Bringing Packet Queueing to XDP

Freysteinn Alfreðsson
freysteinn.alfredsson@kau.se
Department of Computer Science
Karlstad, Sweden

Per Hurtig
per.hurtig@kau.se
Department of Computer Science
Karlstad, Sweden

Anna Brunstrom
anna.brunstrom@kau.se
Department of Computer Science
Karlstad, Sweden

Toke Høiland-Jørgensen
toke@redhat.com
Red hat
Denmark

Jesper Dangaard Brouer
jbrouer@redhat.com
Red hat
Denmark

ABSTRACT

The Linux eXpress Data Path, or XDP, has found numerous uses in the industry, such as mitigating DoS attacks, load-balancers, and intrusion prevention systems. XDP provides a high-performance programmable network data path using the BPF framework and allows programmers to process packets early out of the driver. While XDP excels in forwarding packets, it currently has no mechanism for queueing or reordering packets and cannot implement traffic scheduling policies. In this paper, we present our ongoing work to address this challenge. We have designed a programmable packet scheduling extension for the XDP framework using recently proposed schemes for programmable queues. This extension allows programmers to define their packet schedulers using BPF while benefiting from the XDP fast data path.

KEYWORDS

XDP, BPF, eBPF, Queueing, Linux, Scheduling, Programmable scheduling

1 INTRODUCTION

The Linux kernel is a widely used platform in devices that range from IoT, cellphones, home and enterprise routers to servers and cloud offerings. One of the prominent technologies of the Linux kernel is the BPF [1] framework [2], which gives us a flexible way to extend the kernel. The BPF framework is an in-kernel runtime environment that allows domain-specific code to execute within the kernel safely using predefined hooks. One such hook is the Linux eXpress Data Path [7], or XDP, which introduces a fast datapath within the Linux kernel and has found numerous uses in the industry, such as mitigating Denial-of-Service [6] [8] (DoS) attacks, load-balancers, and intrusion prevention systems.

XDP provides a high-performance programmable network data path and allows programmers to process packets early out of the driver and even bypass the kernel's network stack for increased performance. While XDP excels in forwarding packets, it currently has no mechanism for queuing or reordering packets and can not implement traffic scheduling policies, pacing, or shaping. Packet scheduling is a common task on network equipment, and most devices that provide packet scheduling algorithms provide only a handful of predefined packet schedulers. The Linux kernel's network stack provides a traffic control system called Queuing discipline (Qdisc) capable of packet scheduling. However, it does not provide the means of implementing complete packet schedulers in BPF, nor can it be used in XDP when bypassing the kernel's networking stack. Therefore, a new extension to XDP is needed.

This paper presents our work on adding programmable packet scheduling to XDP. We have designed a programmable packet scheduling framework in BPF using recently proposed schemes for programmable queues. This extension allows programmers to define their packet schedulers using BPF while benefiting from the XDP fast data path.

The rest of the paper is structured as follows. In Section I, we present a short overview of packet schedulers. Section II gives an overview of the Linux Kernel's networking subsystem and the need for solutions like XDP. Section III explains programmable packet schedulers and introduces the PIFO, a fast and generalized data structure to express most schedulers. Section IV gives a short overview of the BPF framework and XDP. Section V introduces our contribution, a packet scheduling framework in BPF. Finally, sections VI, VII, and VIII conclude our paper with future work, the conclusion, and acknowledgments.

2 THE LINUX KERNEL NETWORKING SUBSYSTEM

The Linux kernel networking stack is highly mature and a flexible piece of software. It contains a hardware abstraction layer, a traffic control layer responsible for routing and firewall rules, and layers responsible for handling the TCP/IP networking layers. Part of the traffic control layer is the queueing discipline layer or the Qdisc layer for short. This layer gives the Linux kernel flexible packet scheduling capabilities and an interface to create new packet schedulers as loadable kernel modules. However, due to the immense speed increases in modern network devices, the kernel's networking stack has become a bottleneck. This obstacle has prompted network vendors and researchers to create alternative solutions that, in various degrees, bypass the networking stack. One such solution is the Data Plane Development Kit [1] (DPDK), which entirely bypasses the Linux kernel and communicates directly to the networking hardware. An alternative solution to completely bypassing the Linux kernel is XDP (described in Section 4.1). XDP creates a fast data path within the Linux kernel while retaining the kernel and user-space separation. Figure 1 provides a simplified diagram

[1] The BPF framework used to be called eBPF, but today it is referred to as BPF and its predecessor cBPF.

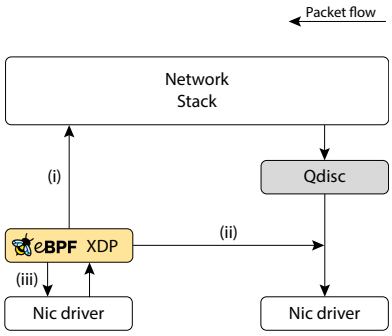

Figure 1: XDP allows network application programmers to write code that runs on every network packet using the BPF framework. XDP gives the programmer the capability to modify the packet and redirect it using the code's return value. The possible return values are as follows: (i) passing the packet to the kernel's networking stack; (ii) redirecting the packet to another network device; (iii) send the packet directly out on the same device; or (iv) dropping the packet altogether.

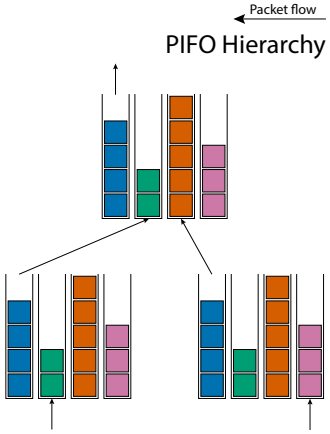

Figure 2: We can implement complex packet scheduling algorithms by forming hierarchies of PIFO data structures. Some packet schedulers must use hierarchies to function, but we can also use hierarchies to combine different schedulers to handle complex requirements.

of the Linux kernel's networking infrastructure depicting the relevant parts related to our work in this paper. The diagram shows how XDP can steer traffic through the network stack or bypass it altogether.

3 PROGRAMMABLE PACKET SCHEDULERS

Traditionally, network equipment and operating systems provide a handful of packet scheduling algorithms. This limited number of schedulers forced operators to tune the parameters of these algorithms to meet their requirements. However, modern networking equipment has started to provide programmable network capabilities, which a network engineer can leverage using specialized programming languages such as P4 [3] or standard programming languages such as C, as in the case of traditional operating systems. This new offering allows network designers to create custom logic and customized packet scheduling solutions optimized for their unique use case.

Consequently, the building blocks of a programmable packet scheduling framework need to be straightforward, fast, and easy to understand. Therefore, the essential part of creating a programmable packet scheduling framework is to provide a flexible data structure. Subsequently, this has become an active research area to develop a flexible data structure that programmers can leverage to implement their packet schedulers. Nevertheless, due to the vast speeds of modern network interfaces, severe time limitations are put on these data structures, which tend to make generic priority queues, such as red-black trees and binary heaps, unsuitable choices.

In this paper, we focus on the Push-In First-Out (PIFO) priority queue [10] and the Eiffel [9] extension to PIFO, described in Section 3.1 and Section 3.2. The PIFO is a simple data structure that has the following qualities: (i) it uses an integer-based ranking function to queue packets by their priority and relies on a fixed range of ranks known at initialization time; and (ii) it dequeues packets according to the scheduled rank order. Furthermore, it has the flexibility to create complex packet scheduling algorithms by using hierarchies, as seen in Figure 2. The Eiffel extension allows scheduling decisions on dequeue, while a traditional PIFO only makes scheduling decisions on enqueue. This distinction stems from the fact that PIFOs are well suited for hardware and software. However, Eiffel is software-only and is capable of more complex operations, such as flow-based scheduling, which requires scheduling on dequeue.

3.1 PIFO

The Push-In First-Out (PIFO) data structure [10] is a priority queue where the programmer can queue packets based on their rank, where each rank is queued in a FIFO order, as shown in Figure 3. However, the programmer can only dequeue from the head of the PIFO and only order packets on enqueue. Also, the PIFO does not allow reordering the packets within a FIFO. This limitation inhibits the PIFO from implementing some packet scheduling algorithms, such as pFabric [5].

Despite its simplicity, the PIFO data structure is quite versatile and allows the programmer to implement various packet schedulers. From a programmer's perspective, the programmer only needs to decide what order to schedule packets and when to schedule them for non-work conserving algorithms. However, one limitation of the PIFO is that each FIFO, and therefore, each rank, consumes memory. Consequently, the more granular ranks, the more resources the PIFO needs. This limitation limits how large the PIFO can be in hardware and software implementations. A mitigation to this limitation is the SP-PIFO [4], which approximates a large PIFO using a smaller PIFO.

3.2 Eiffel extension to PIFO

Eiffel [9] is a software-only extension to the PIFO data structure and adds a couple of alterations to the traditional PIFO. First, with

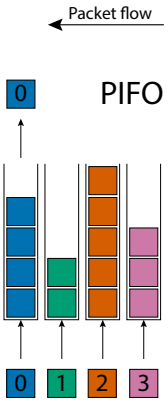

Figure 3: The PIFO data structure is a priority queue where the programmer enqueues packets by rank. However, the dequeue operation can only dequeue packets from the top of the structure. Hence, the lower priority queues will starve if the highest priority queue keeps getting packets without being emptied.

Eiffel, the programmer can schedule flows and packets, depending on the algorithm, where each flow is a separate FIFO that contains only the flow's packets. The Eiffel extended data structure can schedule references to those FIFOs instead of the individual packets. Second, the programmer can do on-dequeue scheduling; this allows the algorithm to virtually rearrange all the packets from the same flow by simply requeueing the reference to that flow back into the Eiffel extended PIFO. These two additions allow the Eiffel PIFO to represent more scheduling algorithms than a traditional PIFO, such as pFabric [5].

Also, because Eiffel is software-only, the Eiffel paper proposes a few software-based optimizations to improve the performance of the Eiffel data structure. One such optimization is to use a bit-field to keep track of queues containing packets and use the CPU's Find-First-Set [2] instruction to accelerate the lookup in the bit lookup table. The lookup table can be further segmented into a tree-based lookup table when the Eiffel has many ranks.

4 THE BPF FRAMEWORK

One of the problems with extending the Linux kernel with specialized code, such as packet schedulers, is writing the extensions as kernel modules. From a security perspective, the modules have access to the entire kernel without limits, making it more prone to programming errors, such as pointer mishandling and infinite loops. The kernel community provides BPF as a safer way of extending the kernel. This framework is an in-kernel runtime environment that allows specialized programs to execute in the kernel safely. To support BPF, kernel programmers add appropriate BPF hooks to the kernel to offer domain-specific extendability to programmers. These hooks allow the programmers to attach BPF code that runs each time these hooks are triggered. The Linux kernel comes with numerous hooks related to networking, tracing, and security, where each hook limits what the BPF code can access and which helper functions it can call.

Underneath, BPF programs are binaries written in the BPF instruction set that can be loaded using the bpf system call into the kernel. The BPF instruction set and runtime are deliberately limited in functionality. It supports a handful of instructions, and the runtime only has a 512-byte stack. When the kernel loads the binary, it runs it through a BPF verifier that makes sure that all pointers are within bounds, that the program does not call disallowed functions, and that the code cannot run more than a million instructions. After verifying the code, the user-space application can attach the BPF program to the desired hook.

A salient feature of BPF is that it offers interprocess communication using BPF maps. These maps are key-value stores that act as global variables within the framework and are the primary method to interact with BPF programs. They come in different types, such as arrays and hashmaps, and some of the more advanced maps offer per-CPU storage for additional performance. They provide a common way for different BPF hooks and user-space applications to communicate with each other.

4.1 XDP

Express Data Path (XDP) [7] is a type of BPF hook that resides early in the RX path of a network interface to allow the programmer direct manipulation of packets from the network driver. Its primary use cases are high-performance packet processing and the ability to bypass the kernel's network stack by redirecting the packet to different locations, such as back out of the same device, to another device, or dropping the packet entirely. XDP also provides helper functions that allow the programmer to call particular parts of the network stack, such as routing lookups.

5 A PROGRAMMABLE PACKET SCHEDULING FRAMEWORK IN BPF

While XDP excels at forwarding packets, it does not support packet reordering or scheduling. Our contribution is the extension of XDP to support packet scheduling in BPF using PIFO data structures with Eiffel capabilities. This support includes adding helper functions, a new PIFO BPF map data type, and an extra hook for dequeuing packets from the PIFO data structure. Figure 4 show the design and basic building blocks of our new programmable packet scheduling framework without the specifics of XDP or the Qdisc subsystem.

A more specific description of each component of this new addition is as follows:

- PIFO map: This new BPF map implements a PIFO and is the main building block for the programmer to create packet schedulers. The framework provides two variants of this map, a PIFO that can store network packets and a PIFO that can store any data type, including other PIFOs. The second data type gives the programmer a convenient way of creating PIFO hierarchies and scheduling flows instead of packets. These maps come with new helper functions to enqueue and dequeue new packets, and a peek function for the generic PIFO map.

[2]The Find-First-Set machine instruction finds the first set bit in a CPU register.

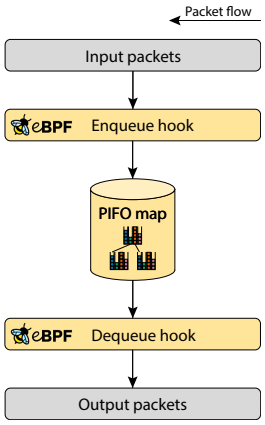

Figure 4: The diagram shows the flow of packets through our proposed BPF-based programmable packet scheduling framework. The framework adds a new dequeue hook to the Linux kernel and a PIFO map. For a fully functional scheduler, the XDP hook must redirect packets to a PIFO, and the dequeue hook must dequeue packets to an interface.

- Enqueue hook: This is the standard XDP hook, and in addition to the original capabilities of XDP, it is now capable of redirecting packets to our new PIFO maps.
- Dequeue hook: This new hook is responsible for delivering the packet from the packet scheduling algorithm. While transmitting individual packets from the dequeue hook is slow, it is possible to transmit packets in bulk by calling the hook multiple times before triggering the transmit procedure from the interface.

Representing the PIFO data structure as a BPF map presents programmers with a familiar map interface that they have come accustomed to in BPF. From a programmer's perspective, the BPF hooks reference the queues like any other map type. The enqueue hook decides which queue to direct the packet to by a map reference. Similarly, the dequeue hook picks which queue map to dequeue from and returns a reference to the dequeued packet to the kernel for transmission.

6 FUTURE WORK

Our current implementation gives XDP the capability to schedule packets; however, it is still missing the capability to implement pacing and other traffic shaping features. We are working on adding pacing to our packet scheduling framework by using the newly added BPF timers, which will allow the programmer to trigger time-based packet dequeues from PIFOs. These timers allow the PIFO dequeue hooks to enqueue the packets into other PIFOs that dequeue into live interfaces or other timer-triggered PIFOs—giving our framework pacing capabilities.

While we believe that scheduling in XDP will benefit today's high-speed environments, we are still in the early stages of our implementation to do any benchmarks. Future benchmarks that we want to conduct are comparing the performance of the PIFO map structure against the Linux kernel's standard red-black trees and comparing our implementation to the existing kernel Qdisc layer.

7 CONCLUSION

Current networking technologies have reached speeds that have pushed the industry and researchers to explore alternative solutions to remove bottlenecks in the networking stack. One of these solutions is XDP, which allows programmers to bypass the kernel. However, XDP did not have a way to rearrange or schedule packets. In our work, we have created a packet scheduling framework for XDP, which allows the programmer to write domain-specific packet schedulers while retaining all the benefits of XDP.

8 ACKNOWLEDGMENTS

We thank our anonymous reviewers and Simone Ferlin for their feedback and suggestions that helped improve this paper. We also like to thank Red Hat Research which funded this work.

REFERENCES

[1] [n.d.]. Data Plane Development Kit. https://www.dpdk.org/
[2] [n.d.]. extended BPF. https://lwn.net/Articles/599755/
[3] [n.d.]. P4 open source programming language. https://p4.org/
[4] Albert Gran Alcoz, Alexander Dietmüller, and Laurent Vanbever. 2020. SP-PIFO: Approximating push-in first-out behaviors using strict-priority queues. *Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020* (2020), 59–76.
[5] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. 2013. pfabric: Minimal near-optimal datacenter transport. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 435–446.
[6] Gilberto Bertin. 2017. XDP in practice: integrating XDP into our DDoS mitigation pipeline. In *Technical Conference on Linux Networking, Netdev*, Vol. 2.
[7] Toke Høiland-Jørgensen, Jesper Dangaard Brouer, Daniel Borkmann, John Fastabend, Tom Herbert, David Ahern, and David Miller. 2018. The express data path: Fast programmable packet processing in the operating system kernel. In *Proceedings of the 14th international conference on emerging networking experiments and technologies*. 54–66.
[8] Sebastiano Miano, Roberto Doriguzzi-Corin, Fulvio Risso, Domenico Siracusa, and Raffaele Sommese. 2019. Introducing SmartNICs in server-based data plane processing: The DDoS mitigation use case. *IEEE Access* 7 (2019), 107161–107170.
[9] Ahmed Saeed, Yimeng Zhao, Nandita Dukkipati, Mostafa Ammar, Ellen Zegura, Khaled Harras, and Amin Vahdat. 2019. Eiffel: Efficient and flexible software packet scheduling. *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019* (2019), 17–31. arXiv:1810.03060
[10] Anirudh Sivaraman, Suvinay Subramanian, Mohammad Alizadeh, Sharad Chole, Shang Tse Chuang, Anurag Agrawal, Hari Balakrishnan, Tom Edsall, Sachin Katti, and Nick McKeown. 2016. Programmable packet scheduling at line rate. *SIGCOMM 2016 - Proceedings of the 2016 ACM Conference on Special Interest Group on Data Communication* (2016), 44–57. https://doi.org/10.1145/2934872.2934899