Building the Next Generation of Programmable Networking – Powered by Linux

Frey Alfredsson, Simon Sundberg, Per Hurtig, Anna Brunstrom,

Toke Høiland-Jørgensen, Simone Ferlin-Reiter, Jesper Dangaard Brouer

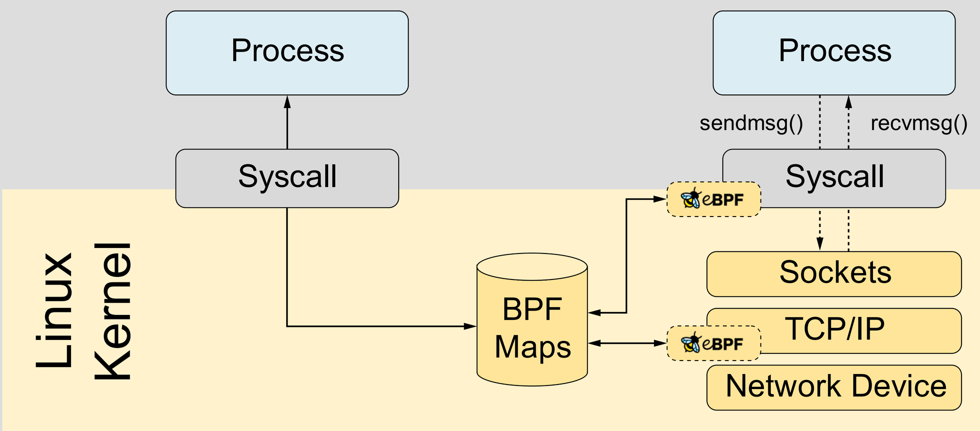
BPF unlocks the kernels potential for innovation

BPF enabling a programmable network in Linux

- Programmable networks allow innovative new network applications/features/functionality.
  - Can quickly implement new network functionality in code.
  - Can run on off-the-shelf hardware.
  - Can easily deploy updates, no need to reinstall hardware.
- Different options for adding new network functionality on Linux.
  - Modify the kernel: limited performance, long time to get it upstreamed and widely deployed, need to re-compile kernel.
  - DPDK: Hard to integrate with existing functionality in the Linux network stack.
  - BPF: Modify the existing network stack live in a safe and performant manner.
- Thanks to the eXpress Data Path (XDP), we can use BPF for custom packet handling.

The Linux kernel's BPF framework

- BPF allows programmers to attach BPF program code into predefined hook points in the Linux kernel. These hook points allow programmers to create specialized kernel code that runs safely and reliably within the kernel without recompiling code or writing kernel modules.
- BPF is an entire ecosystem with an in-kernel runtime environment, assembly language, compilers, tools, and libraries.
- The BPF framework provides inter-process communication between user-space and BPF programs using BPF maps. These maps are predefined data structures with different functionalities, such as arrays, hashmaps, per CPU arrays, and more.
- One of the primary BPF hooks we work with in our research is called XDP. It provides the means to attach BPF code in the RX path of the network interface, which allows the programmer to drop, alter, and monitor packets. It also provides the means of bypassing the network stack entirely by forwarding packets to different network interfaces.

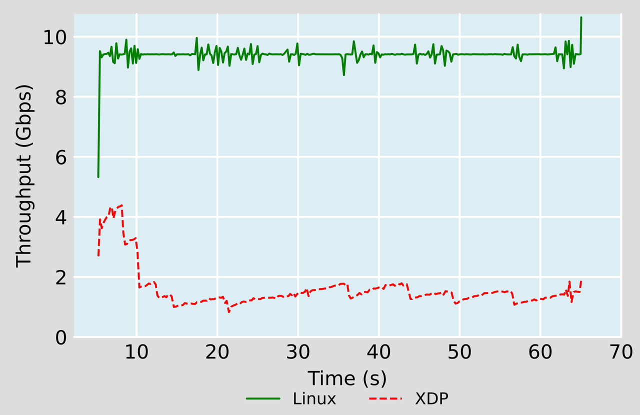


Our research

- Red Hat and Karlstad University are collaborating in a project focused on using BPF/XDP for programmable networking in Linux.
- Two directions explored:
  - Adding queue management to XDP.
  - Using BPF to efficiently monitor network latency.

Adding queue management to XDP

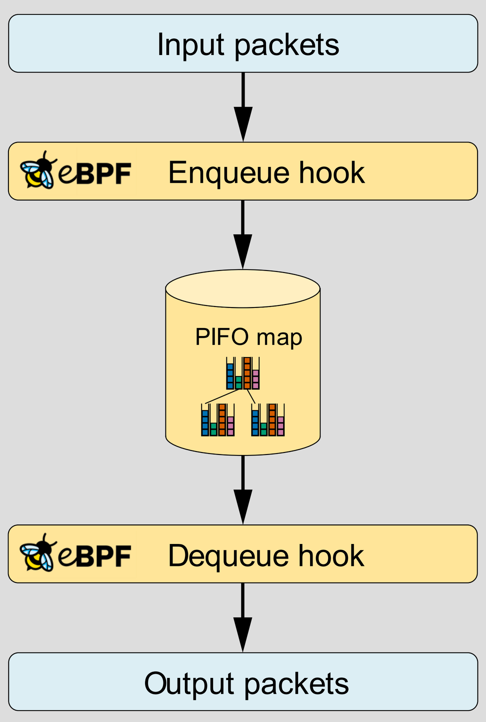
- A limitation of XDP is that while XDP excels in forwarding packets, it currently lacks:
  - A mechanism for queueing or reordering packets.
  - A way of implementing traffic scheduling policies.

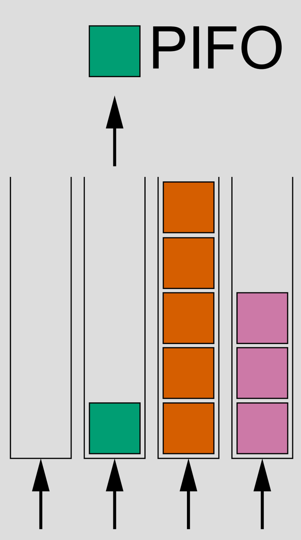


- The shallow buffers of XDP can work against network congestion control in protocols such as TCP. The figure above shows an example of this issue using XDP and the Linux kernel's network stack. The example uses a 100 Gbps interface that transmits to a server with a 10 Gbps link, and they have a simulated 10ms propagation delay.

Contribution

- Our contribution is the extension of XDP to support queuing and packet scheduling using BPF with the following:
  - A new priority queue BPF map modeled on the Push-In First-Out[1] (PIFO) data structure.
  - New BPF helper functions.
  - And a new dequeue BPF hook.
- The PIFO data structure gives programmers the following:
  - A flexible priority queue capable of implementing most scheduling algorithms, where hierarchies of PIFOs form more complex scheduling algorithms.
  - A clock capability via extensions to the PIFO called Eiffel[2], proposed by Saeed et al.





For questions, please reach out to freysteinn.alfredsson@kau.se

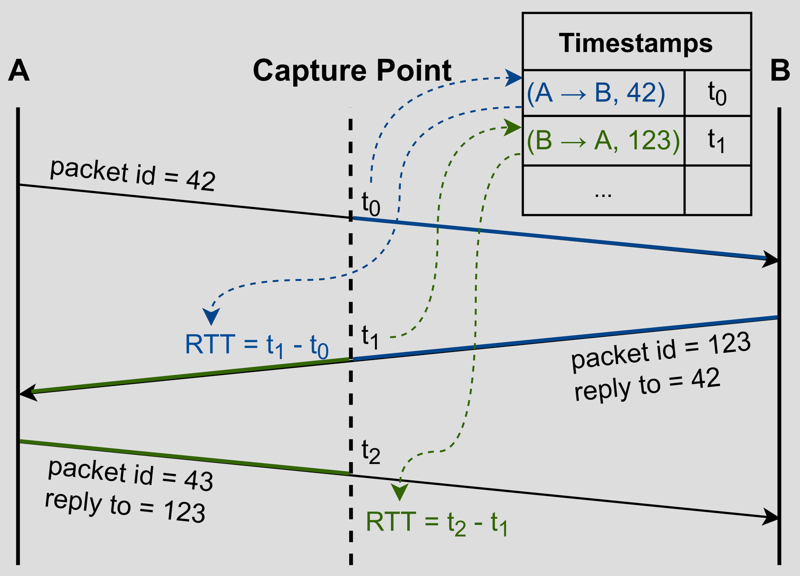
References:

[1] A. Sivaraman et al. "Programmable packet scheduling at line rate," SIGCOMM 2016 - Proceedings of the 2016 ACM Conference on Special Interest Group on Data Communication, pp. 44–57, 2016.

[2] A. Saeedet al. "Eiffel: Efficient and flexible software packet scheduling," Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019, pp. 17–31, 2019.

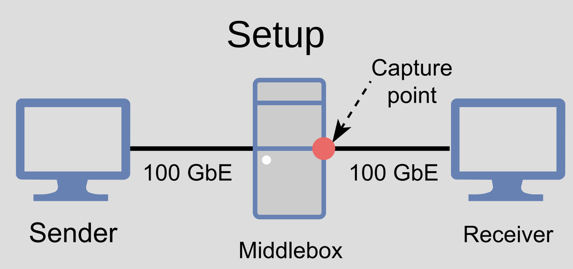
Using BPF to efficiently monitor network latency

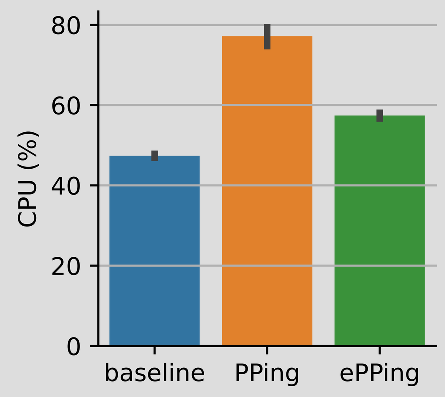
- Limitations with existing network latency monitoring tools:
  - Active monitoring like ping: fail to capture RTT of application traffic.
  - Passive monitoring like Wireshark and PPing [3]: rely on packet capturing – inefficient.
- Use BPF to efficiently passively monitor packets.
  - ePPing – evolved Passive Ping: Implement logic of PPing in kernel space using BPF.
  - Avoids overhead of packet capturing without modifying or bypassing the kernel.

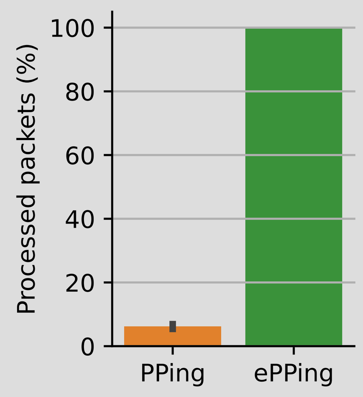


- BPF programs at capture point parse all packets.
  - Saves identifier and timestamp of packets.
  - Match replies with previous packets in reverse direction.
  - Calculate RTT as time difference between original packet and reply.
- Capture point can be any device which sees the traffic in both directions.

Select Results







- Middlebox forwarding 10 TCP flows from sender at ~55 Gbps.
  - Baseline shows middlebox performance when only forwarding (no monitoring).
  - CPU overhead of ePPing ~1/3rd of PPing (10% vs 30%-units above baseline).
  - ePPing runs on every packet, PPing cannot keep up and only handles ~6% of packets.

For questions, please reach out to simon.sundberg@kau.se

References:

[3] K. Nichols. 2018. "Passive ping network monitoring utility". https://github.com/pollere/pping

Project Contacts

Karlstad University
Anna Brunstrom: anna.brunstrom@kau.se

Red Hat
Toke Høiland-Jørgensen: toke@redhat.com

Acknowledgements

This project is a collaboration between Karlstad University and Red Hat Research; with additional funding from the Knowledge Foundation of Sweden. For more information and additional resources, see the project page.

Project page