Offloading SCION Packet Forwarding to XDP BPF

Lars-Christian Schulz and David Hausheer
Networks and Distributed Systems Lab
Otto-von-Guericke-University
Magdeburg, Germany
{lschulz, hausheer}@ovgu.de

***Abstract*—Existing fixed-function packet processing hardware does not support novel protocols like the next-generation Internet architecture SCION. While SCION routers have previously been accelerated using DPDK and Tofino ASICs, we present an alternative lightweight but feature-complete approach based on the eXpress data path (XDP). We discuss implementation challenges and opportunities unique to the XDP BPF environment and demonstrate a first prototype capable of forwarding at least 0.676 million packets per second per CPU core.**

I. Introduction

As highlighted by DDoS attacks, BGP route hijacking attacks and countless other security incidents in recent years, security and reliability guarantees are hard to come by in today's Internet. In response to the shortcomings of the current Internet, a number of next-generation Internet architectures have been proposed. A particularly promising candidate with an existing real-world deployment is SCION. SCION is a path-aware inter-domain routing architecture founded on the principles of transparency and isolation of trust [1].

Since SCION is a clean-slate approach, no fixed-function switching ASICs are compatible with it. Consequently, most SCION traffic is switched in software, which makes deployment in high-speed networks difficult. In this work, we describe our approach to accelerating SCION routers by means of the eXpress Data Path (XDP) available in the Linux kernel. XDP bypasses the kernel's network stack by attaching BPF programs directly to the network driver [2]. For every packet entering the system, XDP programs have the opportunity to manipulate the packet before the full network stack is invoked.

In addition to the early packet processing capability, XDP offers another kernel-bypass mechanism that forwards packets from the driver directly to a userspace application through a special AF_XDP socket. While the AF_XDP mechanism is a promising alternative to DPDK, this work focuses on processing SCION packets entirely in kernel space.

II. Related Work

The reference SCION border router is implemented in Go and performs all network I/O through the BSD socket interface. Its main focus is not on performance. The Swiss company Anapaya internally employs a DPDK-based SCION border router, which is not publicly available.

There are two noteworthy implementations of the SCION data plane in P4. The first targets the NetFPGA SUME platform through the use of a special P4 to hardware description compiler [3]. It achieves 10 Gbit/s line rate forwarding, but is based on an older version of the SCION specification and is not actively maintained. The second implementation targets the Intel Tofino ASIC [4]. Since Tofino does not offer the cryptographic primitives required by the SCION data plane, any SCION implementation on Tofino is limited to offloading cryptographic packet verification to the control plane. This is possible for the standard SCION protocol, but some advanced protocols built on top of SCION require cryptographic verification of every single packet, rendering workarounds ineffective.

The aim of our work is to explore a middle ground between conventional software implementations of SCION and P4-programmable hardware. XDP programs bypass the kernel as DPDK does, but can – if necessary – run without explicit driver support. Moreover, SmartNICs capable of XDP-hardware offload exist.

III. Design

Our XDP program (available at https://github.com/netsys-lab/scion-xdp-br) is designed to work in conjunction with the regular SCION border router in userspace. The XDP router decides whether a packet can be processed directly in BPF, or whether it has to be forwarded to the conventional router. In this way, we can implement a *fast path* for common cases and leave uncommon types of packets or packets that require special processing to the userspace border router.

Forwarding SCION packets is simplified by the fact that the entire end-to-end path at the AS level is encoded in the SCION header, a concept known as *packet-carried forwarding state* (PCFS). In concrete terms, the SCION header contains a list of so-called hop fields. Each hop field is protected by a Message Authentication Code (MAC) based on the AES-CMAC algorithm. Hop fields within a path segment are chained together via an updatable header field *SegID*, which is included in the MAC computation. From a high level perspective, a SCION border router has to (1) identify the current hop field, (2) verify the hop field, (3) update header fields to point to the next hop field, and (4) forward the packet to the AS egress port indicated in the current hop field. Figure 1 shows the packet processing flow in BPF.

Since XDP programs operate on the raw packet buffers, the first processing step is parsing the outer Ethernet, IP and UDP headers SCION is encapsulated in. If the packet does not contain a valid SCION header it is passed to the regular network stack. When we identify a supported SCION packet, a number of processing steps must be performed on the SCION header, depending on whether the packet reached the system from an external interface (*AS ingress*) and whether the next-hop router is inside another AS (*external*) or the same AS (*internal*). The active hop field in the SCION header indicates the logical interface ID towards which we have to forward the packet. We employ a combination of lookups in SCION specific BPF-maps populated from userspace and a lookup in the kernel's forwarding information base (FIB) to determine which physical interface the packet has to be redirected to and what source/destination UDP ports, IP and MAC addresses have to be used.

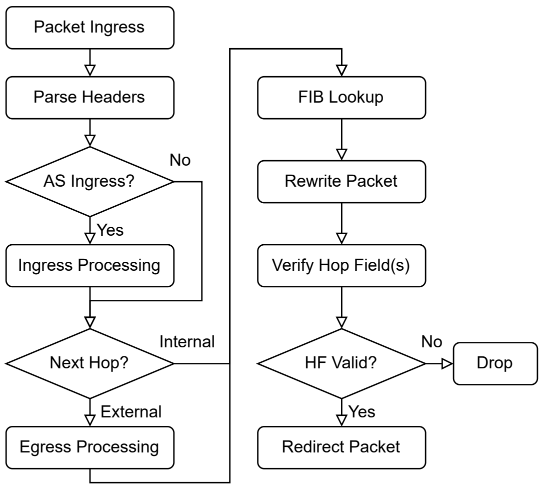

Fig. 1. Simplified XDP packet processing flow.

Modification of the packet buffer is delayed until we can be sure the packet will not have to be passed to user space to avoid corrupting packets.

IV. Implementation

To allow safe execution of user-provided code in kernel space BPF imposes many restrictions on the programs. For instance, all loops must be bounded and all memory accesses have to be guarded by bounds checks. External function calls are limited to a small set of helper functions. BPF-to-BPF function calls are possible, but limited by the small stack size of only 512 bytes as of kernel 5.13. Furthermore, the validator enforcing the rules imposes additional restrictions on the complexity of the code, e.g., the maximum number of branches and the operations permitted on pointers.

The most computationally demanding step of SCION forwarding is the computation of MACs using the AES-CMAC algorithm. Usually, the SCION border router can rely on hardware-acceleration for AES, but this is not possible in BPF. Therefore, we include an implementation of AES in the XDP router. Since the combined program is fairly large, care must be taken to pass the BPF validation. A valuable feature to that end are BPF-to-BPF function calls, because BPF functions are validated independently, effectively resetting the "budget" for each function. However, in many cases it is still preferable to inline functions as not to exhaust the small stack. Moreover, since the validation is reset between function calls, passing pointers between non-inlined functions is difficult.

Many decisions on the structure of our program were dictated by the BPF validator. For example, we check hop field validity last so we can free up stack space used by temporary variables and pointers into the packet buffer before the AES subroutines are invoked, which requires a large amount of stack space. Another well known technique we employ is statically allocating space for large structures in BPF maps which can be arbitrarily large.

In future work, we will explore the possibility of implementing an AES helper function natively in the kernel. Such a function would free us from having to calculate MACs in the BPF VM directly, which could drastically improve performance. Another issue is the inability to rely on receive side scaling (RSS) to distribute work over multiple CPU cores caused by the SCION underlay using a fixed five-tuple per AS-link. By distributing flows using XDP we will be able to harness multiple CPU cores per border router interface.

V. Preliminary Evaluation

To verify whether the XDP border router delivers sensible performance without hardware-accelerated AES, we have created a testbench consisting of two virtual ethernet pairs [5] (veths) bridged by the XDP router. We use `tcpreplay` to inject synthetic SCION traffic into the border router, which forwards the packets to a second virtual interface. The receiving veth pair is configured to count and drop all packets. On an AMD Ryzen 3700X processor running Ubuntu 21.10 inside a VM we have observed a throughput of 0.676 Mpps with AES and of 0.895 Mpps if we disable hop field verification. Additional details are available in our source repository.

VI. Preliminary Conclusions

We presented our approach to accelerate SCION packet processing using XDP and discussed the implementation challenges presented by the BPF runtime environment. Our border router demonstrates the suitability of XDP for rapid prototyping of novel network protocols in the Linux kernel, but we are not yet able to determine whether competitive performance with comparable approaches such as DPDK is achievable.

References

[1] D. Barrera, L. Chuat, A. Perrig, R. M. Reischuk, and P. Szalachowski, "The SCION internet architecture," in *Communications of the ACM*, vol. 60, no. 6, 2017, pp. 56–65.

[2] T. Høiland-Jørgensen, J. D. Brouer, D. Borkmann, J. Fastabend, T. Herbert, D. Ahern, and D. Miller, "The express data path: Fast programmable packet processing in the operating system kernel," in *CoNEXT*, 2018, p. 54–66.

[3] K. Součková, "FPGA-based line-rate packet forwarding for the SCION future internet architecture," Master's thesis, ETH Zürich, 2019.

[4] J. de Ruiter and C. Schutijser, "Next-generation internet at terabit speed: SCION in p4," in *CoNEXT*. New York, NY, USA: Association for Computing Machinery, 2021, p. 119–125.

[5] T. Makita and W. Tu, "Veth XDP: XDP for containers," 2019, netdev 0x13. [Online]. Available: https://legacy.netdevconf.info/0x13/session.html?talk-veth-xdp