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.

Adding queue management to XDP

- A limitation of XDP is that while XDP excels in forwarding packets, it currently lacks:
  - A mechanism for queuing 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 10Gbps 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 (PIFO) data structure.
  - New BPF helpers functions.
  - 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 Eiffle[2], proposed by Saeed et al.

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

References:


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.

**Select Results**

- BPF programs at capture point parse all packets.
  - Saves identifier and timestamp of packets.
  - Match replaces 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.

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

References:


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 Contacts**

**Karlstad University**
Anna Brunstrom: anna.brunstrom@kau.se
Toke Høiland-Jørgensen: toke@redhat.com

**Red Hat**

For more information and additional resources, see the project page.

For more information and additional resources, see the project page.