P4-programmable Data Plane for Content-based Publish/Subscribe

Christian Wernecke, Helge Parzyjegla, and Gero Mühl
Institute of Computer Science
University of Rostock, 18051 Rostock, Germany
{christian.wernecke, helge.parzyjegla, gero.muehl}@uni-rostock.de

***Abstract*—The domain-specific programming language P4 enables developers to specify how data plane devices (e.g., switches, routers) process packets. This opens up novel opportunities for efficient information dissemination at the network layer drastically reducing notification delays compared to application-layer publish/subscribe using broker overlay networks.**

In this paper, we present three different strategies that use P4 to realize content-based publish/subscribe. We start with a source routing strategy that is, then, improved to exploit and adapt a preinstalled forwarding tree by adding or removing branches if required. The most advanced strategy enables multiple forwarding trees to be dynamically stitched and adapted as needed.

I. Introduction

Publish/subscribe is a flexible messaging pattern that decouples senders and receivers. On the senders' side are the *publishers* that publish messages (*notifications*) and send them into a *middleware*. On the receivers' side are the *subscribers* that receive notifications of interest from the middleware. Subscribers use *subscriptions* to tell the middleware about the content they are interested in. In a content-based system, subscriptions consist of filters that select desired message content. Similarly, publishers can formulate *advertisements* as filters to announce their messages content. The middleware makes use of subscriptions and advertisements to forward and deliver published notifications to interested subscribers. The middleware is traditionally implemented by a network of applications-layer brokers. A broker de-serializes an incoming packet and makes forwarding decisions based on the packet content, and finally creates a serialized copy of the packet content for each direction in which the packet needs to be forwarded. However, this approach is inefficient and introduces significant delays because the brokers operate at the application layer and a notification message traverses the entire network stack to be de-/serialized. In addition, the brokers communicate using IP unicast, which can result in multiple copies being sent over the same network link.

The P4 language [1] allows the development of customized protocols on the network layer for publish/subscribe and enables increasingly intelligent network devices, such as switches, to take over broker functions. A large fraction of the broker functionality can, thus, be replaced by programmable network elements. Implementations on the network layer can significantly improve delivery latency and efficiency.

II. SDN-based Middleware

Switches are restricted in processing the packet header and there is no possibility to inspect the payload. To circumvent this restriction, a preprocessing operation on the publisher's side is used to encode all relevant forwarding information into the header. To achieve this, a SDN controller first registers all publishers and subscribers and computes appropriate distribution trees that connect the publishers and their subscriber sets. Then, the controller installs corresponding forwarding rules for the distribution trees and notifies the publishers of their potential recipients. Now, when a sender publishes a notification, the sender first determines his individual recipient set, then it labels the packet with routing information, and finally it sends the notification message with the attached header stack labels into the switch infrastructure. The switches inspect the labels and perform matching forwarding actions, according to their flow rules preinstalled by the controller. This label-steered forwarding continues, until all interested recipients have received the packet.

We have implemented several content-based routing protocols using P4. Our pub/sub protocols implement a *perfect* multicast, in the sense that false positive notifications are avoided. The protocols encode routing information into the packet header and switch memory, in different parts and proportions via header stack labels and flow rules. Please note that our publishers have global knowledge of the pub/sub network, enabling them to encode the routing information.

A. Source-based Routing

In our first and source-based approach [8], we implement *multicast source routing* and encode the complete distribution tree of a publisher's notification as set of labels into the packet header. A label refers to one or multiple output ports of a switch. The switches examine the packet header, extract all labels intended for them, and execute associated forwarding rules. We evaluated our source-based protocols on a large pub/sub network and compared their performance with other approaches (i. e., unicast, broadcast, and broker-based approaches). The results show that our protocols can achieve significant bandwidth and latency reductions.

B. Referring Forwarding Trees

Our second approach [9] combines header stacks with distribution trees stored in the switches that use *tree labels* to

address stable sets of receivers that get a majority of a sender's published messages. A tree label reduces the number of header stack entries by assembling sub-paths that are commonly used for notification delivery. Additionally, a tree can be extended by *hop labels*. By encoding tree and hop labels, a publisher can both refer to preinstalled trees and supplement missing paths that lead from a node of the tree to a final destination.

C. Stitching Forwarding Trees

Our third approach [7] exploits forwarding along multiple distribution trees. We combine multiple stored trees and encode corresponding tree labels into the header stack, supplemented with extra *hop* and opposing *stop* labels. Hops interconnect trees and bridge network parts not covered by trees, while stops conversely identify links of tree branches that have to be truncated and where forwarding has to be stopped. We encode distribution information with given stored forwarding trees into the notification header. We also derived several strategies for computing forwarding trees which are described next.

D. Computing Forwarding Trees

Our strategies for computing multicast distribution trees are based on three different levels of increasing application-dependent knowledge and/or runtime statistics.

(i) In the basic case, we derive trees from topological information only. For this purpose, we can, for instance, partition the network and compute a broadcast tree for each partition. These trees, then, cover all hosts in the network, regardless of whether they act as publisher or subscriber.

(ii) In an extended case, we consider publisher-subscriber relationships and prune the topology-derived trees so that only hosts acting as publishers or subscribers remain in the trees. Please note that this strategy presupposes a proper subscriber differentiation and classification.

(iii) In a more advanced case, we consider notification frequencies and determine the fraction of delivered messages per subscriber in order to exclude rarely addressed subscribers from the stored trees. Hence, we reduce the number of stop labels to be encoded and flow rules to be installed.

We have evaluated the strategies' performance in a larger emulated network. Our results show that combining trees with additional distribution information in the notification header can further reduce header length and bandwidth requirements compared to the pure source-based approach. Moreover, the results indicate that the more application-specific knowledge is involved during the installation process of forwarding trees, the more efficiently the notifications can be delivered.

III. Related Work

Several approaches are known in literature that realize publish/subscribe over IP multicast by defining a corresponding IP multicast address for each possible set of receivers. As an example, *OpenFlow Multicast* [3] stores multicast trees in the switches and rely on IP layer forwarding. Unfortunately, this is less suitable for use in content-based pub/sub, since the number of possible combinations may rise enormously to cover all possible permutations of subscribers. To reduce the number of multicast groups, the authors in [2] propose a solution that clusters subscribers with similar interests. With this solution, notifications may be unintentionally delivered to uninterested recipients. In contrast, BIER [6] follows a multicast approach that avoids false positives and encodes a bit-string into the header of a multicast message. Each bit in the bit-string stands for a specific receiver which gets a copy of the message if its bit is set. This allows arbitrary permutation of targets, but the bit-string overhead can be heavy with many potential recipients. Kundel et al. [5] take a data-centric approach that encodes event attributes into the notification header and installs flow rules based on predicate filters into the switch infrastructure. However, the flow rules require an update when subscriptions change. Another data-centric approach [4] introduces a description language to build binary decision trees. The decision trees in turn consist of logical predicates or filter expressions that are compiled into forwarding rules for the P4 switches. However, for large pub/sub networks this approach requires a high filter selectivity for the subscriptions to keep the number of rules on a manageable level.

IV. Conclusions and Future Work

We presented three different approaches to implement publish/subscribe using P4. Our approaches make forwarding decisions at the network layer and feature low latency. In the future, we aim to study different real-world networks and intend to develop different strategies for deriving worthwhile forwarding rules tailored to both, the required application scenarios and the topological characteristics of the network.

References

[1] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.

[2] F. Cao and J. Singh. Efficient event routing in content-based publish-subscribe service networks. In *IEEE INFOCOM 2004*, volume 2, pages 929–940. IEEE, 2004.

[3] M. Hungyo and M. Pandey. SDN based implementation of publish/ subscribe paradigm using OpenFlow multicast. In *IEEE Int. Conf. Adv. Netw. Telecommun. Syst. (ANTS)*, pages 1–6. IEEE, 2017.

[4] T. Jepsen, M. Moshref, A. Carzaniga, N. Foster, and R. Soulé. Packet subscriptions for programmable asics. In *ACM Workshop Hot Top. Netw.*, pages 176–183. ACM, 2018.

[5] R. Kundel, C. Gärtner, M. Luthra, S. Bhowmik, and B. Koldehofe. Flexible content-based publish/subscribe over programmable data planes. In *IEEE/IFIP Netw. Oper. Manag. Symp. (NOMS)*, pages 1–5. IEEE, 2020.

[6] D. Merling, S. Lindner, and M. Menth. P4-based implementation of BIER and BIER-FRR for scalable and resilient multicast. *J. Netw. Comput. Appl.*, 169:102764, 2020.

[7] C. Wernecke, H. Parzyjegla, G. Mühl, P. Danielis, E. Schweissguth, and D. Timmermann. Stitching notification distribution trees for content-based publish/subscribe with p4. In *IEEE Conf. Netw. Funct. Virtualiz. Softw. Defin. Netw. (NFV-SDN)*, pages 100–104. IEEE, 2020.

[8] C. Wernecke, H. Parzyjegla, G. Mühl, P. Danielis, and D. Timmermann. Realizing content-based publish/subscribe with P4. In *IEEE Conf. Netw. Funct. Virtualiz. Softw. Defin. Netw. (NFV-SDN)*, pages 1–7. IEEE, 2018.

[9] C. Wernecke, H. Parzyjegla, G. Mühl, E. Schweissguth, and D. Timmermann. Flexible notification forwarding for content-based publish/subscribe using P4. In *IEEE Conf. Netw. Funct. Virtualiz. Softw. Defin. Netw. (NFV-SDN)*, pages 1–5. IEEE, 2019.