Lightweight Always-on Network Latency Monitoring with eBPF

Simon Sundberg, Anna Brunström, Simone Ferlin-Reiter,
Toke Høiland-Jørgensen & Jesper Dangaard Brouer

Agenda

- Why network latency matters
- Problems with existing network latency monitoring tools
- What eBPF is and how it can solve these problems
- Results and future work

Why monitor network latency?

- Interactive applications are latency sensitive
  - Tactile Internet
  - Video conferencing / VoIP
  - Gaming
  - Browsing / Web shopping

- To monitor QoE → monitor latency
  - Also useful for SLA validation, network management, attack detection etc.

What’s wrong with ping?

- Many latency monitoring tools rely on active probing
  - (f/h/n)ping, IRTT, netlatency, RIPE Atlas
- Drawbacks:
  - Adds network overhead
  - Only monitors between agents
  - Not indicative of application traffic

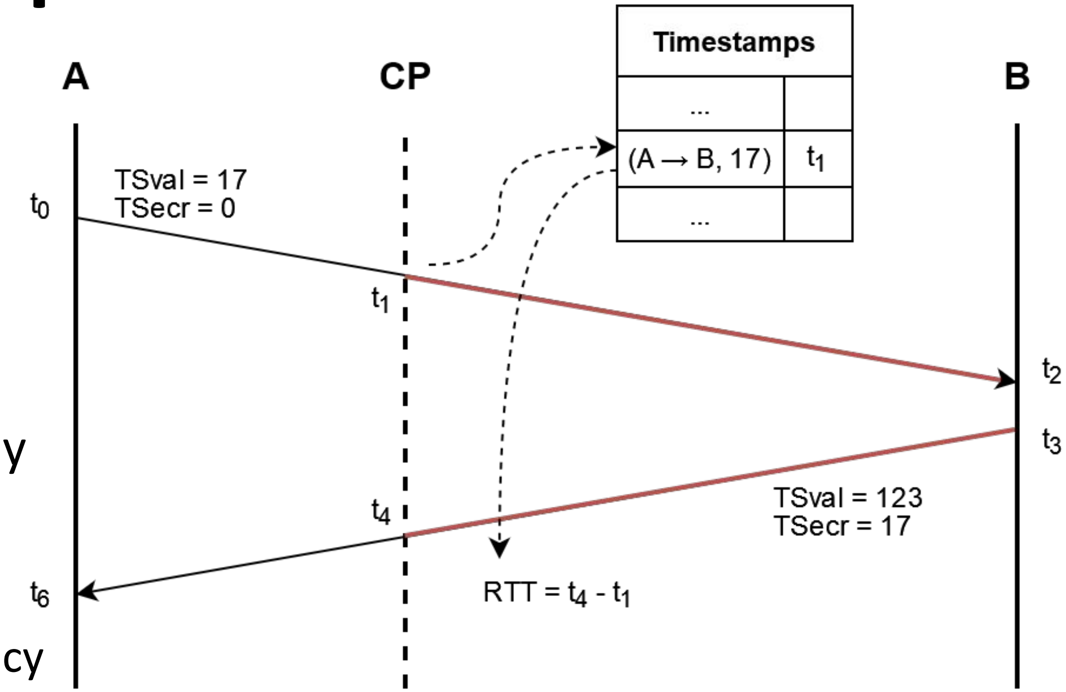
Does it take the same time for cars and scooters to move through a traffic jam?

Photo by Ferliana Febritasari

Passively measure RTT

- Infer RTT from real traffic
  - Match packet and replies

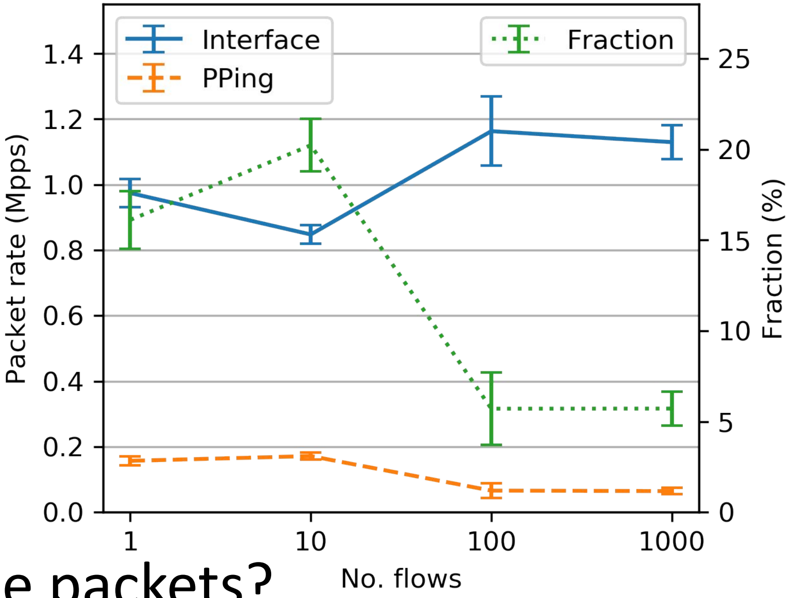
- Passive Ping (PPing)
  - Can run live and continuously report RTTs
  - Uses TCP timestamps
    - Updated at limited frequency



PPing available at https://github.com/pollere/pping

So what's wrong with PPing then?

- Packet capturing has high overhead
  - Can't keep up with high packet rates
- PPing consequences
  - Misses RTT samples
  - May missmatch packets



- What if we didn't need to capture the packets?
  - With eBPF we can peek at packets in the kernel

What is eBPF?

- Runtime environment in kernel
  - Attach small programs to various hooks

- Use cases
  - Observability, Security, Networking

- Workflow
  - Compile to eBPF bytecode
  - Load into kernel
    - ○ Verified
    - ○ Jitted
  - Attach to hook

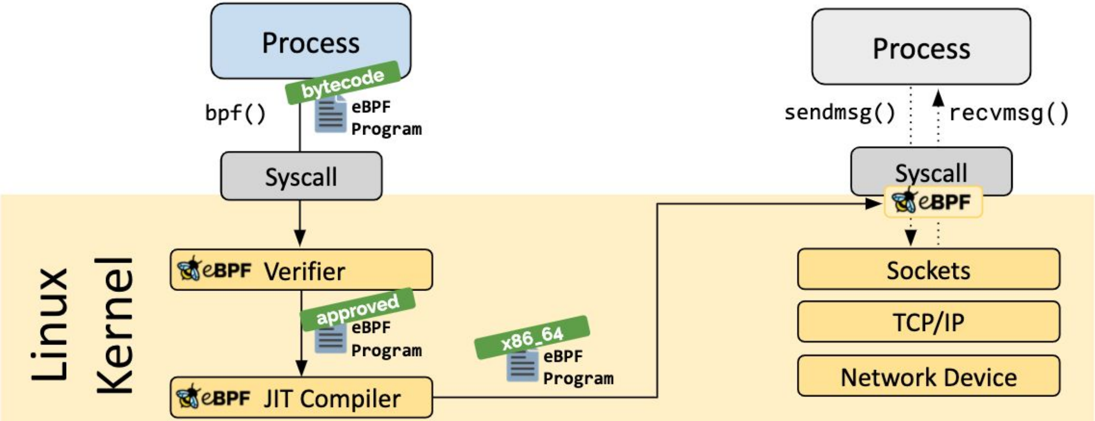


Figure from https://ebpf.io/what-is-ebpf

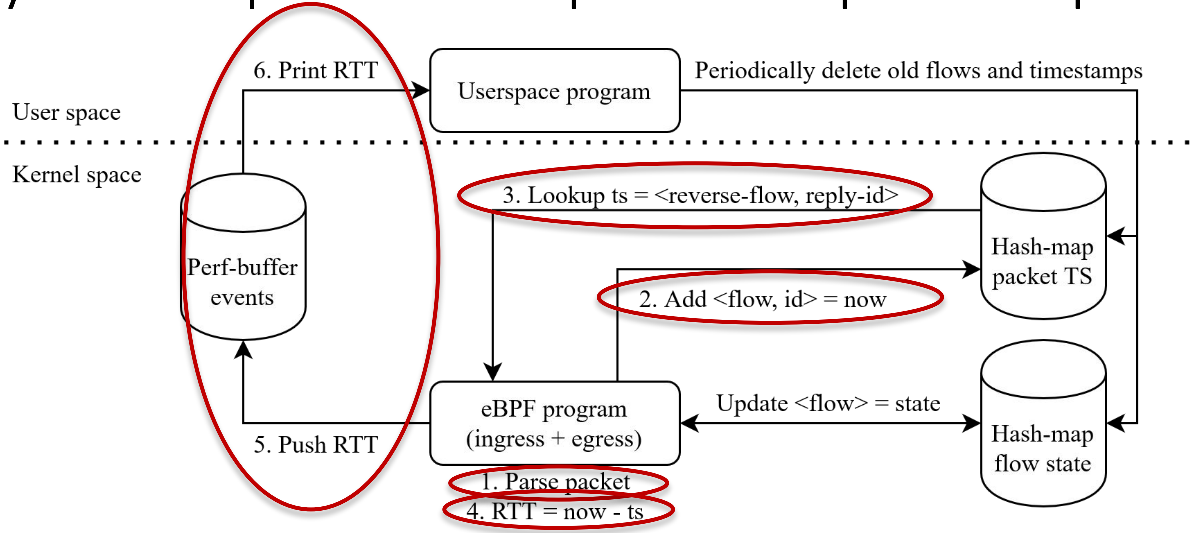
What is XDP and tc BPF?

- eXpress Data Path (XDP)
  - Ingress hook at the earliest part of network stack
- Traffic control (tc)
  - Ingress or egress hook inside the network stack

- Hooks that enable a programmable data plane in the Linux kernel
  - Can inspect and modify packets
  - Take actions such as accepting, dropping and redirecting packets

An evolved PPing (ePPing)

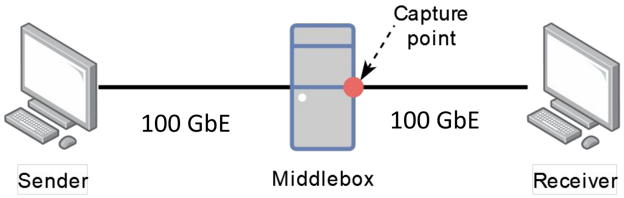
- Implement all packet processing logic in eBPF
  - Only send computed RTT samples to userspace for reporting



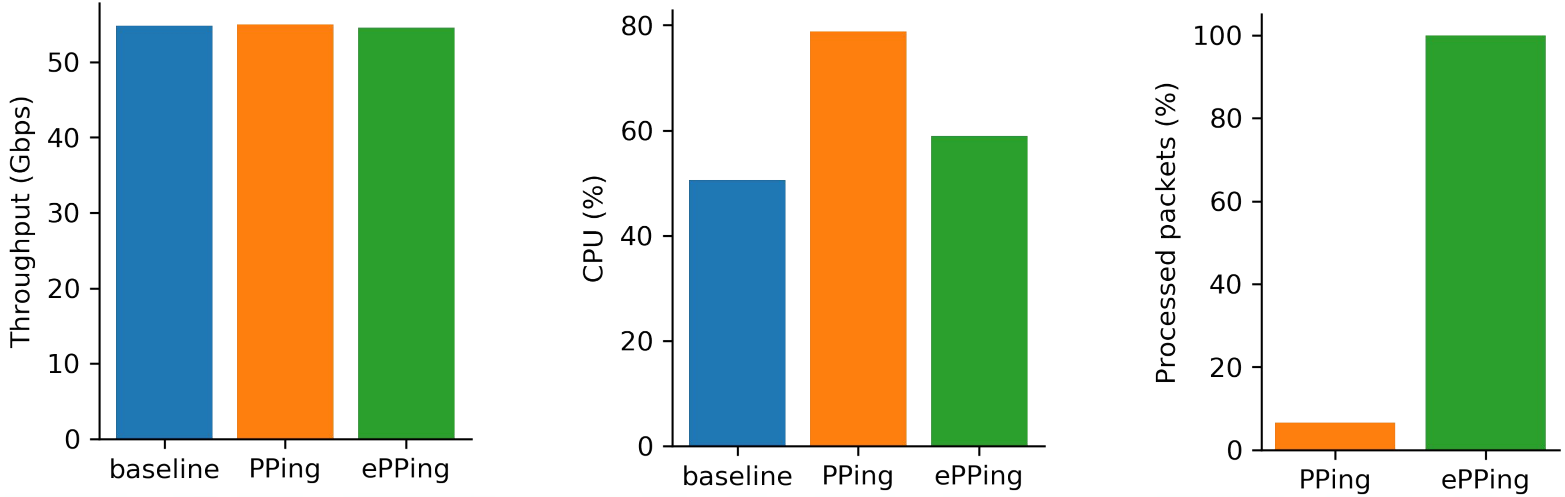
ePPing available at https://github.com/xdp-project/bpf-examples/tree/master/pping

How does it perform?

- Setup:

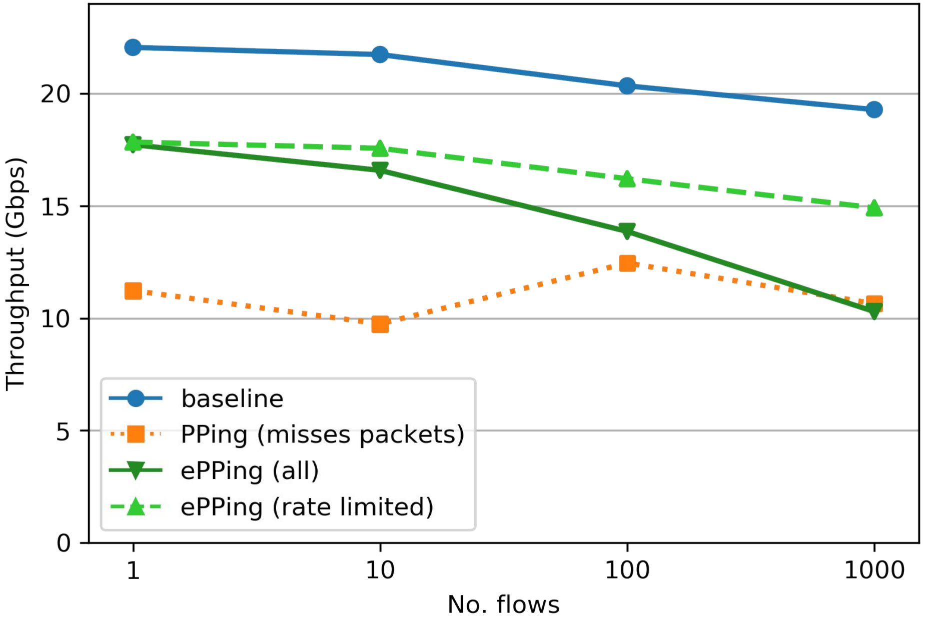


- 10 Iperf flows:



Performance in bottlenecked scenario

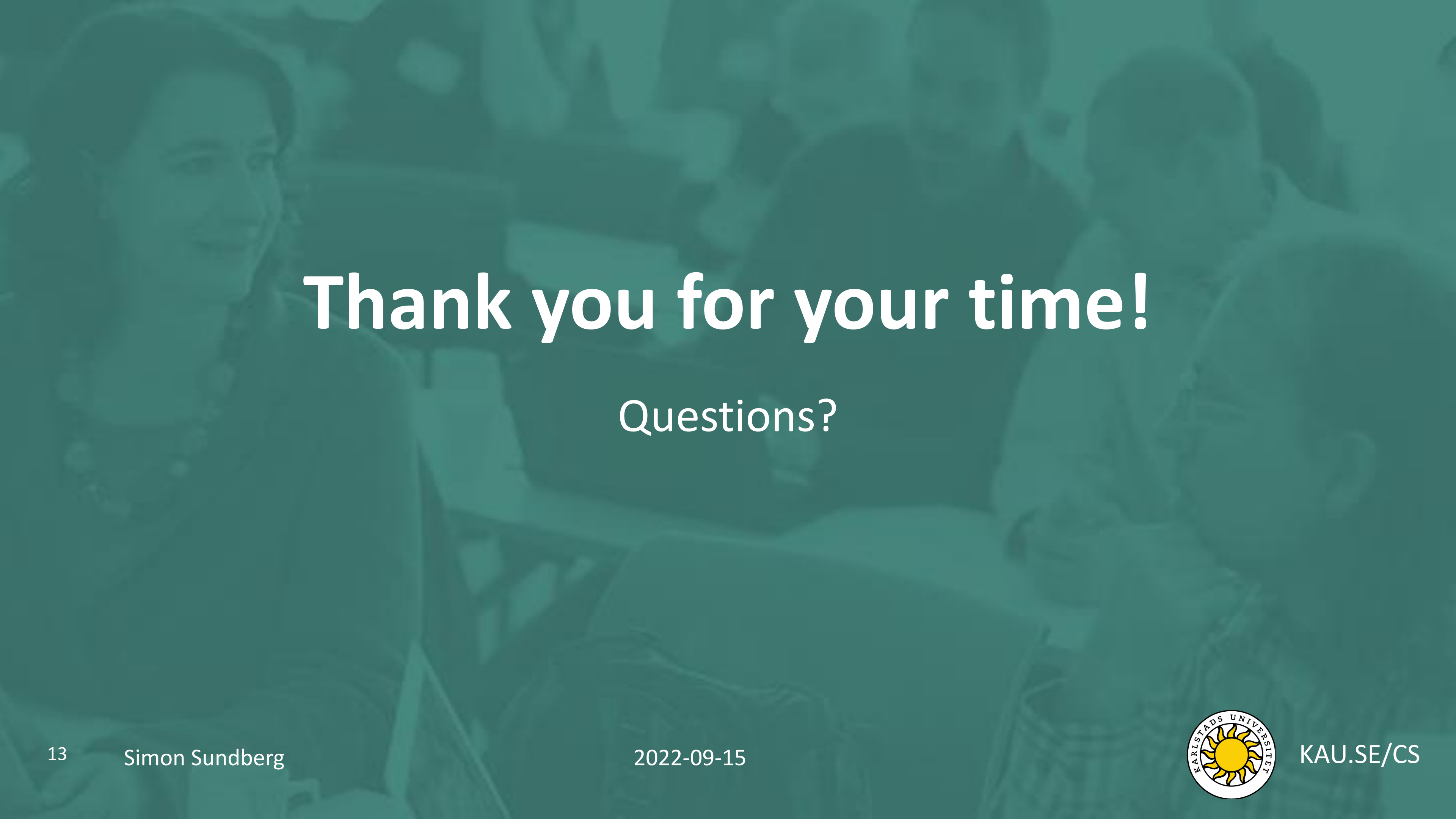
- Limit CPU to single core
  - Core is 100% utilized
- ePPing vs no. of flows
  - More flows → more RTTs
  - Reporting all RTTs has high overhead
  - Sampling RTTs per flow reduces overhead



Conclusion and future work

- We have:
  - Implemented passive latency monitoring in eBPF
    - Can run on any Linux device which sees the traffic
    - Measures RTT live and continuously
  - Tested ePPing's performance
    - Can handle 10+ Gbps on single core

- We want to:
  - Improve reporting of RTT by sampling/aggregating
  - Add support for additional protocols (QUIC, DNS)



Thank you for your time!

Questions?