Efficient continuous latency monitoring with eBPF

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

Network latency matters

- Latency impacts QoE of interactive applications
  - Current applications: video conferencing, gaming, web browsing
  - Future applications: AR/VR, tactile Internet, autonomous vehicles

- Need tools to continuously monitor latency
  - Latency can rapidly change on a network
  - Latency within a flow can fluctuate (jitter)
  - To solve latency issues we must first monitor the latency

Current solutions for latency monitoring

- Active monitoring
  - Ex. Ping, IRTT, pingmesh, RIPE Atlas
  - Great for controlled measurements
  - Don't capture latency of actual application traffic

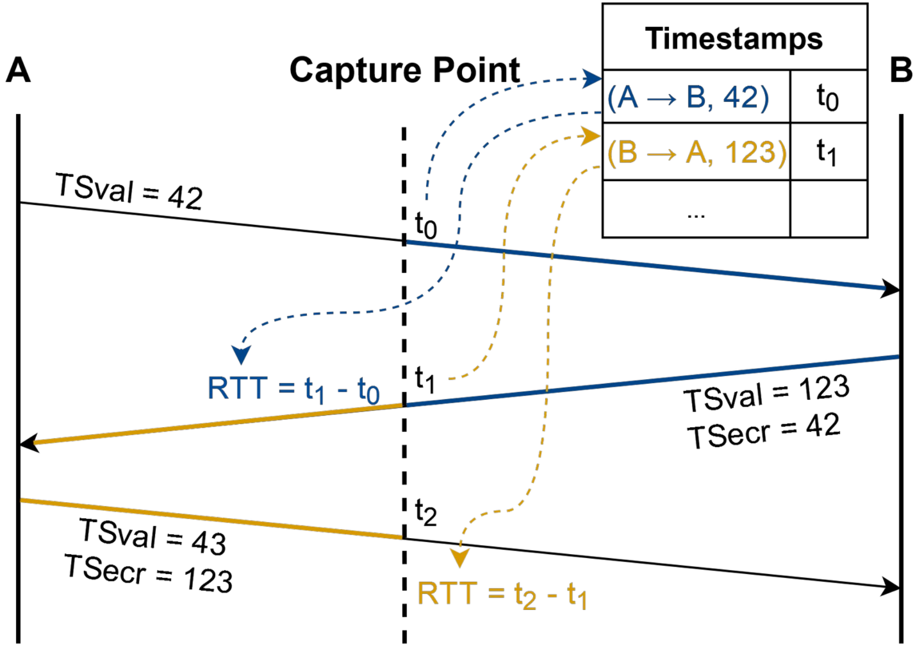
- Passive monitoring
  - Ex. Wireshark/tshark, PPing[1]
  - Captures latency of real application traffic
  - High overhead from packet capturing

[1] https://github.com/pollere/pping

How Passive Ping works

- Uses TCP timestamps
  - Matches TSval and TSecr
  - Can be extended to other identifiers

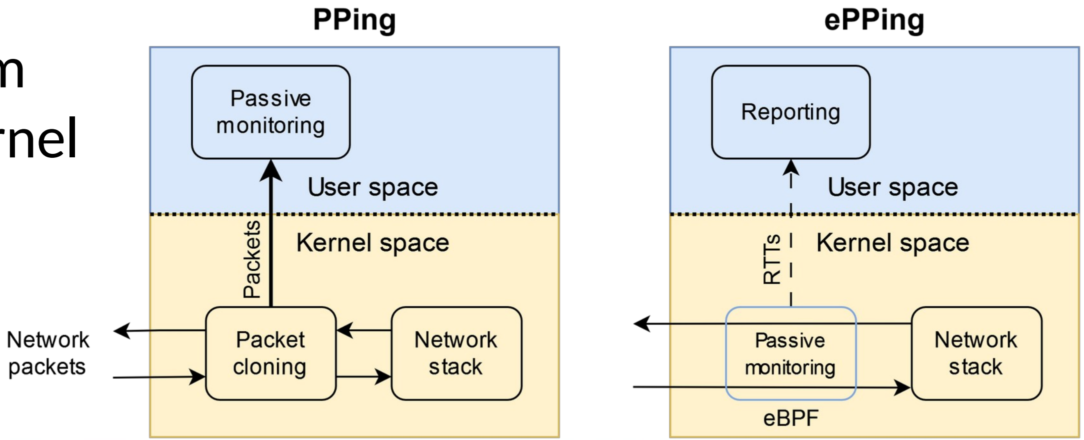
- Captures RTT between capture point and end host



Our solution – an evolved Passive Ping

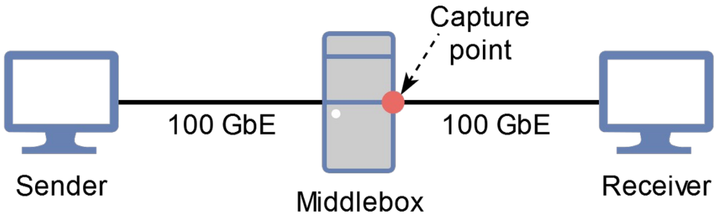
- Use eBPF to implement passive monitoring in kernel space
  - Direct access to packet buffer, no cloning needed
  - Only send computed RTTs to user space (not entire packets)

- eBPF allows attaching custom programs to hooks in the kernel
  - No need to recompile kernel

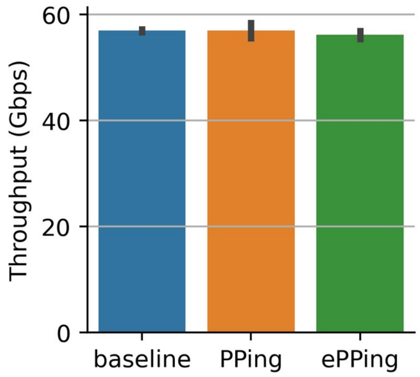


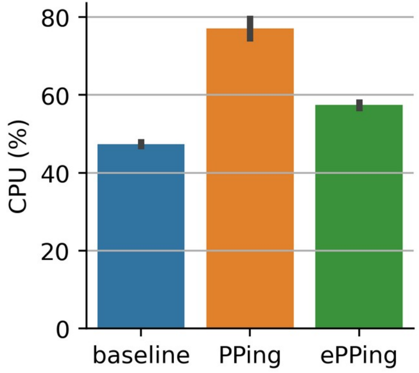
Performance results

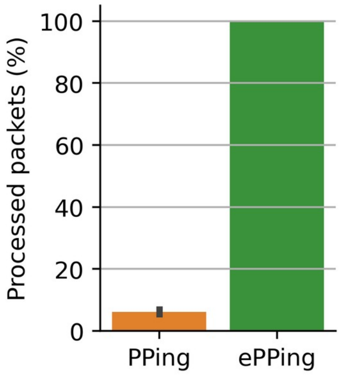
- Setup:



- When the end hosts are bottlenecks:

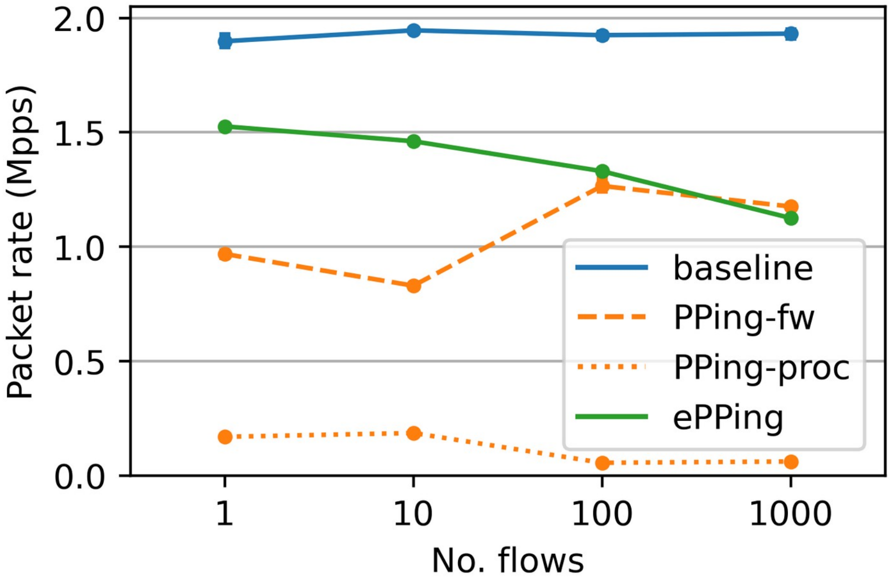






When running on bottleneck

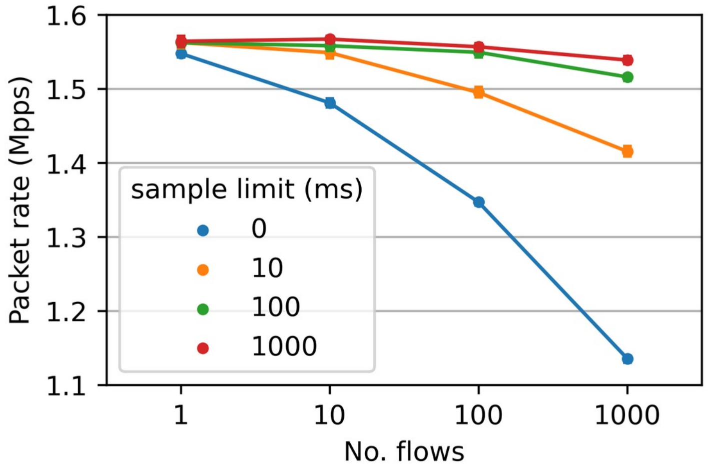
- Limit middlebox to single core
  - Core is 100% utilized
  - Overhead reduces forwarding rate
- PPing misses most packets
- More flows → more RTTs
  - ePPing starts to struggle due to reporting >100k RTTs/s



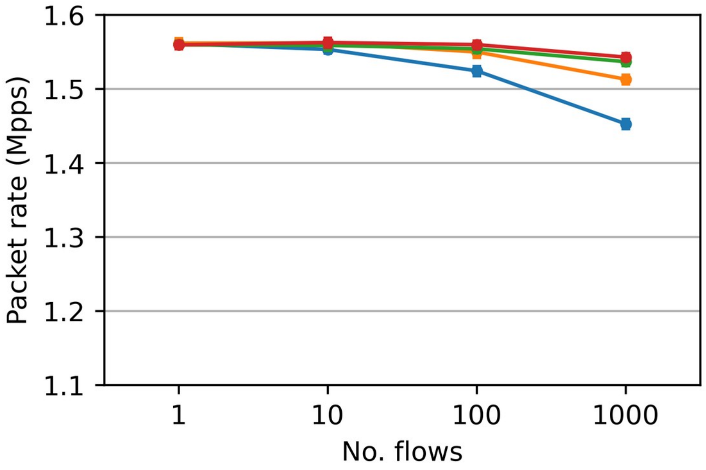
Further reducing overhead

- In-kernel sampling and aggregation greatly reduces overhead

Reporting individual RTTs



Aggregated RTT reports



Conclusion

- Summary:
  - Implemented continuous passive latency monitoring in kernel using eBPF
  - Can process packets at over 10x the rate of PPing
    - Over 1 Mpps / 10 Gbps on a single core
  - In-kernel sampling and aggregation can further reduce overhead

- Future work:
  - Improve aggregation of RTTs
  - Evaluate ePPing from an ISP vantage point
  - Add support for additional protocols (QUIC, DNS)

Try it yourself!

- ePPing is open source
  - https://github.com/xdp-project/bpf-examples/tree/master/pping

- Data, script and instructions to repeat experiments
  - https://doi.org/10.5281/zenodo.7555409

Thank you for your time!

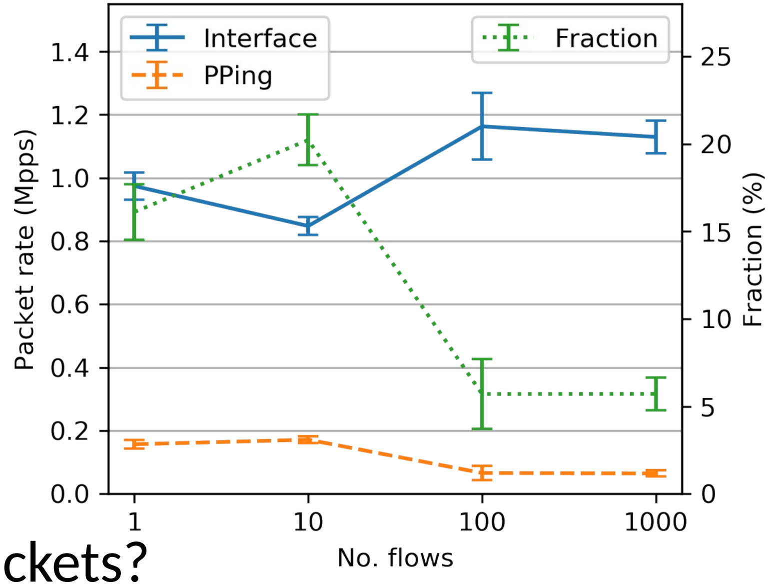
Questions?

simon.sundberg@kau.se

The problem with passive monitoring

- Packet capturing has high overhead
  - Can't keep up with high packet rates

- Consequences
  - Miss potentially valuable samples
  - Algorithms don't function properly



- 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 custom programs to various hooks at runtime
- Workflow
  - Compile to eBPF bytecode
  - Load into kernel
    - Verified
    - JITted
  - Attach to hook
- Use cases
  - Observability, Security, Networking

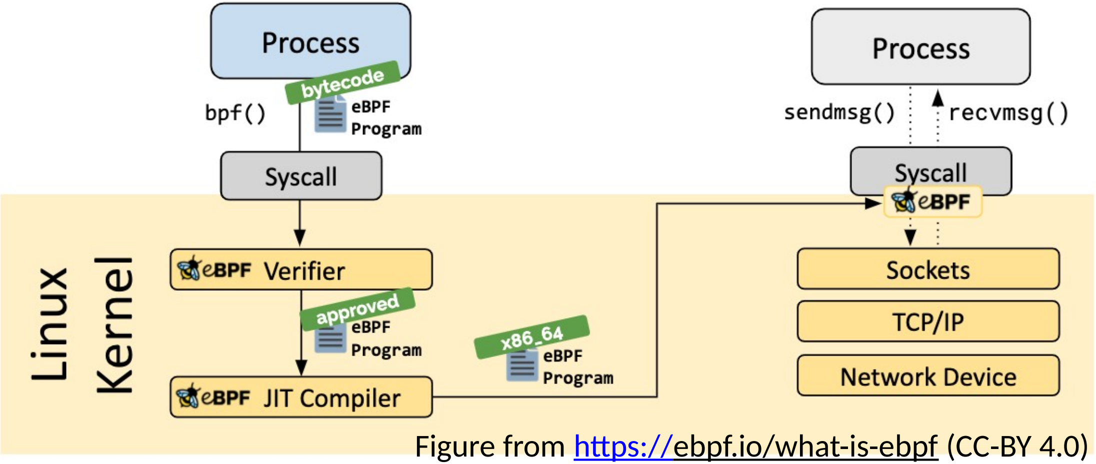
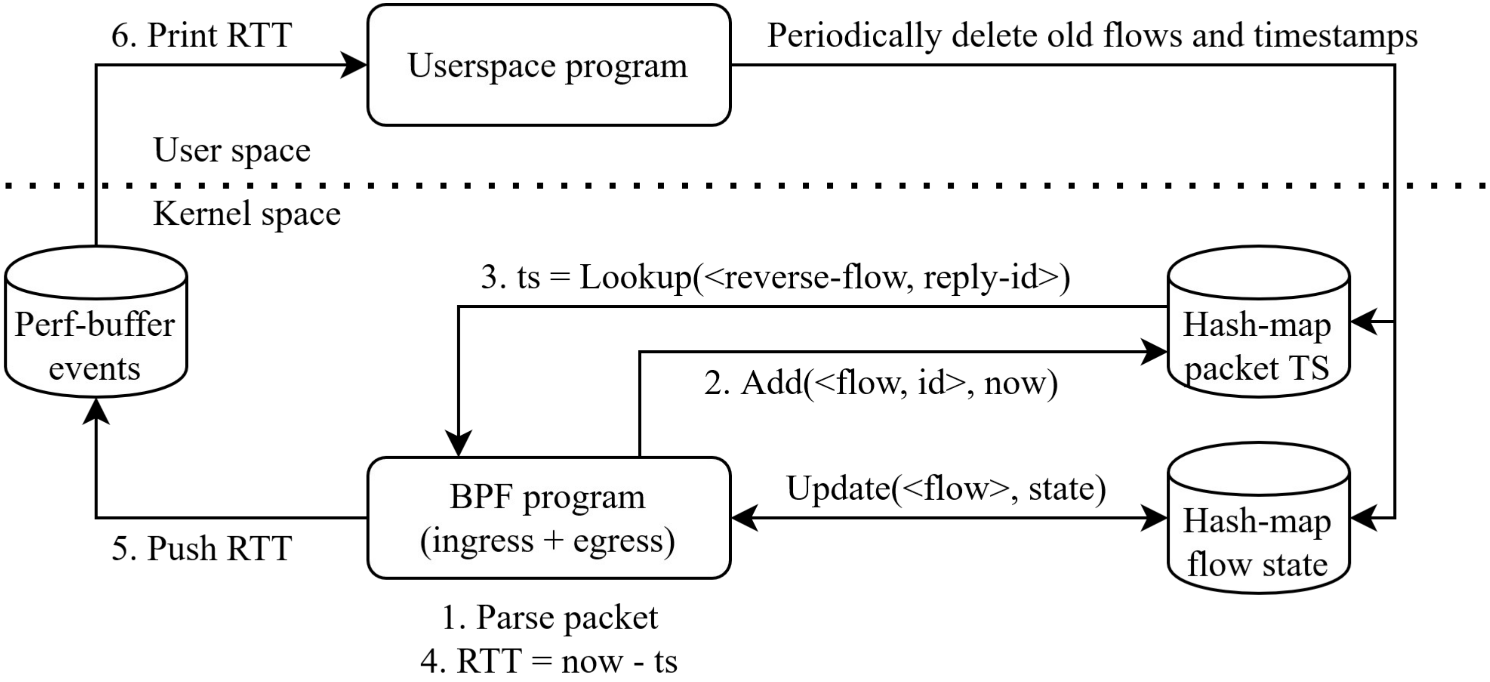


Figure from https://ebpf.io/what-is-ebpf (CC-BY 4.0)

ePPing design



Limitations

- Relies on TCP timestamps
  - Not available in all TCP traffic

- Delayed ACKs may inflate the RTTs
  - Impacts the TCP stack, but not necessarily applications above

- Evaluation mainly based on bulk flows
  - Plan to evaluate from ISP vantage point