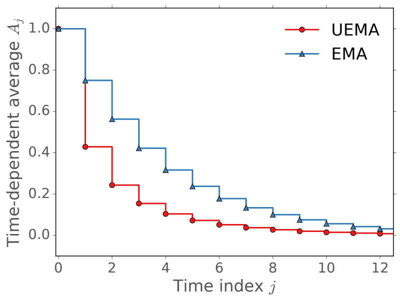
and the weighted number of samples is $N_j = 1$. Thus, S_j already computes A_j (see Equation (1)). These formulae fit



(a) CumMean $(M = \infty)$, WMA, DWMA (both with $w = 4$, $M = 4 \cdot \Delta t$), and UEMA $(a = \frac{3}{4}, M = 4 \cdot \Delta t)$ for time series $(X_0, ..., X_{11}) = (1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0)$.



(b) Comparison of UEMA for $a \in \{\frac{1}{2}, \frac{3}{4}, \frac{9}{10}\}$ $(M \in \{2 \cdot \Delta t, 4 \cdot \Delta t, 10 \cdot \Delta t\})$ for time series $(X_0, ..., X_{11}) = (1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0)$.



(c) EMA and UEMA with $a = 0.75$ $(M = 4 \cdot \Delta t)$ for time series $(X_0, ..., X_{11}) = (1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$: EMA has an initial bias.

Figure 1: Timely evolution of MAs for evenly spaced time series.

the Definitions (2) – (4) for sample-specific weights

$$g_i(k) = \begin{cases} a^{-k} & i = 0 \\ (1-a) \cdot a^{-k} & \text{otherwise} \end{cases}. \tag{17}$$

The two different weight functions yield the same memory $M = \frac{\Delta t}{1-a}$ and delay of $D = \frac{a \cdot \Delta t}{1-a}$ which equal those of UEMA. However, the contribution of X_0 is $C_0 = \frac{\Delta t}{1-a}$ while the one of all other samples is $C_i = \Delta t$. This causes a bias towards X_0 in the time series of the resulting MA A_j. To illustrate this effect, Figure 1(c) compares the resulting averages for EMA and UEMA for consecutive samples $(X_0, ..., X_{11}) = (1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$. EMA and UEMA are both configured with $a = 0.75$, i.e., $M = 4 \cdot \Delta t$. The difference between the two curves is the bias induced by EMA's larger contribution of X_0. As the bias vanishes over time, EMA may be used if accuracy for initial values A_j does not matter. This may be advantageous as EMA is slightly simpler than UEMA. While EMA is already applied in many technical systems and research papers, UEMA is a novel method proposed in this work.

3.2 MAs for Unevenly Spaced Time Series

Let $(X_i)_{t_i \in \mathcal{T}, 0 \leq i < \infty}$ be an unevenly spaced time series of samples with different size X_i. Figure 2(a) contrasts two examples to an evenly spaced time series.

MAs for unevenly spaced time series respect the time structure of samples such that the impact of a sample on resulting average values diminishes over time instead with progressing time index. We define a MA A_t for observation point t by

$$S_t = \sum_{\{i:t_i \leq t\}} g_i(t_i - t) \cdot X_i \tag{18}$$

$$N_t = \sum_{\{i:t_i \leq t\}} g_i(t_i - t) \tag{19}$$

$$A_t = \begin{cases} \frac{S_t}{N_t} & N_t > 0 \\ 0 & \text{otherwise} \end{cases}. \tag{20}$$

Definitions (18) – (20) are analogous to Definitions (2) – (4) but account for the fact that sample times are now continuous. Therefore, the sample-specific weight functions $g_i(.)$ are also continuous.

3.2.1 *Metrics*

The metrics are also analogous to those of MAs for evenly spaced time series. The contribution C_i of sample X_i can be calculated as

$$C_i = \int_{-\infty}^{0} g_i(t)dt \tag{21}$$

and the memory is

$$M_i = \frac{C_i}{g_i^{max}} \tag{22}$$

with $g_i^{max} = \max_{-\infty < t \leq 0}(g_i(t))$. The delay is

$$D_i = \frac{\int_{-\infty}^{0} |t| \cdot g(t)dt}{C_i}. \tag{23}$$

3.2.2 *Time Window Moving Average (TWMA)*

TWMA computes the average of the samples within a recent time window of duration W. Let $\mathcal{W}_t = \{i : t_i \in (t - W; t]\}$ be the index set of samples arriving within that window at time t. Average values for TWMA can be computed by $N_t = |\mathcal{W}_t|$ and

$$S_t = \begin{cases} 0 & |\mathcal{W}_t| = 0 \\ \sum_{i \in \mathcal{W}_t} X_i & \text{otherwise} \end{cases}. \tag{24}$$

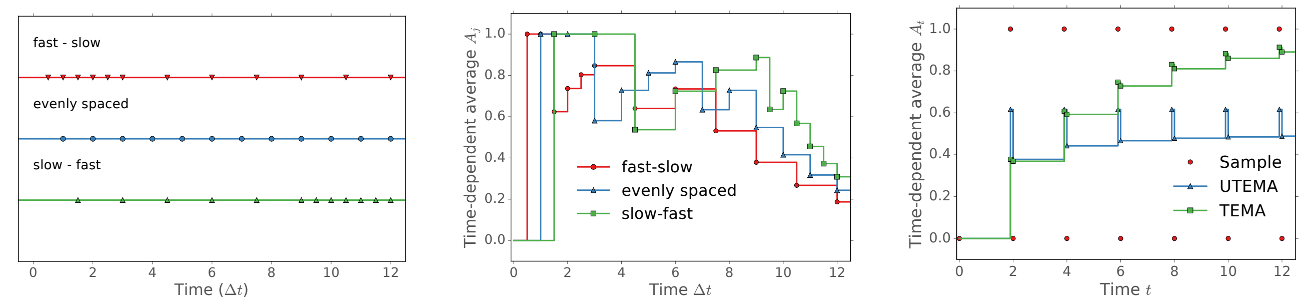
This fits the Definitions (18) – (20) for the weight function

$$g(t) = \begin{cases} 1 & -W < t \leq 0 \\ 0 & \text{otherwise} \end{cases}. \tag{25}$$

Contribution and memory are $C = M = W$ and the delay is $D = \frac{W}{2} = \frac{M}{2}$. The resulting MA A_t returns to zero if the last sample is older than W, which may be an undesired property. We omit illustrations of TWMA due to space limitations.

3.2.3 *Disjoint Time Windows Moving Average (DTWMA)*

DTWMA partitions the time axis into consecutive measurement intervals of duration W, computes the arithmetic mean of samples observed within a measurement interval at its end, and uses this average value until the next average value is available. Let $\mathcal{W}_k = \{i : t_i \in (k \cdot W; (k+1) \cdot W]\}$ be the index set of samples arriving within window number k since measurement start at time $t = 0$. Average values for DTWMA are computed by $N_t = |\mathcal{W}_{(\lceil \frac{t}{W} \rceil - 1)}|$ and



(a) Three considered arrival times: samples first arrive fast and then slowly, vice-versa, and at constant speed.

(b) UTEMA with $\beta = \frac{1}{4 \cdot \Delta t}$ ($M = 4 \cdot \Delta t$) for time series $(1,1,0,1,1,1,0,1,0,0,0,0)$ and the arrival times in Figure 2(a).

(c) TEMA and UTEMA with $\beta = \frac{1}{\Delta t}$ ($M = 1 \cdot \Delta t$): TEMA exhibits an initial and a persistent bias. Sample arrivals and sizes are represented by dots.

Figure 2: Timely evolution of MAs for unevenly spaced time series.

$$S_t = \begin{cases} 0 & t < W \vee |\mathcal{W}_{(\lceil \frac{t}{W} \rceil - 1)}| = 0 \\ \sum_{i \in \mathcal{W}_{(\lceil \frac{t}{W} \rceil - 1)}} X_i & \text{otherwise} \end{cases}. \quad (26)$$

These equations fit Definitions (18) – (20) for sample-specific weight functions

$$g_i(t) = \begin{cases} 1 & \begin{cases} t_i - (\lceil \frac{t_j}{W} \rceil + 1) \cdot W \\ < t \leq \\ t_i - \lceil \frac{t_j}{W} \rceil \cdot W \end{cases} \\ 0 & \text{otherwise} \end{cases}. \quad (27)$$

Thus, contribution and memory are $C_i = M_i = W$. The sample-specific delay is $D_i = (\lceil \frac{t_i}{W} \rceil \cdot W - t_i) + \frac{W}{2}$ and the average delay is $\overline{D} = W$. The MA A_t returns to zero after a measurement interval without any samples. We omit illustrations of DTWMA due to space limitations.

3.2.4 *Unbiased Time-Exponential Moving Average (UTEMA)*

We propose UTEMA as a novel MA method which uses the exponential weight function

$$g(t) = e^{\beta \cdot t} \quad (28)$$

in Definitions (18) – (20). The underlying exponential model allows for recursive equations:

$$S_t = \begin{cases} 0 & t < t_0 \\ X_0 & t = t_0 \\ e^{-\beta \cdot (t - t_{i-1})} \cdot S_{t_{i-1}} + X_i & t = t_i \\ e^{-\beta \cdot (t - t_i)} \cdot S_{t_i} & t_i < t < t_{i+1} \end{cases} \quad (29)$$

$$N_t = \begin{cases} 0 & t < t_0 \\ 1 & t = t_0 \\ e^{-\beta \cdot (t - t_{i-1})} \cdot N_{t_{i-1}} + 1 & t = t_i \\ e^{-\beta \cdot (t - t_i)} \cdot N_{t_i} & t_i < t < t_{i+1} \end{cases}. \quad (30)$$

UTEMA has a contribution, memory, and delay of $C = M = D = \frac{1}{\beta}$.

Figure 2(b) shows that UTEMA respects the time structure of the sample processes given in Figure 2(a). If samples first arrive fast and then slowly, UTEMA yields a lower average $A_{t=12 \cdot \Delta t}$ at the end of the observation interval than for the evenly spaced time series because observed '1' are older. Likewise, if samples first arrive slowly and then fast, UTEMA leads to a larger $A_{t=12 \cdot \Delta t}$ than for the evenly spaced time series because observed '1' are younger. When UEMA is applied for one of the unevenly spaced time series, it yields UTEMA's average values for the evenly spaced time series.

UEMA can provide exactly the same average values for an evenly spaced time series as UTEMA if its smoothing factor is set such that it yields the same weights for integral multiples of Δt, i.e., $a = e^{-\beta \cdot \Delta t}$. Figure 3 compares such weight functions of UTEMA and UEMA with a smoothing rate of $\beta = \frac{0.25}{\Delta t}$ and smoothing factor of $a = 0.7788$, respectively. The memory for UTEMA is $M = 4 \cdot \Delta t$ while the one for UEMA is $M = 4.52 \cdot \Delta t$. The discrepancy is due to the fact that UTEMA's weight function $g_{UTEMA}(t)$ interpolates only the lower corners of UEMA's weights $g_{UEMA}(k)$ and, therefore, leads to a lower integral value. This difference converges to $\frac{\Delta t}{2}$ for large memory M.

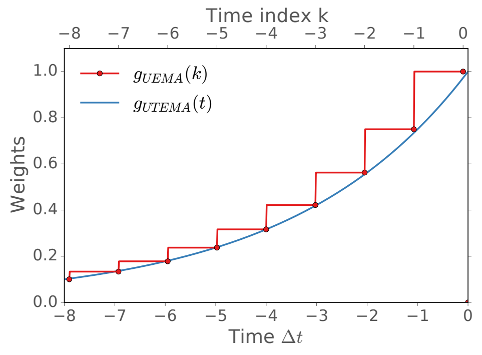


Figure 3: Weight functions for UEMA and UTEMA configured such that they reveal equal weights for integral multiples of Δt ($a = 0.7788$ and $\beta = \frac{0.25}{\Delta t}$, respectively). UTEMA yields a lower memory because its weight function $g_{UTEMA}(t)$ interpolates only the lower corners of UEMA's step function $g_{UEMA}(k)$.

The difference between UTEMA's average curves for evenly and unevenly spaced time series quantifies the error caused by UEMA. Its practical significance depends on the time scale of interest because the difference decreases for increasing memory M. If the time structure of the observed process is essential, UTEMA is a good alternative to UEMA and also to TWMA which does not respect the time struc-

Table 1: Considered MAs including properties.

Method	Param.	Contribution C	Memory M	Delay D	Delay D dep. on M	Comments
MAs for evenly spaced time series						
CumMean	Δt	∞	∞	∞	∞	Does not discount old samples.
WMA	$w, \Delta t$	$w \cdot \Delta t$	$w \cdot \Delta t$	$\frac{(w-1)\cdot\Delta t}{2}$	$\frac{M}{2} - \frac{\Delta t}{2}$	Does not account for samples outside window and sample position within window.
DWMA	$w, \Delta t$	$w \cdot \Delta t$	$w \cdot \Delta t$	$(w-1)\cdot\Delta t$ (avg.)	$M - \Delta t$ (avg.)	Like WMA, additional delay
UEMA	$a, \Delta t$	$\frac{\Delta t}{1-a}$	$\frac{\Delta t}{1-a}$	$\frac{a\cdot\Delta t}{1-a}$	$a \cdot M$	Method proposed in this work
EMA	$a, \Delta t$	$\frac{\Delta t}{1-a}$, Δt	$\frac{\Delta t}{1-a}$	$\frac{a\cdot\Delta t}{1-a}$	$a \cdot M$	Bias towards X_0
MAs for unevenly spaced time series						
TWMA	W	W	W	$\frac{W}{2}$	$\frac{M}{2}$	May return to zero (invalid value).
DTWMA	W	W	W	W (avg.)	M (avg.)	Like TWMA, additional delay
UTEMA	β	$\frac{1}{\beta}$	$\frac{1}{\beta}$	$\frac{1}{\beta}$	M	Method proposed in this work
TEMA	β	$\frac{1}{\beta}$ for X_0, $\frac{1-e^{-\beta\cdot(t_i-t_{i-1})}}{\beta}$ for X_i	$\frac{1}{\beta}$	$\frac{1}{\beta}$	M	Bias towards X_0 and other samples

ture within its measurement window and may return to zero in the absence of sufficiently young samples.

For UTEMA, the half-life time may be used as another metric. It can be computed by $H = \frac{\ln(2)}{\beta}$, but it is not applicable to most other MA variants.

3.2.5 *Time-Exponential Moving Average (TEMA)*

TEMA is sometimes used as adaptation of EMA to unevenly spaced time series [4]. Its sample sum is computed by

$$S_t = \begin{cases} 0 & t < t_0 \\ X_0 & t = t_0 \\ e^{-\beta\cdot(t_i-t_{i-1})}\cdot S_{t_{i-1}} + & t_i \leq t < t_{i+1} \\ (1-e^{-\beta\cdot(t_i-t_{i-1})})\cdot X_i & \end{cases}. \quad (31)$$

and its weighted number of samples is $N_j = 1$. Thus, TEMA is hardly simpler than UTEMA since it also requires the calculation of exponential functions. The equations fit the Definitions (18) – (20) for the sample-specific weight functions

$$g_i(t) = \begin{cases} e^{\beta\cdot t} & i = 0 \\ (1-e^{-\beta\cdot(t_i-t_{i-1})})\cdot e^{\beta\cdot t} & i > 0 \end{cases}. \quad (32)$$

While EMA has only two different sample-specific weight functions, those of TEMA may all be different. The contributions are also sample-specific:

$$C_i = \begin{cases} \frac{1}{\beta} & i = 0 \\ \frac{1-e^{-\beta\cdot(t_i-t_{i-1})}}{\beta} & i > 0 \end{cases}. \quad (33)$$

Nevertheless, the weight functions of all samples reveal the same memory $M = \frac{1}{\beta}$ which equals the one of UTEMA. TEMA's bias towards some samples is more severe than the one of EMA because it does not vanish over time. Its impact is illustrated in Figure 2(c). Sample sizes are 0 after a short inter-sample time of $0.1 \cdot \Delta t$ and 1 after a long inter-sample time of $1.9 \cdot \Delta t$. While UTEMA yields average values converge to 0.5 most of the time, TEMA's average values continuously increase as large samples have a larger contribution than small samples in this process. We also simulated a Poisson arrival process over $10^6 \cdot \Delta t$ time with rate $\lambda = \frac{1}{\Delta t}$ and set the sample size X_i to the value of the preceding inter-arrival time divided by Δt. We averaged the obtained time-dependent average values over time. UTEMA yields 1.11, 1.05, and 1.02 for $M \in \{\Delta t, 4\cdot\Delta t, 10\cdot\Delta t, 25\cdot\Delta t\}$ while corresponding values for TEMA are 1.80, 1.91, and 1.96. Thus, the bias is significant and even increases with larger memory.

3.3 Summary

Table 1 summarizes properties of considered MAs.

4. ANALYSIS OF UEMA

We illustrate the impact of UEMA's memory on the accuracy and timeliness of obtained averages. Accuracy addresses the deviation of UEMA's computed average from the true mean μ of an observed stationary sample process $(X_i)_{0\leq i<\infty}$. Timeliness addresses UEMA's ability to early reflect changes regarding μ in the observed process.

Table 2: Averaged squared deviation of computed averages A_i from the known sample mean μ.

	σ^2					
$M(\Delta t)$	1	3	10	30	100	300
3	0.1999	0.5997	1.9990	5.9969	19.9896	59.9687
10	0.0527	0.1580	0.5266	1.5798	5.2655	15.7966
30	0.0170	0.0511	0.1734	0.5112	1.7039	5.1116
100	0.0051	0.0152	0.0508	0.1525	0.5082	1.5246
300	0.0017	0.0051	0.0171	0.0514	0.1712	0.5136
1000	0.0005	0.0016	0.0053	0.0158	0.0527	0.1580
CumMean	0.0000	0.0000	0.0001	0.0004	0.0012	0.0036

4.1 Impact of Memory on Accuracy

We generate $n = 10^6$ normally distributed random variables X_i with mean $\mu = 0$ and variance $\sigma^2 \in \{1, 3, 10, 30, 100, 300\}$ and track them by UEMA with a memory of $M \in \{3, 10, 30, 100, 300, 1000\} \cdot \Delta t$. Table 2 shows the arithmetic mean of the squared deviations of the averages from the observed random variable's mean μ: $dev^2(n) = \frac{1}{n} \cdot \sum_{0\leq i<n}(A_i - \mu)^2$. The accuracy of the estimate depends on the variance of the observed process and UEMA's memory. It can be explained as follows. According to Equations (2) – (4) and (13), UEMA's average value A_j is the sum S_j of weighted, independent random variables multiplied by the scalar value $\frac{1}{N_j}$. Therefore, the variance of A_j can be calculated as

$$VAR[A] = \frac{\sum_{0 \leq i < \infty} a^{2 \cdot i} \cdot \sigma^2}{(\sum_{0 \leq i < \infty} a^i)^2} = \frac{(1-a)^2}{1-a^2} \cdot \sigma^2 = \frac{\sigma^2}{2 \cdot \frac{M}{\Delta t} - 1} \tag{34}$$

which explains the observation in our experiment.

Equation (34) helps to find a minimum smoothing factor a if the variance σ^2 of the samples is roughly known. We assume that the averaged values A_j are distributed according to a normal distribution. With $E[A] = \mu$ and $VAR[A] = \frac{1-a}{1+a} \cdot \sigma^2$ we define $Y = \frac{A-\mu}{\sqrt{\frac{1-a}{1+a} \cdot \sigma^2}}$ which is distributed according to a standard normal distribution[2]. Therefore, we get

$$P(-z_{1-\frac{\alpha}{2}} \leq Y \leq z_{1-\frac{\alpha}{2}}) = 1 - \alpha \tag{35}$$

$$P(\mu - \delta(a,\alpha) \leq A \leq \mu + \delta(a,\alpha)) = 1 - \alpha \tag{36}$$

with $z_{1-\frac{\alpha}{2}}$ being the $(1-\frac{\alpha}{2})$-quantile of the standard normal distribution and $\delta(a,\alpha) = z_{1-\frac{\alpha}{2}} \cdot \sqrt{\frac{1-a}{1+a} \cdot \sigma^2}$. We conclude that A deviates at most δ from the true mean μ with probability $1-\alpha$. This derivation helps to understand the accuracy of UEMA depending on a, but cannot calculate the accuracy when random variables with unknown variance σ^2 are observed. A solution is to assume orders of magnitude for σ^2.

We use this derivation to choose a lower bound $a_{min}(\alpha, \varepsilon)$ for a such that a maximum error ε can be achieved with probability $1-\alpha$:

$$a_{min}(\alpha, \varepsilon) \geq \frac{1 - \left(\frac{\varepsilon}{\sigma \cdot z_{1-\frac{\alpha}{2}}}\right)^2}{1 + \left(\frac{\varepsilon}{\sigma \cdot z_{1-\frac{\alpha}{2}}}\right)^2}. \tag{37}$$

Table 3 illustrates the accuracy of that approach for 10^7 normally distributed random variables and compares it with a standard smoothing factor of $a = 0.9$.

Table 3: Percentage ρ of UEMA averages deviating at most $\varepsilon = 1$ from the actual mean $\mu = 0$ depending on smoothing factor a. Smoothing factor a_{min} is chosen for $\alpha = 0.1$ ($z_{1-\frac{\alpha}{2}} = 1.645$) according to Equation (37).

σ^2	1	3	10	30	100	300
$a_{min}(\alpha, \varepsilon)$	0.4579	0.7795	0.9283	0.9755	0.9926	0.9975
$\rho(a_{min}(\alpha, \varepsilon))$	0.8990	0.8989	0.8990	0.8902	0.8991	0.9001
$\rho(a = 0.9)$	1.0000	0.9881	0.8319	0.5740	0.3371	0.1987

Equation (37) helps to estimate probabilities with a certain accuracy. Probabilities p can be determined by counting $X = 1$ if a sample fulfills a certain condition, and $X = 0$ otherwise. This yields a Bernoulli process with variance $\sigma^2 = (1-p) \cdot p \leq 0.25$ which may be used to determine an appropriate smoothing factor. Thus, $a_{min} = 0.99971$ may be used to estimate probabilities with accuracy of $\varepsilon = 0.01$ and error probability of $\alpha = 0.1$.

4.2 Impact of Memory on Timeliness

To illustrate the ability of UEMA to reveal changed process behavior, we simulate $n+1 = 10001$ samples of normally distributed random variables X_i. The mean of X_i increases over time and is set to $E[X_i] = \frac{i}{n}$, $0 \leq i \leq n$ while the variance $\sigma^2 = 1$ remains stable. We performed this experiment $n_{runs} = 10000$ times to calculate average $\overline{A_i}$ and the empirical variance $S^2(A_i)$ of the average values A_i. Table 4 shows these values estimated after $i \in \{3333, 6666, 10000\}$ steps by UEMA for different memory and for CumMean.

Table 4: Samples with increasing expectations $E[X_i]$. Average ($\overline{A_i}$ and empirical variance $S^2(A_i)$ of average values A_i for UEMA with different memory and for CumMean (CM).

i	3333		6666		10000	
$M(\Delta t)$	$\overline{A_i}$	$S^2(A_i)$	$\overline{A_i}$	$S^2(A_i)$	$\overline{A_i}$	$S^2(A_i)$
1	0.3355	0.9987	0.6773	1.0051	0.9886	1.0128
3	0.3297	0.1983	0.6680	0.2021	0.9949	0.2022
10	0.3320	0.0533	0.6648	0.0530	0.9975	0.0521
30	0.3305	0.0174	0.6623	0.0169	0.9964	0.0166
100	0.3233	0.0052	0.6557	0.0051	0.9899	0.0049
300	0.3032	0.0017	0.6361	0.0017	0.9702	0.0016
1000	0.2458	0.0005	0.5672	0.0005	0.9901	0.0005
3000	0.1972	0.0003	0.4475	0.0002	0.7370	0.0002
10000	0.1762	0.0003	0.3701	0.0002	0.5820	0.0001
CM	0.1670	0.0003	0.3334	0.0002	0.5001	0.0001
$E[X_i]$	0.3333	-	0.6666	-	1.0000	-

The table shows that the averaged averages $\overline{A_i}$ approximate the configured expectations $E[X_i]$ well for small memory and clearly underestimate them for larger memory. Best values are obtained in this specific experiment for a memory of at most $M = 100 \cdot \Delta t$. We also observe that the sample variance $S^2(A_i)$ depends on the memory M but not on the index i. The latter is due to the fact that all X_i have the same variance σ^2 so that the expectation of the sample variance can be approximated by $VAR[A_i] = \frac{1}{2 \cdot \frac{M}{\delta t} - 1} \cdot \sigma^2$ which is well approximated by the values in the table. As A_i computed with small memory, e.g., $M = 10 \cdot \Delta t$ or smaller, reveal a large variance, they often deviate from their average $\overline{A_i}$ and are only little reliable. Thus, there is a tradeoff between accuracy and timeliness that can be controlled by UEMA's memory.

To guarantee timeliness of computed averages, the smoothing factor a must be low enough. The last m samples contribute an overall weight of $\sum_{0 \leq i < m} a^i = \frac{1-a^m}{1-a}$ to the average while the average contains an overall weight of at most $\sum_{0 \leq i < \infty} a^i = \frac{1}{1-a}$. To limit the influence of samples older than m time steps to a fraction γ, $\frac{1-a^m}{1-a} \geq \frac{1-\gamma}{1-a}$ must hold, i.e., $a \leq \sqrt[m]{\gamma}$ must be met. Conversely, for a given a, the impact of samples older than $\lceil \frac{\ln(\gamma)}{\ln(a)} \rceil$ time steps is limited to γ.

5. MOVING HISTOGRAMS (MH)

A histogram partitions the sample range into k intervals and associates with each of them a counter $bin(i)$, $0 \leq i < k$. We denote their lower and upper bound by $l(i)$ and $u(i)$. We define that the lower bound is part of the preceding interval and that the upper bound is part of the considered interval. Left- and rightmost intervals are extended towards $\pm\infty$. All bins are initialized with zero. At sample arrival, the corresponding bin of a cumulative histogram (CumHist) is incremented by 1. The relative frequency for samples in a certain interval i can be calculated by

$$h(i) = \frac{bin(i)}{\sum_{0 \leq j < k} bin(j)}. \tag{38}$$

While other relative frequencies such as $h(X \leq x)$ can also be determined by MAs, histograms allow the approximation of quantiles. The p-quantile Q_p is the infimum of values x for which $P(X \leq x) \leq p$ holds. Histograms can approximate it by

$$\widehat{Q_p} = u(i) : \sum_{0 \leq j < i} h(j) < p \leq \sum_{0 \leq j \leq i} h(j). \tag{39}$$

[2]The approach taken is similar but not equal to the estimation of confidence intervals for the mean of independent samples. In our case, consecutive average values A_j are highly correlated so that the variance cannot be derived from samples. We solve that problem by assuming an appropriate value for σ^2.

We adapt the concept of MAs to histograms. Moving histograms (MHs) can provide time-dependent quantiles. We presented an application in [12]. In the following, we discuss MHs on the base of UEMA and UTEMA.

5.1 MHs for Evenly Spaced Time Series

We present the unbiased exponential moving histogram (UEMH) which extends UEMA. When a new sample arrives, all bins are devaluated by the smoothing factor a. Afterwards, the bin associated with the new sample is incremented by 1. Thereby, the contribution of older samples to the histogram decreases. Relative frequencies are determined according to Equation (38). A single bin corresponds to UEMA's S_j and the sum of all bins to UEMA's N_j. Properties like contribution C, memory M, and delay D of UEMA also apply.

This straightforward adaptation requires high multiplication effort. As an alternative, devaluation may be omitted and $\frac{1}{a^i}$ may be used as increment for sample X_i instead of 1. This causes numerical instability as $\frac{1}{a^i}$ rises exponentially. A compromise is to avoid that increments exceed a value η. To that end, we devaluate bins only after $n_{dev} = \lceil \frac{-\ln(\eta)}{\ln(a)} \rceil$ steps by a factor $\frac{1}{a^{n_{dev}}}$ and choose $a^{i \mod n_{dev}}$ as increment for X_i.

5.2 MHs for Unevenly Spaced Time Series

The unbiased time-exponential moving histogram (UTEMH) extends UTEMA. In contrast to UEMH, bins are devaluated by $e^{-\beta\cdot(t_i-t_{i-1})}$ instead of a when X_i arrives. Similarly, increments of size $e^{\beta\cdot(t_i-t_0)}$ may be used instead of $\frac{1}{a^i}$ to avoid devaluation of all bins. To avoid numerical problems, bins are devaluated after $t_{dev} = \frac{\ln(\eta)}{\beta}$ time by $e^{-\beta\cdot t_{dev}}$ and increments $e^{\beta\cdot((t_i-t_0) \mod t_{dev})}$ are chosen.

Table 5: 10%-quantiles of UEMH with different memory and cumulative histogram (CH) for samples with increasing expectations $E[X_i]$.

i	3333		6666		10000	
$M(\Delta t)$	$\overline{Q_{10\%,i}}$	$\widehat{Q_{10\%,i}}$	$\overline{Q_{10\%,i}}$	$\widehat{Q_{10\%,i}}$	$\overline{Q_{10\%,i}}$	$\widehat{Q_{10\%,i}}$
1	-0.27	-0.89	-0.06	-0.58	+0.18	-0.36
3	-0.75	-0.92	-0.43	-0.55	-0.09	-0.29
10	-0.89	-0.92	-0.53	-0.55	-0.25	-0.26
30	-0.90	-0.92	-0.57	-0.57	-0.29	-0.28
100	-0.93	-0.94	-0.60	-0.60	-0.29	-0.29
300	-0.97	-0.97	-0.63	-0.63	-0.31	-0.31
1000	-1.03	-1.03	-0.71	-0.71	-0.38	-0.39
3000	-1.08	-1.08	-0.85	-0.85	-0.57	-0.58
10000	-1.11	-1.11	-0.93	-0.39	-0.75	-0.75
CH	-1.12	-1.11	-0.97	-0.97	-0.83	-0.83
$Q_{10\%,i}$	-0.949		-0.615		-0.282	

5.3 Timeliness of UEMH and CumHist

We illustrate the timeliness of UEMH with different memory M and cumulative histograms. We perform the same experiment as in Section 4.2 and set up histograms with equal-size ranges between -3 and 1 of width 0.1. Table 5 shows estimates of 10%-quantiles after i samples. They are computed as average values $\overline{Q_{10\%,i}}$ of 10%-quantiles gained from $n_{runs} = 100$ runs or as 10%-quantiles $\widehat{Q_{10\%,i}}$ based on the aggregated histogram information after sample i from all the n_{runs} different runs which is closer to the true value of the 10% quantile due to reduced variance. However, we see that both methods yield very similar results for $M = 30\cdot\Delta t$ or larger. The analytical values are given on the bottom of the table. They are well approximated by both methods with memories between 30 and $300\cdot\Delta t$. The 10% quantiles derived from a single run significantly fluctuate, but the values for $\overline{Q_{10\%,i}}$ show that they yield the right values on average. The quantiles estimated with cumulative histograms increase only slowly over time and clearly underestimate the analytical values. Smaller memory for UEMH cannot hold enough data for sufficiently accurate calculation of quantiles. MHs with larger memory are too much influenced by older samples which were generated with lower mean and cause lower estimates. Thus, there is also a tradeoff between timeliness and accuracy for MHs.

6. TIME-DEPENDENT RATE MEASUREMENT (TDRM)

A rate denotes an average number of samples per time, possibly weighted by their size. A time-dependent rate reflects mainly the recent past of the observed process. We present various techniques for TDRM and provide a comparison. Among the considered methods, TDRM-UTEMA is new and excels through timeliness, ease of configuration, and the fact that its measured rates are continuous with regard to configured memory.

6.1 TDRM Methods

We suggest a framework for the definition of TDRM methods and present five different instantiations.

6.1.1 A Framework for TDRM Methods

A time-dependent rate R_t at time t may be determined by

$$R_t = \begin{cases} \frac{S_t}{T_t} & T_t > 0 \\ 0 & \text{otherwise} \end{cases} \quad (40)$$

$$T_t = \int_0^t g(\tau - t)d\tau \quad (41)$$

where S_t is the weighted sample sum, taken from some MA method, and T_t is the weighted measurement interval which is computed analogously to S_t. Measured rates depend on a time scale which is the duration over which samples are considered for rate computation. In the presented definition the time scale is inherited from the memory M of the applied MA method. Also the concepts contribution C and delay D are inherited.

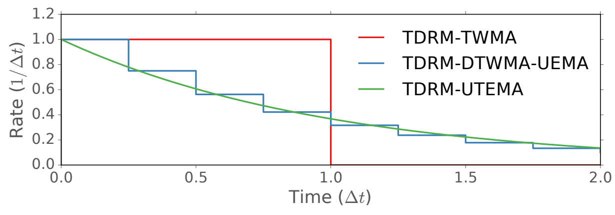


Figure 4: Rate impulses for TDRM-{TWMA, DTWMA-UEMA, UTEMA} with $t \to \infty$ used for calculation of T_t.

6.1.2 TDRM with Time Window Moving Average (TDRM-TWMA)

TDRM-TWMA calculates the sample sum S_t according to Equation (24). The corresponding weighted measurement interval is $T_t = \min(t, W)$ under the assumption that the measurement process starts at $t = 0$. The memory is

$M = W$ and the delay is $D = \frac{W}{2} = \frac{M}{2}$. As S_t may be zero, measured rates may be zero. TDRM-TWMA requires temporary storage of the samples within the measurement window and a timer indicating when the next sample leaves it. Therefore, the computation memory needed for TDRM-TWMA scales with the configured memory M and the rate of the process to be measured.

Figure 4 visualizes rate impulses for different TDRM methods. TDRM-TWMA generates a rate $\frac{1}{M}$ over a duration M for every observed sample of unit size. This rate can be considered as an impulse and the superposition of impulses from all measured samples (scaled by their size) yields the measured rate. The figure depicts rate impulses for various TDRM methods. They have the same memory M but contribute with a different time-dependent rate impulse to an overall measured rate. The impulses have an integral of 1 as $t \to \infty$ is assumed for computation of T_t. Superposition of such impulses ensures that the average of the overall measured rate approximates the sum of the observed sample sizes divided by the duration of the observation interval.

6.1.3 TDRM with Disjoint Time Windows Moving Average (TDRM-DTWMA)

TDRM-DTWMA computes rates for disjoint time windows of duration W. It uses the weighted sum S_t for DTWMA in Equation (26) and $T_t = W$. Memory is $M = W$ and average delay is $\overline{D} = M$. As S_t may be zero, measured rates may be zero. In contrast to TDRM-TWMA, TDRM-DTWMA does not necessarily require storage of samples. It yields the same rate impulse like TDRM-TWMA, but the impulse for a sample may become visible only in the next measurement window.

6.1.4 TDRM with DTWMA and (U)EMA (TDRM-DTWMA-(U)EMA)

TDRM-DTWMA-(U)EMA calculates rates according to TDRM-DTWMA and smoothes them with (U)EMA. It is used in [9] for dequeue rate estimation and requires two parameters: the window size W and the smoothing parameter a. The resulting memory is $M = \frac{W}{1-a}$ and the average delay is $\overline{D} = \frac{W}{2} + \frac{a \cdot W}{1-a} + \frac{W}{2} = M$. An advantage of this method over TDRM-DTWMA is that the effect of measured samples becomes visible after at most W instead of M time. Furthermore, samples contribute with vanishing degree to all future measured rates instead of only to the measured rate of the following measurement window.

The rate impulse starts with rate $\frac{1-a}{W} = \frac{1}{M}$ and is reduced by a factor of a for consecutive measurement intervals of duration W. Again, the rate impulse of a sample may become visible only at the beginning of the next measurement interval. This takes at most $W = M \cdot (1-a)$ time and makes TDRM-DTWMA-(U)EMA react faster than TDRM-DTWMA although they both exhibit the same average delay. Unlike TDRM-DTWMA-EMA, TDRM-DTWMA-UEMA does not suffer from a bias towards the measured rate of the first measurement window.

6.1.5 TDRM with UTEMA (TDRM-UTEMA)

TDRM-UTEMA leverages the weighted sum S_t of UTEMA according to Equation (29) and uses the weighted time

$$T_t = \int_0^t e^{-\beta\cdot(t-\tau)} d\tau = \frac{1}{\beta} \cdot (1 - e^{-\beta\cdot t}). \qquad (42)$$

The memory and delay of TDRM-UTEMA are $M = D = \frac{1}{\beta}$. The rate impulse of TDRM-UTEMA is an exponentially decreasing function. In Figure 4 it is visualized for $T_t = \frac{1}{\beta}$, i.e., idealized for $t \to \infty$, and resembles the rate impulse of TDRM-DTWMA-UEMA. In fact, TDRM-UTEMA can be viewed as the limit of TDRM-DTWMA-UEMA for decreasing window sizes W.

The advantage of TDRM-UTEMA over TDRM-DTWMA-UEMA is its immediate reaction and the need for only a single parameter β. The advantage of TDRM-DTWMA-UEMA is its computational efficiency as it does not require the computation of exponential functions.

6.1.6 TDRM with UTEMA and Continuous Packet Arrivals (TDRM-UTEMA-CPA)

In [8], the following recursion formula has been applied for online rate computation:

$$R_{t_i} = \begin{cases} \frac{X_0}{t_0} & i = 0 \\ e^{-\beta\cdot(t_i - t_{i-1})} \cdot R_{t_{i-1}} + \\ (1 - e^{-\beta\cdot(t_i - t_{i-1})}) \cdot \frac{X_i}{t_i - t_{i-1}} & \text{otherwise} \end{cases}. \qquad (43)$$

The start of the measurement period is at $t = 0$. Later in this section we show that this recursive formula essentially implements TDRM-UTEMA with the assumption that packets continuously arrive during their preceding inter-arrival time instead of arriving instantly at their actual arrival time.

6.2 Comparison of TDRM Methods

We first visualize the results of TDRM-{TWMA, DTWMA, DTWMA-UEMA, UTEMA} for burst arrivals with equal inter-arrival times. Then we measure a Poisson arrival process using different memory. Finally, we show that TDRM-UTEMA-CPA can be derived from TDRM-UTEMA and point out its shortcomings.

6.2.1 Measuring a Burst

The bottom line of Figure 5 shows a burst of equal-size packets arriving with equal inter-arrival times at a rate of $\lambda = \frac{1}{\Delta t}$. Before and after the burst the arrival rate is zero. The figure illustrates the measured rates for the above mentioned TDRM methods, each configured with a memory of $M = 10 \cdot \Delta t$.

TDRM-TWMA produces a step function that first linearly increases, reaches a measured rate of $\frac{1}{\Delta t}$ after $M = 10 \cdot \Delta t$ time, stays constant for a while, then linearly decreases, and reaches zero again after M time. The method exhibits a high timeliness as the rate increases shortly after the arrival of the first packet and returns to zero shortly after the arrival of the last packet.

TDRM-DTWMA reveals a positive rate only with the start of the next measurement window after the arrival of the first packet and captures the full rate only one measurement window later. The end of the burst is reflected late by TDRM-DTWMA's measured rate.

TDRM-DTWMA-UEMA also yields a step function. Its measured rate reflects the beginning of the burst earlier than TDRM-DTWMA since it uses a shorter measurement window. However, it takes some time to approach the full observed rate of $\frac{1}{\Delta t}$ because it does not completely forget about the past when no packets arrived. In a similar way, the measured rate geometrically decreases after the arrival of the last

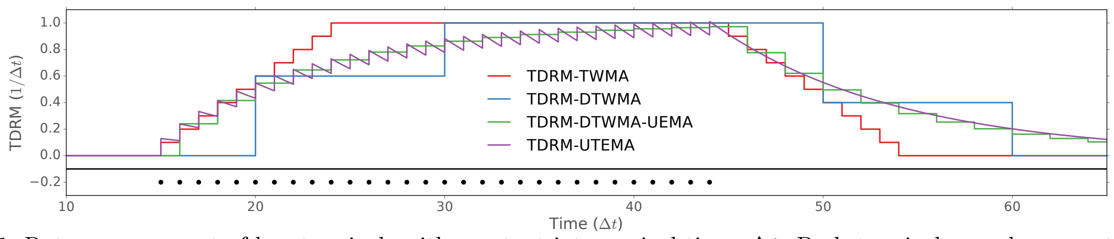


Figure 5: Rate measurement of burst arrivals with constant inter-arrival times Δt. Packet arrivals are shown on the bottom of the figure. The TDRM methods are configured with a memory of $M = 10 \cdot \Delta t$. TDRM-DTWMA-UEMA uses a window of $W = 2 \cdot \Delta t$ and $a = \frac{4}{5}$.

packet. Therefore, it takes long to approach zero. We used a measurement window of $W = 2 \cdot \Delta t$ for the rate curve in the figure. A measurement window of $W = 2.5 \cdot \Delta t$ alternately covers 2 or 3 arrivals which imposes an oscillating behavior on the obtained rates although the observed process has constant rate. This is a general, undesired artifact of window-based TDRM methods.

The rate measured by TDRM-UTEMA jumps with every packet arrival and exponentially decreases in the absence of new samples. Therefore, its shape at large resembles the one measured by TDRM-DTWMA-UEMA, but TDRM-UTEMA does not exibit the above mentioned artefact.

6.2.2 *Measuring a Poisson Process*

We consider a Poisson arrival process with an arrival rate of $\lambda = \frac{1}{\Delta t}$ and equal-size samples over a duration of $10^6 \cdot \Delta t$. A cutout between $40 \cdot \Delta t$ and $240 \cdot \Delta t$ is illustrated in Figure 6(a). Figures 6(b)–6(e) illustrate time-dependent rates of this process measured by TDRM-{TWMA, DTWMA, DTWMA-UEMA, UTEMA} with a memory of $M = 20 \cdot \Delta t$ and $M = 40 \cdot \Delta t$ as well as the rate difference between these curves.

TDRM-TWMA's measured rate in Figure 6(b) is a step function and changes whenever a new sample arrives or an old sample leaves the measurement window. Frequently arriving samples cause rising or high rates while frequently leaving samples cause falling or low rates. The curves measured with the longer memory of $M = 40 \cdot \Delta t$ are influenced by the same arriving samples but different leaving samples compared to the curves measured with shorter memory. Therefore, the memory has a clear and non-continuous impact on measured TDRM-TWMA's rates. The resulting rate difference is also shown in the figure. We calculate the average absolute rate difference and denote it by R^{abs}_{diff}. It is given in the captions of the figures and it is relatively high for TDRM-TWMA. The variance of TDRM-TWMA's rate curves is rather high because it computes the rate only from the small number of samples within its measurement window. To quantify this observation, we compute the coefficients of variation $c_{var}(M)$ of the rate curves and also report them in the captions of the figures.

The rate curves for TDRM-DTWMA in Figure 6(c) suffer from the same problems as TDRM-TWMA which is quantified by R^{abs}_{diff} and $c_{var}(M)$. They behave similarly as those for TDRM-TWMA but are clearly delayed. Some low (high) values of TDRM-TWMA are suppressed, e.g., between $t = 160 \cdot \Delta t$ and $t = 180 \cdot \Delta t$ because TDRM-DTWMA's few discrete measurement windows comprise both low- and high-frequent arrivals.

We consider TDRM-DTWMA-UEMA configured with a window of size $5 \cdot \Delta t$. Its rates are reported in Figure 6(d) and are represented by step functions. TDRM-DTWMA-UEMA's rates measured with $M = 20 \cdot \Delta t$ and $M = 40 \cdot \Delta t$ exhibit significantly less difference compared to TDRM-TWMA and TDRM-DTWMA because TDRM-DTWMA-UEMA is continuous with regard to the configured memory. The measured rates reveal a lower coefficient of variation because their calculation takes all previous samples into account.

Rates measured by TDRM-UTEMA are illustrated in Figure 6(e). They jump at each packet arrival and exponentially decrease in between. They are very similar to those measured by TDRM-DTWMA-UEMA which exhibit geometric decay in the absence of samples. TDRM-UTEMA's rates are also continuous with regard to memory. TDRM-UTEMA exhibits the least average difference R^{abs}_{diff} between rates measured with different memory. Its rate computation also respects all past samples and its rates reveal the least variance among all TDRM methods.

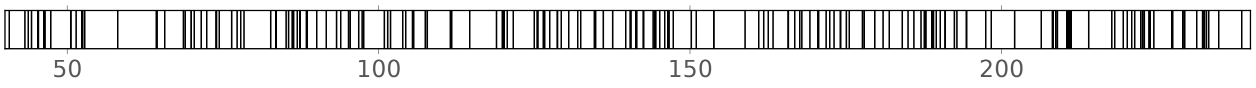
The variance of measured rates decreases for all TDRM methods with increasing memory. However, the variance for TDRM-TWMA and TDRM with $M = 40 \cdot \Delta t$ is about as large as the variance for TDRM-DTWMA-UEMA and TDRM-UTEMA with $M = 20 \cdot \Delta t$.

6.2.3 *Comparison of TDRM-UTEMA-CPA and TDRM-UTEMA*

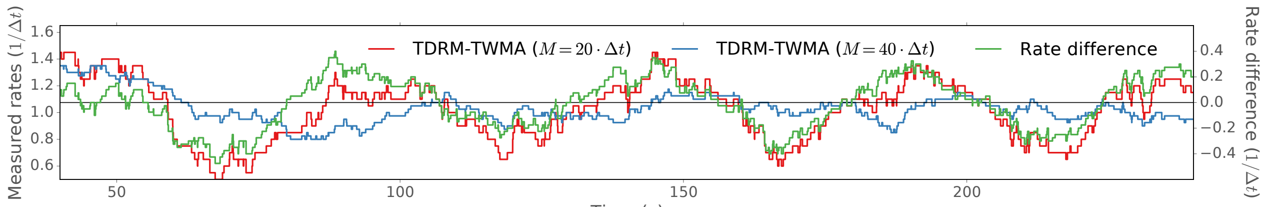
We derive the recursion formula in Equation (43) to point out its connection with TDRM-UTEMA. We assume that packets X_i continuously arrive with rate $\frac{X_i}{t_i - t_{i-1}}$ during their preceding inter-arrival time and partial packets are already devaluated with passing time τ by $e^{-\beta \cdot (t_i - \tau)}$, similar to Equation (29). As a result, the remaining packet size at t_i is the modified sample size

$$\begin{aligned} X_i^* &= \int_{t_{i-1}}^{t_i} \frac{X_i}{t_i - t_{i-1}} \cdot e^{-\beta \cdot (t_i - \tau)} d\tau \\ &= \frac{X_i}{\beta \cdot (t_i - t_{i-1})} \cdot (1 - e^{-\beta \cdot (t_i - t_{i-1})}). \qquad (44) \end{aligned}$$

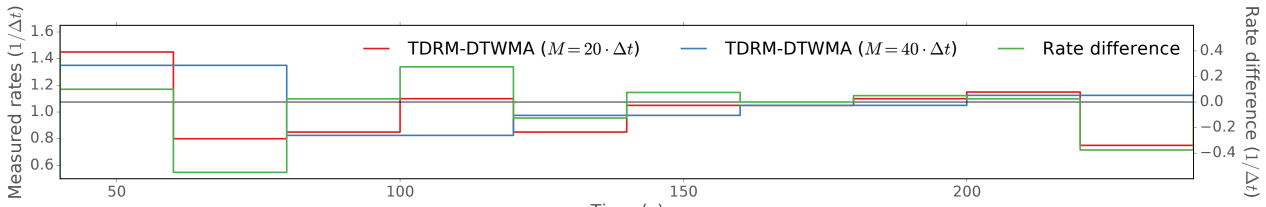
An exception is the first modified sample whose size is computed as $X_0^* = \frac{X_0}{\beta \cdot t_0}$. Application of TDRM-UTEMA to these modified samples yields TDRM-UTEMA-CPA's recursion formula in Equation (43). More specifically, the sample sum in Equation (29) is computed based on X_i^* instead of X_i and $T_t = \frac{1}{\beta}$ is taken as weighted time which is exact for $t \to \infty$. Therefore, TDRM-UTEMA-CPA has only an initial bias due to the different computation of X_0^*, but does not exhibit a persistent bias due to its consistency with TDRM-UTEMA. This is not obvious because Equation (43) looks at first sight like an application of TEMA to short-term rates



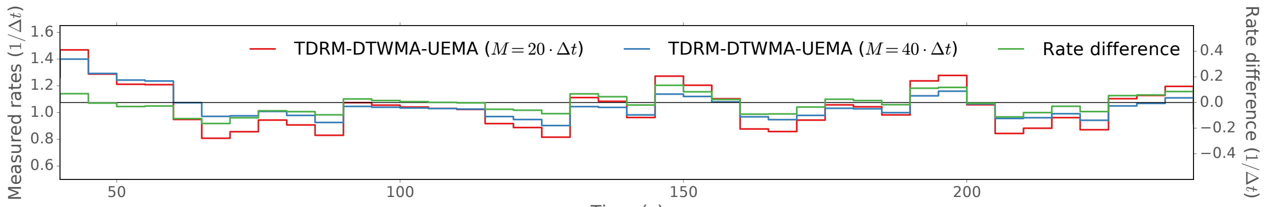
(a) Cutout of the observed Poisson arrival process with arrival rate $\lambda = \frac{1}{\Delta t}$.



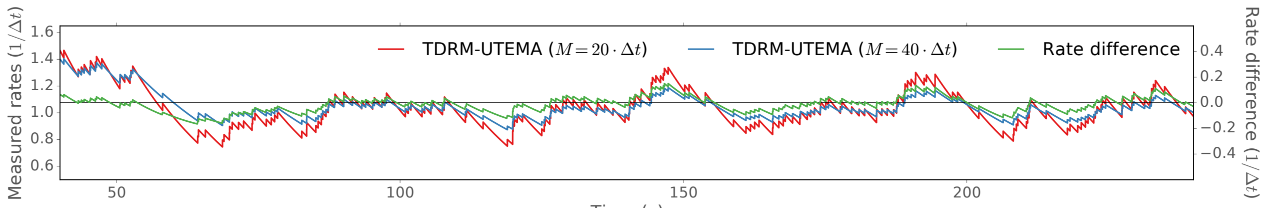
(b) TDRM-TWMA: $R^{abs}_{diff} = 0.126$, $c_{var}(20 \cdot \Delta t) = 0.224$, $c_{var}(40 \cdot \Delta t) = 0.158$



(c) TDRM-DTWMA: $R^{abs}_{diff} = 0.172$, $c_{var}(20 \cdot \Delta t) = 0.223$, $c_{var}(40 \cdot \Delta t) = 0.159$



(d) TDRM-DTWMA-UEMA: $R^{abs}_{diff} = 0.059$, $c_{var}(20 \cdot \Delta t) = 0.169$, $c_{var}(40 \cdot \Delta t) = 0.116$



(e) TDRM-UTEMA: $R^{abs}_{diff} = 0.052$, $c_{var}(20 \cdot \Delta t) = 0.158$, $c_{var}(40 \cdot \Delta t) = 0.112$

Figure 6: Poisson arrival process and time-dependent rates measured by various TDRM methods with $M = 20 \cdot \Delta t$ and $M = 40 \cdot \Delta t$; coefficient of variations (c_{var}) of and average deviations R^{abs}_{diff} between the two curves are given in the captions.

$R^*_i = \frac{X_i}{t_i - t_{i-1}}$ computed at each packet arrival, and TEMA exhibits both an initial and a persistent bias.

After the initial bias towards X^*_0 has vanished, TDRM-UTEMA-CPA yields rates for arrival instants that are slightly lower than those of TDRM-UTEMA. While TDRM-UTEMA yields a piecewise exponentially decaying function, TDRM-UTEMA-CPA updates rates only at arrival instants and leads to a step function. The missing decay during inter-arrival times can overestimate rates over time. To quantify this effect, we measured the rates of a Poisson process ($c_{var} = 1.0$) and an arrival process whose inter-arrival times have a coefficient of variation of $c_{var} = 2.0$. The processes have an arrival rate of $\lambda = \frac{1}{\Delta t}$ and take $10^6 \cdot \Delta t$ time. Table 6 shows average rates measured by TDRM-UTEMA and TDRM-UTEMA-CPA with a memory of $M \in \{10, 100\} \cdot \Delta t$. TDRM-UTEMA does not overestimate rates while the overestimation through TDRM-UTEMA-CPA is significant for short memory M and the arrival process with highly varying inter-arrival times.

Table 6: Average rates measured by TDRM-{UTEMA,UTEMA-CPA}.

	$c_{var}(A) = 1.0$		$c_{var}(A) = 2.0$	
Memory M	$10 \cdot \Delta t$	$100 \cdot \Delta t$	$10 \cdot \Delta t$	$100 \cdot \Delta t$
TDRM-UTEMA	1.000	1.000	1.000	1.000
TDRM-UTEMA-CPA	1.048	1.005	1.186	1.020

6.3 Summary

We have proposed a framework for TDRM. We considered four methods from literature and a novel one: TDRM-UTEMA. Only TDRM-TWMA and TDRM-UTEMA immediately reflect rate changes in measured time-dependent rates. In contrast, TDRM-DTWMA and TDRM-DTWMA-UEMA take some time until rate changes become visible. Thereby, TDRM-DTWMA-UEMA uses shorter measurement windows than TDRM-DTWMA and exponential smoothing so that it can react faster to rate increases than TDRM-DTWMA. For small measurement windows, rates measured by TDRM-DTWMA-UEMA converge to those measured by TDRM-UTEMA. While TDRM-TWMA and TDRM-DTWMA yield measurement curves with high variance, TDRM-UTEMA and TDRM-DTWMA-UEMA exhibit clearly lower variance when being configured with the same memory because they average over all past samples. While time-dependent rates measured by TDRM-DTWMA-UEMA and TDRM-UTEMA are continuous with regard to configured memory, rates measured by TDRM-TWMA and TDRM-DTWMA significantly depend on the configured memory. This makes TDRM-TWMA and TDRM-DTWMA more difficult to interpret and use. The novel TDRM-UTEMA is the only of these studied methods (1) whose measured rates are continuous with regard to memory and (2) immediately react to rate changes, (3) which can be configured by a single parameter, and (4) which cannot not produce window-based artifacts. Therefore, its measured rates depend least on the chosen memory. Nevertheless, the two-parametric TDRM-DTWMA-UEMA can possibly serve as a less computation-intensive approximation of TDRM-UTEMA. We showed that TDRM-UTEMA-CPA is a variant of TDRM-UTEMA but may overestimate rates.

7. CONCLUSION

We have presented a framework for the definition of moving averages (MAs) including performance metrics like memory and delay. We presented several MA variants for evenly and unevenly spaced time series, showed that they fit well into that framework, and demonstrated that some of them have a bias. We proposed the unbiased exponential MA (UEMA) and the unbiased time-exponential MA (UTEMA) as novel MA methods that avoid such a bias. We configured MA methods such that they revealed the same memory and produced comparable results. Our analysis of UEMA showed that the memory allows for a tradeoff between accuracy and timeliness when mean values are determined. We also suggested some equations that help to choose appropriate smoothing parameters when UEMA should provide average values with a certain accuracy. We discussed moving histograms (MHs) as an extension of MAs and showed how they may be used to determine quantiles. Finally, we extended the framework to time-dependent rate measurement (TDRM). We embedded four existing TDRM methods in that framework and suggested TDRM-UTEMA as a novel method that excels by its timeliness, ease of configuration, and by the fact that its measured rates continuously depend on the configured memory. We performed experiments to illustrate and compare these TDRM methods and pointed out their pros and cons.

We presented a demo of all methods discussed in this work at [18] and made the source code available at [19].

This work focused on online measurement, i.e., on the assumption that future samples of the measured process are unknown. It has applicability in self-adaptive systems. The work may be extended to offline measurement so that time series can be smoothed while taking both past and future samples into account when calculating averages, histograms, or time-dependent rates.

8. REFERENCES

[1] Information Sciences Institute, "RFC793: Transmission Control Protocol," September 1981.

[2] G. Jenkins and D. Watts, *Spectral analysis and its applications*. Holden-Day, 1968.

[3] R. M. Chiulli, *Quantitative Analysis: An Introduction*. CRC Press, 1999.

[4] A. Eckner, "Algorithms for Unevenly Spaced Time Series: Moving Averages and Other Rolling Operators," Apr. 2012.

[5] E. Zivot and J. Wang, *Modelling Financial Time Series with S-Plus*. Springer, 2006.

[6] L. Burgstahler et al., "New Modifications of the Exponential Moving Average Algorithm for Bandwidth Estimation," in *ITC Specialist Sem.*, 2002.

[7] M. Alizadeh *et al.*, "CONGA: Distributed Congestion-Aware Load Balancing for Datacenters," in *ACM SIGCOMM*, Chicago, IL, USA, Aug. 2014.

[8] I. Stoica et al., "Core-Stateless Fair Queueing: A Scalable Architecture to Approximate Fair Bandwidth Allocations in High-Speed Networks," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, Feb. 2003.

[9] R. Pan *et al.*, "RFC8033: Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem," https://tools.ietf.org/html/rfc8033, Feb. 2017.

[10] R. Martin and M. Menth, "Improving the Timeliness of Rate Measurements," in *GI/ITG MMB*, 2004.

[11] G. Bianchi et al., "On-Demand Time-Decaying Bloom Filters for Telemarketer Detection," *ACM CCR*, vol. 41, no. 5, Oct. 2011.

[12] M. Menth et al., "Time-Exponentially Weighted Moving Histograms (TEWMH) for Application in Adaptive Systems," in *IEEE Globecom*, 2006.

[13] W. D. Kelton and A. M. Law, *Simulation Modeling and Analysis*. McGraw Hill Boston, 2000.

[14] R. G. Brown and R. F. Meyer, "The Fundamental Theorem of Exponential Smoothing," *Operations Research*, vol. 9, no. 5, pp. 673–685, 1961.

[15] C. Chatfield *et al.*, "The Holt-Winters Forecasting Procedure," *Appl. Statistics*, vol. 27, no. 3, 1978.

[16] S. J. Hunter, "The Exponentially Weighted Moving Average," *Jrnl. of Quality Techn.*, vol. 4, no. 18, 1986.

[17] P. E. Maravelakis *et al.*, "An EWMA Chart for Monitoring the Process Standard Deviation when Parameters are Estimated," *Comp. Statistics & Data Analysis*, vol. 53, no. 7, 2009.

[18] M. Menth and F. Hauser, "Demo: Time Series Online Measurement for Python (TSOMpy)," in *ACM/SPEC International Conference on Performance Engineering (ICPE)*, L'Aquila, Italy, Apr. 2017.

[19] ——, "TSOMpy – Time Series Online Measurement for Python," https://www.github.com/uni-tue-kn/TSOMpy, 2017.

3.4 Demo: Time Series Online Measurement for Python (TSOMpy)

Demo: Time Series Online Measurement for Python (TSOMpy)

Michael Menth and Frederik Hauser
Department of Computer Science
University of Tuebingen, Germany
{menth, frederik.hauser}@uni-tuebingen.de

ABSTRACT

TSOMpy is a Python library for online measurement of time series, i.e., it provides functions to calculate moving averages, moving histograms, and time-dependent rates. The demo illustrates various methods for these concepts and points out their differences. The tool can be used to apply online measurement to time series randomly generated according to specified stochastic processes or to own data sets. The library furthermore allows the reproduction of the tables and figures presented in [1].

1. INTRODUCTION

Adaptive systems monitor their states and use these measurement values for control purposes. Mostly, neither the last observation value nor long-term averages are of interest, but averages of observations over the recent past are needed. We address this issue by online measurement that includes moving averages (MA), moving histograms (MH), and time-dependent rate measurement (TDRM). With online measurement only past samples can be used since future samples are not yet available. In [1], we proposed a framework for the definition of MAs, MHs, and TDRMs. We pointed out shortcomings of existing methods and suggested improvements.

We developed a library for time series online measurement in Python (TSOMpy) implementing all investigated methods of [1] including test and plot functions. The proposed demo illustrates the use of TSOMpy, reproduces the experiments and their visualizations in [1], and allows to apply the various MA, MH, and TDRM methods to configurable stochastic processes and own sample traces.

We implemented TSOMpy in Python because this programming language is most suitable for rapid prototyping and it is one of the most widely used programming language in statistics and data science today. It allows programmers to produce readable code through plain syntactical concepts, and profits from a wealth of available external community-based libraries that can be easily used. Python is an interpreted programming language supporting, e.g., the object-oriented, the procedural, and the functional programming paradigms. We use Python in version 3.5.0 and limited the dependency on external libraries to matplotlib [2] for data visualization.

The source code of TSOMpy will be published under a GPLv3 license on Github [2].

The authors acknowledge the funding by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/2-1.

ICPE'17 April 22-26, 2017, L'Aquila, Italy
© 2017 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-4404-3/17/04.
DOI: http://dx.doi.org/10.1145/3030207.3053673

2. RELATED WORK

General averages like CumSum or (biased) exponential MAs (EMA) are simple and quickly programmed or available as functions in many programming languages or external libraries. One example is the qcc package [3] for the R programming language. The Python community also provides numerous external libraries for basic online measurement: Numpy [4] supports a method for a customizable, weighted average. Pandas [5] includes implementations for CumSum and the (biased) EMA. Other libraries provide CumSum and (biased) EMA for analysis of stock market data [6] or implement the window-based moving average (WMA) [7]. However, we could not find packages implementing advanced online measurement methods like unbiased EMA, time-dependent MAs, time-dependent MHs, novel TDRM methods, and allow their evaluation and comparison. Therefore, we developed the TSOMpy library.

3. CONCEPT

TSOMpy allows the generation of time series according to stochastic processes or alternatively read time series from CSV files. It implements the online measurement methods presented in [1] and allows their application to the time series with a graphical user interface. Additional functions reproduce the tables and figures shown in [1]. In the following, we explain the time series generation offered by TSOMpy. We give an overview of the implemented online measurement methods including class structures and challenges for their illustration.

3.1 Generation and Import of Time Series

Evenly spaced time series can be represented by a vector of values which are associated with evenly spaced time instants whose inter-sample time is a basic time unit Δt. In contrast, unevenly spaced time series consist of both a vector of values and a vector of time instants with possibly different, non-zero inter-sample times. TSOMpy supports generation of constant, equally, exponentially, normally, Gamma-, and

Table 1: Class names of online measurement methods and their full names with reference to their descriptions in [1].

Class name	Full method's name
MAs (for evenly-spaced time series)	
CumMean	Cumulative Mean (3.1.2)
WMA	Window MA (3.1.3)
DWMA	Disjoint Windows MA (3.1.4)
UEMA	Unbiased Exponential MA (3.1.5)
EMA	Exponential MA (3.1.6)
Time-dependent MAs (for unevenly-spaced time series)	
TWMA	Time Window MA (3.2.2)
DTWMA	Disjoint Time Windows MA (3.2.3)
UTEMA	Unbiased Time-Exponential MA (3.2.4)
TEMA	Time-Exponential MA (3.2.5)
MHs (for evenly-spaced time series)	
MH_CumMean	MH with CumMean (5.1)
MH_WMA	MH with WMA (not considered in [1])
MH_DWMA	MH with DWMA (not considered in [1])
MH_UEMA	MH with UEMA (5.1)
MH_EMA	MH with EMA (not considered in [1])
Time-dependent MHs (for unevenly-spaced time series)	
TDMH_TWMA	TDMH with TWMA (not considered in [1])
TDMH_DTWMA	TDMH with DTWMA (not considered in [1])
TDMH_UTEMA	TDMH with UTEMA (5.2)
TDMH_TEMA	TDMH with TEMA (not considered in [1])
Time-dependent RMs (for unevenly-spaced time series)	
TDRM_TWMA	TDRM with Time Window MA (6.1.2)
TDRM_DTWMA	TDRM with DTWMA (6.1.3)
TDRM_DTWMA_UEMA	TDRM with DTWMA and UEMA (6.1.4)
TDRM_UTEMA	TDRM with UTEMA (6.1.5)
TDRM_UTEMA_CPA	TDRM with UTEMA and Continuous Packet Arrivals (6.1.6)

Pareto-distributed values and also values created according to an autoregressive process. It offers constant, equally, exponentially, Gamma-, and Pareto-distributed inter-sample times for the time vectors. The distributions can be configured with appropriate parameters. Both time and value vectors can be imported from CSV files. It is possible to combine imported time or value vectors with generated counterparts.

3.2 Online Measurement Methods

Table 1 lists all implemented online measurement methods. Normal MAs and MHs are applicable only to evenly spaced time series while time-dependent MAs (TDMA) and MHs (TDMH) are extended for application to unevenly spaced time series. TDRM methods are applied only to unevenly spaced time series. They can also be applied to mere time vectors with a constant value vector or to a value vector combined with an evenly spaced time vector.

We used an object-oriented class hierarchy for the implementation of all online measurement methods. Normal MA methods inherit from "MovingAverage", TDMA methods inherit from "TimeDependentMovingAverage", MH methods inherit from "MovingHistogram", and TDMH methods inherit from "TimeDependentMovingHistogram". "TimeDepenentRateMeasurement" inherits from "TimeDependentMovingAverage" and TDRM methods inherit from both "TimeDependentMovingRate" and possibly from some specific TDMA method.

Some TDRM methods produce step curves with jumps at sample instants and drops some constant time afterwards. Others lead to curves with jumps at sample instants and continuously decreasing phases in between. Illustrations are provided in [1]. TSOMpy produces such curves and offers functions to calculate variance, differences, and accumulates.

4. DEMONSTRATION

The demonstration explains various online measurement methods and points out their differences using visualization and quantitative metrics.

The module *TsomDemo* includes classes demonstrating the usage of TSOMpy, e.g., the reproduction of the tables and figures in [1] with the same or different parameters. It also provides a user interface to apply the online measurement methods to imported data or to specified random data. In addition to visualization, calculated averages and rates can be exported as CSV for further investigation, e.g., in a statistical software tool like R. TSOMpy includes functions that may be used by other processes to calculate the proposed metrics and to visualize them.

5. CONCLUSION

TSOMpy is a Python library for time series online measurement. The associated tool and the demo illustrate various algorithms for the moving averages (MA), moving histograms (MH), and time-dependent rate measurement (TDRM) presented in [1]. Moreover, it reproduces the results presented in the tables and figures of that paper and allows the community to evaluate the proposed online measurement methods with own experiments and data. On the one hand, the tool may serve educational purposes and be used for the processing of research data. On the other hand, the functions provided by the library may directly be integrated in control algorithms of adaptive systems.

6. REFERENCES

[1] M. Menth and F. Hauser, "On Moving Averages, Histograms and Time-Dependent Rates for Online Measurement," in *ACM/SPEC ICPE*, 2017.

[2] ——, "TSOMpy – Time Series Online Measurement in Python," https://www.github.com/uni-tue-kn/TSOMpy, 2017.

[3] L. Scrucca. (2014, Oct.) Cran: Quality control charts (qcc). [Online]. Available: https://cran.r-project.org/web/packages/qcc/

[4] N. Developers. (2016, May) Numpy. [Online]. Available: http://www.numpy.org/

[5] T. P. D. Team. (2016, Dec.) Python data analysis library. [Online]. Available: http://pandas.pydata.org/

[6] C. Zhuang. (2016, Nov.) Python package index: Stockstats 0.2.0. [Online]. Available: https://pypi.python.org/pypi/stockstats

[7] S. Reifschneider. (2011, Nov.) Github repository: Python-movingaverage. [Online]. Available: https://github.com/linsomniac/python-movingaverage

3.5 FunSpec4DTMC – A Tool for Modelling Discrete-Time Markov Chains Using Functional Specification

FunSpec4DTMC – A Tool for Modelling Discrete-Time Markov Chains Using Functional Specification

Frederik Hauser, Dominik Krauß, and Michael Menth

University of Tuebingen, Chair of Communication Networks,
Sand 13, 72076 Tuebingen, Germany
`{frederik.hauser,menth}@uni-tuebingen.de`
`johannes-dominik.krauss@student.uni-tuebingen.de`

Abstract. We present a tool for the analysis of finite discrete-time Markov chains (DTMCs). As a novelty, the tool offers functional specification of DTMCs and implements forward algorithms to compute the stationary state distribution x_s of the DTMC or derive its transition matrix P [19]. In addition, we implement nine direct and indirect algorithms to compute various metrics of DTMCs based on P including an algorithm to determine the period of the DTMC. The tool is intended for both production purposes and as platform for teaching the functional specification of DTMCs. It is published under GPLv3 [3] on Github [2].

1 Introduction

Discrete-time Markov chains (DTMCs) are a widely applied concept for system modelling. Typically, DTMCs are defined by a stochastic matrix P that holds probabilities for transitions among system states. The vector x_n describes the state distribution of a system after n transitions. Consecutive distribution vectors x_n are calculated by $x_{n+1} = x_n \cdot P$. The stationary state distribution fulfills $x_s = x_s \cdot P$. It reflects the average state distribution after multiple transitions and is a useful base for the derivation of further specific performance metrics. Theoretical background of DTMCs is described in [21,22]. There are many scientific analysis tools [6,13,14,16–18,23] and libaries for the field of teaching [6,20]. All utilize the transition matrix P as the base for analysis.

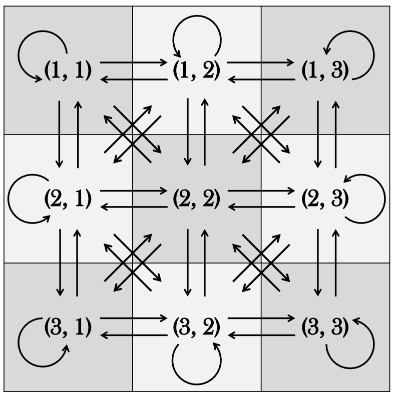
In this work, we present a tool for modelling DTMCs with a finite state space using the novel functional specification suggested in [19]. We introduce the functional specification by an example. We consider a two-dimensional constraint random walk on a grid with coordinates (a, b) and integer values $a \in \{A_{min}, ..., A_{max}\} = \mathcal{A}$ and $b \in \{B_{min}, ..., B_{max}\} = \mathcal{B}$. The walk starts at position (A_{min}, B_{min}). In any transition, the position may change horizontally by integer values $\mathcal{H} = \{H_{min}, ..., H_{max}\}$ and vertically by $\mathcal{V} = \{V_{min}, ..., V_{max}\}$, all with equal probability. We consider the system with $A_{min} = B_{min} = 1$, $A_{max} = B_{max} = 3$, $H_{min} = V_{min} = -1$, and $V_{max} = H_{max} = 1$. It can be modelled by a two-dimensional state space $\mathcal{X} = \mathcal{A} \times \mathcal{B}$ and a factor space $\mathcal{Y} = \mathcal{H} \times \mathcal{V}$ with $\mathcal{H} = \mathcal{V} = \{-1, 0, 1\}$. The random variables $X_n = (A_n, B_n) \in \mathcal{X}$ describe the position of the walk after n transitions. Given a random factor for the move $Y = (H, V) \in \mathcal{Y}$, the next position $(A_{n+1},\ B_{n+1}) \in \mathcal{X}$ of the walk is determined by

$$A_{n+1} = min(A_{max}, max(A_{min}, A_n + H)) \quad (1)$$

$$B_{n+1} = min(B_{max}, max(B_{min}, B_n + V)). \quad (2)$$

This is a stochastic recursive equation that constitutes the state transition function of the system $f : \mathcal{X} \times \mathcal{Y} \to \mathcal{X}$. Together with the distribution y of the factor space $\mathcal{Y}$ this function constitutes a DTMC [19]. A so-called forward algorithm may be used to compute consecutive state distributions x_n based on this description without the use of a transition matrix P. The transition matrix P can be derived by a similar algorithm so that other analytical methods can be applied to it.

Figure 1(a) shows all states of the random walk with potential transitions. Figure 1(b) illustrates the state transition matrix. The stationary state distribution yields $x_s(i) = \frac{1}{9}$ for any $i \in \mathcal{X}$.



(a) State space and potential transitions.

$$
\begin{array}{c|ccccccccc}
 & (1,1) & (1,2) & (1,3) & (2,1) & (2,2) & (2,3) & (3,1) & (3,2) & (3,3) \\
\hline
(1,1) & \frac{4}{9}, & \frac{2}{9}, & 0, & \frac{2}{9}, & \frac{1}{9}, & 0, & 0, & 0, & 0 \\
(1,2) & \frac{2}{9}, & \frac{2}{9}, & \frac{2}{9}, & \frac{1}{9}, & \frac{1}{9}, & \frac{1}{9}, & 0, & 0, & 0 \\
(1,3) & 0, & \frac{2}{9}, & \frac{4}{9}, & 0, & \frac{1}{9}, & \frac{2}{9}, & 0, & 0, & 0 \\
(2,1) & \frac{2}{9}, & \frac{1}{9}, & 0, & \frac{2}{9}, & \frac{1}{9}, & 0, & \frac{2}{9}, & \frac{1}{9}, & 0 \\
(2,2) & \frac{1}{9}, & \frac{1}{9}, & \frac{1}{9}, & \frac{1}{9}, & \frac{1}{9}, & \frac{1}{9}, & \frac{1}{9}, & \frac{1}{9}, & \frac{1}{9} \\
(2,3) & 0, & \frac{1}{9}, & \frac{2}{9}, & 0, & \frac{1}{9}, & \frac{2}{9}, & 0, & \frac{1}{9}, & \frac{2}{9} \\
(3,1) & 0, & 0, & 0, & \frac{2}{9}, & \frac{1}{9}, & 0, & \frac{4}{9}, & \frac{2}{9}, & 0 \\
(3,2) & 0, & 0, & 0, & \frac{1}{9}, & \frac{1}{9}, & \frac{1}{9}, & \frac{2}{9}, & \frac{2}{9}, & \frac{2}{9} \\
(3,3) & 0, & 0, & 0, & 0, & \frac{1}{9}, & \frac{2}{9}, & 0, & \frac{2}{9}, & \frac{4}{9}
\end{array}
$$

(b) State transition matrix P.

Fig. 1: Random walk example.

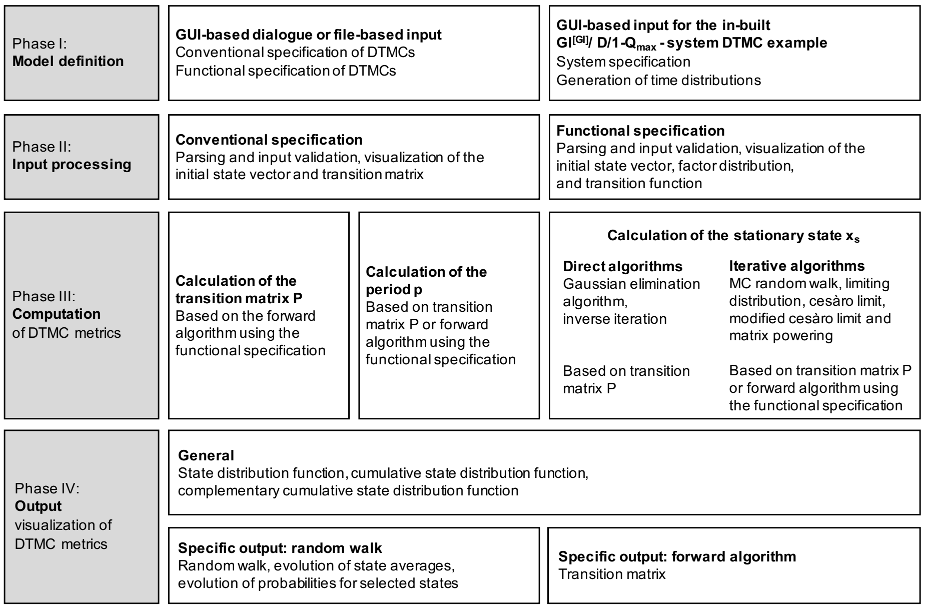
The functional specification might appear more complex, but provides many benefits. First, it allows intuitive modelling of systems with event-triggered state transitions. Events are modelled by factors whose probabilities are described by the factor distribution. The system's state transition in case of particular events is described by the transition function. Therefore, the functional specification is close to the system's behaviour which facilitates modelling of complex systems with even multi-dimensional state spaces. Second, the functional specification allows the modelling of very large DTMCs. The conventional specification requires the transition matrix P which scales quadratically with the number of states. For very large DTMCs, the resulting size of P may be so large (multiple Terabytes) that DTMC analysis based on P may become infeasible. Sparse matrix representation may reduce memory requirements, but its effectiveness depends on the specific use case. With the functional specification, the transition matrix is not needed for the analysis of the DTMC and memory requirements are reduced to the state and factor distribution. The memory requirement for the state transition function is generally small. In [19] further optimization methods are described to speed up the convergence of the consecutive state distributions based on the functional description.

FunSpec4DTMC implements the functional specification and the forward algorithm to calculate consecutive state distributions x_n and the transition matrix P. Besides, the tool offers various direct and iterative computation methods to calculate metrics for DTMCs that are based on P. In particular, the period of finite DTMCs can be derived and methods for output visualization are provided.

The paper is structured as follows. In the next section, we present the core idea and features of FunSpec4DTMC. Section 3 describes the architecture and implementation.

2 Tool Description

FunSpec4DTMC consists of a library implementing the functionality and a graphical user interface (GUI) that allows users to analyse DTMCs in an interactive process. The four phases of FunSpec4DTMC's analysis process for DTMCs are depicted in Figure 2.



Phase I: **Model definition**
GUI-based dialogue or file-based input Conventional specification of DTMCs Functional specification of DTMCs
GUI-based input for the in-built $GI^{[GI]}/D/1$-Q_{max}-system DTMC example System specification Generation of time distributions
Phase II: **Input processing**
Conventional specification Parsing and input validation, visualization of the initial state vector and transition matrix
Functional specification Parsing and input validation, visualization of the initial state vector, factor distribution, and transition function
Phase III: **Computation** of DTMC metrics
Calculation of the transition matrix P Based on the forward algorithm using the functional specification
Calculation of the period p Based on transition matrix P or forward algorithm using the functional specification
Calculation of the stationary state x_s
Direct algorithms Gaussian elimination algorithm, inverse iteration
Iterative algorithms MC random walk, limiting distribution, cesàro limit, modified cesàro limit and matrix powering
Based on transition matrix P
Based on transition matrix P or forward algorithm using the functional specification
Phase IV: **Output** visualization of DTMC metrics
General State distribution function, cumulative state distribution function, complementary cumulative state distribution function
Specific output: random walk Random walk, evolution of state averages, evolution of probabilities for selected states
Specific output: forward algorithm Transition matrix

Fig. 2: Four phases of FunSpec4DTMC's analysis process for DTMCs.

In the first phase (I), the DTMC model is defined by the user. DTMCs can be either defined in a GUI-based input dialogue or imported from JSON project files. As an example for intuitive modelling of DTMCs, our tool offers a system specification dialogue for a $GI^{[GI]}/D/1 - Q_{max}$ queuing system.

In the second phase (II), the DTMC model input is parsed and validated against mistakes in the specification, e.g., state probability vectors that do not sum up to 1. In addition, aspects of the DTMC model such as the initial state vector can be visualized.

In the third phase (III), metrics for DTMCs are calculated. If the DTMC is defined in a conventional way using the transition matrix P, multiple direct and iterative computation algorithms can be applied. The former are accurate and fast but require a large amount of memory. Examples are the Gaussian elimination algorithm and the inverse iteration. The latter requires less memory but lots of iterations to compute the stationary state distribution with high accuracy. The tool offers the following iterative methods to approximate the stationary state distribution x_s:

- DTMC random walk (simulation)
- calculation of the limiting distribution ($\lim_{n\to\infty} x_n$) (applicable to aperiodic DTMCs)
- matrix powering ($\lim_{n\to\infty} P^n$) (applicable to aperiodic DTMCs and to DTMCs with a period of 2^n, $n \in \mathbb{N}_0$)
- calculation of the Cesàro limit ($\lim_{n\to\infty} \frac{1}{n+1} \sum_{i=0}^{n} x_i$)
- modified calculation of the Cesàro limit ($\lim_{n\to\infty} \frac{1}{p} \sum_{n\le i<n+p} x_i$) as introduced in [19]. To that end, the tool analyzes transition structures of the DTMC and computes its period p.

With a functional specification of a DTMC, the tool computes consecutive state distributions x_n and DTMC simulations without the state transition matrix P and uses for this purpose the forward algorithm or just the state transition function f, respectively. Moreover, the state transition matrix P can be derived from the functional specification based on another forward algorithm [19].

In the fourth phase (IV), the output of the DTMC analysis can be visualized. That includes the visualization of general metrics, e.g., the stationary state distribution, and the visualizations of particular computation algorithm specifics, e.g., the random walk.

3 Architecture and Implementation

The architecture of FunSpec4DTMC is based on the model-view-controller (MVC) pattern that separates its functionality from the GUI. We designed an object-oriented class hierarchy and applied design patterns, e.g., the observer or strategy pattern [15], and language constructs, e.g., the signal-and-slot approach [10].

We chose Python in version 3.6.3 [8] as programming language. We use Matplotlib [4] to generate plot figures and SciPy [11] to import common distributions. To apply SciPy's continuous distributions on DTMCs, we implemented mechanisms for discretization and normalization. We implemented the GUI using PyQt5 [7], the Python bindings to the widely applied GUI framework Qt [9]. It is platform-independent and allows the creation of more advanced graphical surfaces compared to simple approaches like Tkinter [12]. Python is an interpreted programming language, i.e., source code is translated at runtime. To compensate performance drawbacks, computational-intense functions are implemented in C and called at runtime. External libraries like NumPy [5] adopt this principle and use efficient implementations, e.g., for vector-matrix and matrix-matrix multiplications. We used the Cython [1] extension to implement the forward algorithm's interleaved loops and the model-specific transition functions. Cython allows to implement CPU-intensive modules as C-extensions in a Python-like syntax with additional annotations such as static type declarations. Afterwards, the Cython source code is transformed into C code, compiled, and called at runtime from within Python. Our tool automatically integrates the model-specific transition function defined by the users into the forward algorithm that is a static part of the tool. Analyzing large DTMCs is limited by the memory on the host system. The functional specification may mitigate but not solve the memory problems that arise with a large number of states. We applied the memory-to-disk swapping mechanism of NumPy so that computing data is stored in a file on the hard disk which can be accessed in small segments.

Within the tool's GUI, users can define multiple projects. For each project, multiple DTMCs can be specified either in the functional or conventional specification within an interactive dialogue or by importing a JSON file. DTMC models and the output of the calculation algorithms can be visualized in multiple plot views. Projects can be stored as files in JSON format and be imported.

References

1. Cython: C-Extensions for Python. `http://cython.org/`
2. Github: FunSpec4DTMC. `https://github.com/uni-tue-kn/funspec4dtmc`
3. GNU General Public License 3. `https://www.gnu.org/licenses/gpl-3.0.en.html`
4. Matplotlib 2.1.0. `https://matplotlib.org/`
5. NumPy - Scientific Computing with Python. `http://www.numpy.org/`
6. PyPI: discreteMarkovChain. `https://pypi.python.org/pypi/discreteMarkovChain`
7. PyQt5. `http://www.numpy.org/`
8. Python 3.6.3. `https://www.python.org/downloads/release/python-363/`
9. Qt. `https://www.qt.io/`
10. Qt5: Signals & Slots. `http://doc.qt.io/qt-5/signalsandslots.html`
11. SciPy. `https://www.scipy.org/`
12. tkinter. `https://docs.python.org/3.6/library/tkinter.html`
13. Benoit, Anne et al.: The Peps Software Tool. In: Proc. from the 13th Int. Conf. on the Technology of Object-Oriented Languages and Systems (TOOLS '13). Springer (2003)
14. Bini, D. A. et al.: Structured Markov Chains Solver: Software Tools. In: Proc. from the Workshop on Tools for Solving Structured Markov Chains (SMCtools '06). ACM (2006)
15. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-oriented Software. Addison-Wesley, Boston, MA, USA (1995)
16. Hermanns, Holger et al.: A Set of Performance and Dependability Analysis Components for CADP. In: Proc. of the 9th Int. Conf. on Theory and Practice of Software (TACAS '03). Springer (2003)
17. Katoen, J.P. et al.: The Ins and Outs of the Probabilistic Model Checker MRMC. In: Proc. of the 6th Int. Conf. on the Quantitative Evaluation of Systems (QUEST '09). IEEE Computer Society Press (2009)
18. Kwiatkowska, M et al.: PRISM 4.0: Verification of Probabilistic Real-time Systems. In: Proc. from the 23rd Int. Conf. on Computer Aided Verification (CAV '11). Springer (2011)
19. Menth, M.: Description and analysis of Markov chains based on recursive stochastic equations and factor distributions. World Journal of Modelling and Simulation, World Academic Union, UK Vol. 7(No. 1, pp. 3-15) (2011)
20. Spedicato, G.A., Kang, T.S., Yalamanchi, S.B., Yadav, D.: The markovchain Package: A Package for Easily Handling Discrete Markov Chains in R
21. Stewart, W.J.: Introduction to the Numerical Solution of Markov Chains. Princeton University Press, Princeton, New Jersey (1994)
22. Stewart, W.J.: Probability, Markov Chains, Queues, and Simulation: The Mathematical Basis of Performance Modeling. Princeton University Press, Princeton, New Jersey (2009)
23. Timmer, M. et al.: Efficient Modelling and Generation of Markov Automata. In: Proc. of the 23rd Int. Conf. on Concurrency Theory (CONCUR '12). Springer (2012)

All online resources were accessed on Nov 6, 2017.