# XDP and BPF research projects

Toke Høiland-Jørgensen
Principal Kernel Engineer

IBM Research Network Programming Workshop
October 2021

# Outline

- Intro: What is BPF and XDP?
- Two research projects:
  - Queueing for XDP
  - In-band latency measurement w/BPF

# What is BPF ?

From: https://ebpf.io/what-is-ebpf

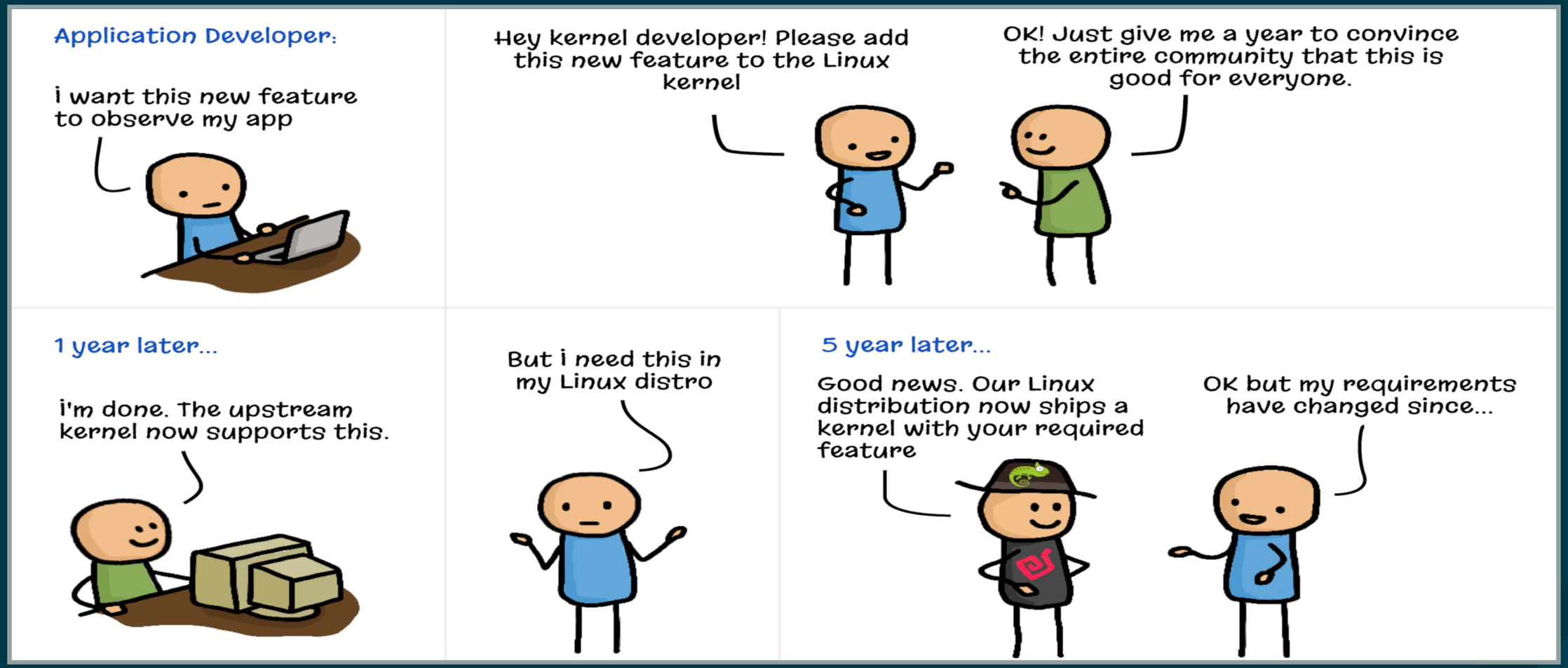
> *eBPF is a revolutionary technology that can run sandboxed programs in the Linux kernel without changing kernel source code or loading a kernel module*

BPF is a technology name: no longer an acronym

Rate of innovation at the operating system level: Traditionally slow

- BPF enables things at OS-level that were not possible before
- BPF will radically increase rate of innovation

# Traditional Kernel development process



# BPF development process

# BPF components

Closer look at the BPF components:

- Bytecode - Architecture independent Instruction Set
  - JIT to native machine instructions (after loading into kernel)
- Runtime environment - Linux kernel
  - Event based BPF-hooks all over the kernel
  - Per hook limited access to kernel functions via BPF-helpers
- Sandboxed by the BPF Verifier
  - Limits and verifies memory access and instructions limit

# BPF networking

Focus on BPF for networking

- XDP (eXpress Data Path) for fast processing at ingress
- TC-BPF hooks inside the regular stack
- BPF hooks for cgroups can also be useful for containers

# Why was an eXpress Data Path (XDP) needed?

Linux networking stack assumes layers L4-L7 are needed for every packet

- Root-cause of slowdown: (relative) high initial RX cost per packet

Needed to stay relevant as NIC speeds increase (time between packet small)

- New faster and earlier networking layer was needed to keep up.

XDP operate at layers L2-L3

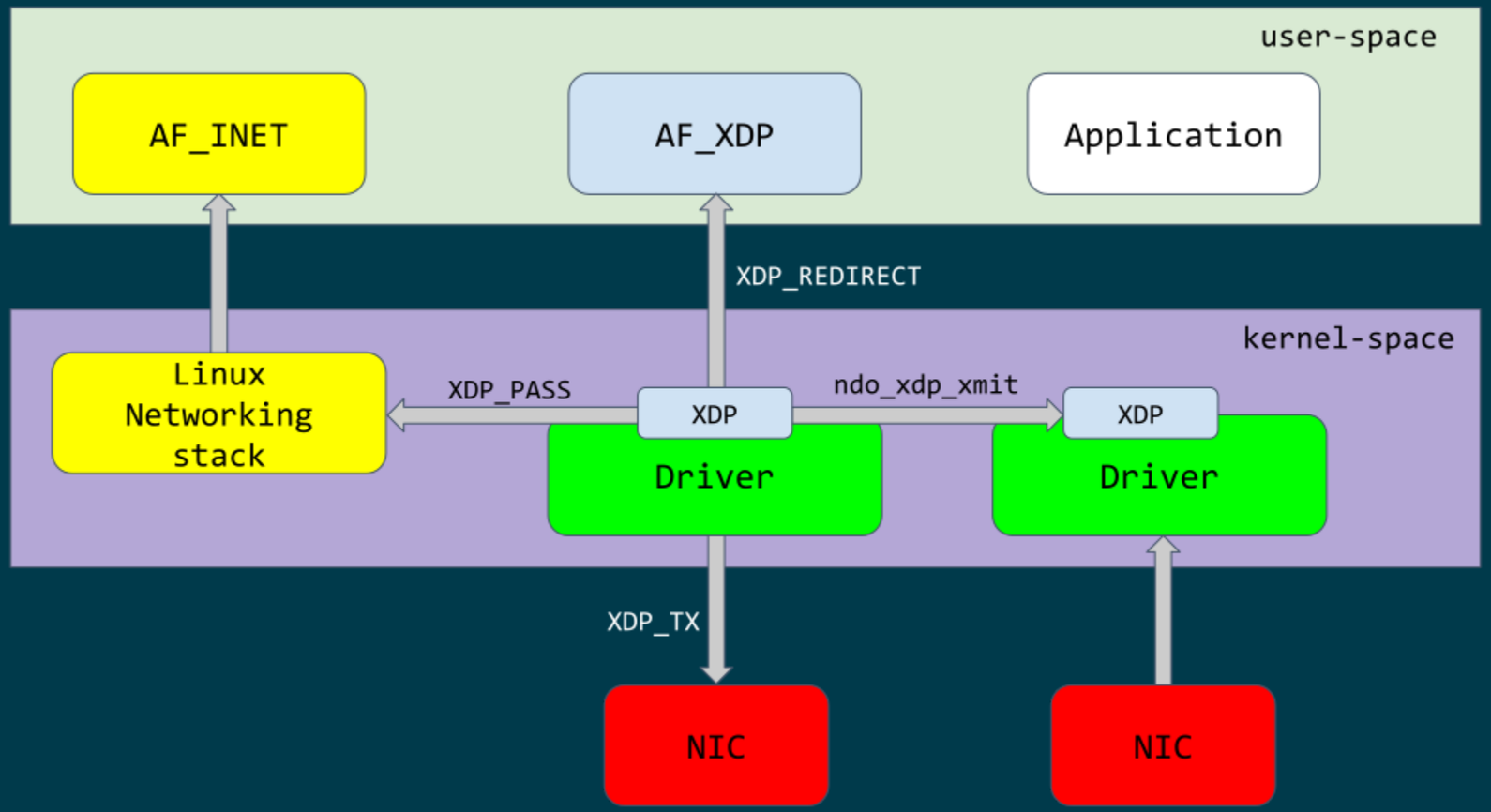
- L4 load-balancer possible when no IP-fragmentation occurs

# What is XDP?

XDP (eXpress Data Path) is a Linux in-kernel fast-path

- New programmable layer in-front of traditional network stack
  - Read, modify, drop, redirect or pass
  - For L2-L3 use-cases: seeing x10 performance improvements!
- Avoiding memory allocations
  - No SKB allocations and no-init (SKB zeroes 4 cache-lines per pkt)
- Adaptive bulk processing of frames
- Very early access to frame (in driver code after DMA sync)
- Ability to skip (large parts) of kernel code
  - Evolve XDP via BPF-helpers

# XDP architecture



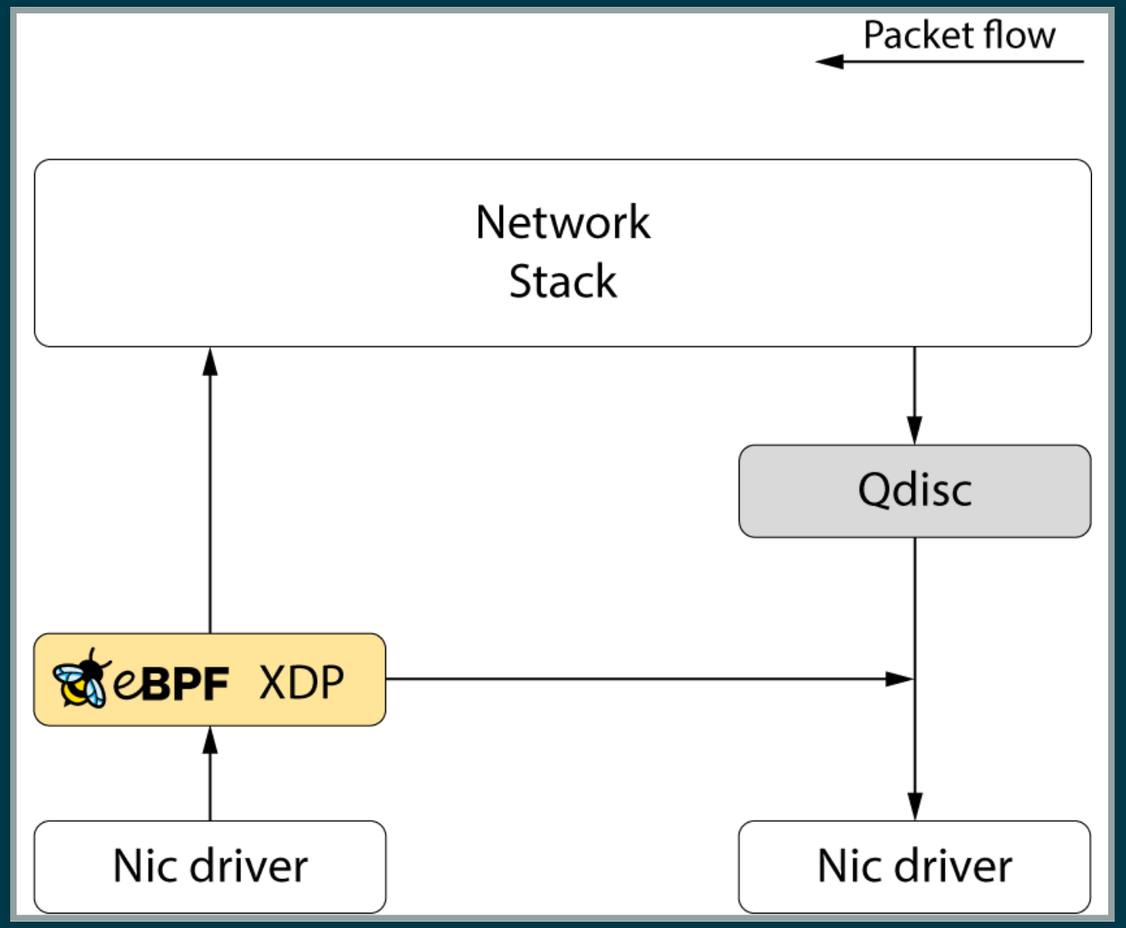
# XDP and BPF research projects

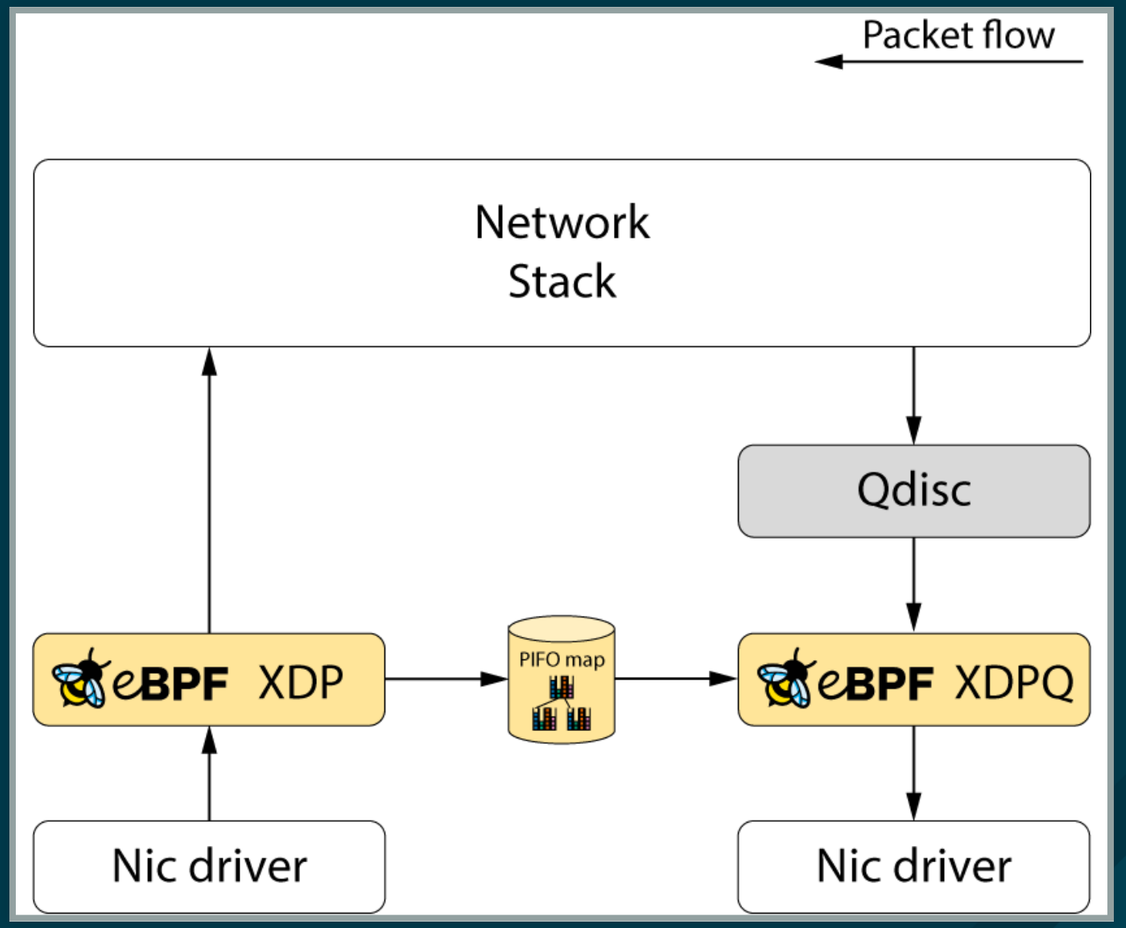
- Research collaboration between Red Hat and Karlstad University in Sweden
- Two PhD students working on BPF/XDP items
- Two engineers @ Red Hat as point of contact (Jesper and myself)

https://research.redhat.com/blog/research_project/building-the-next-generation-of-programmable-networking-powered-by-linux/

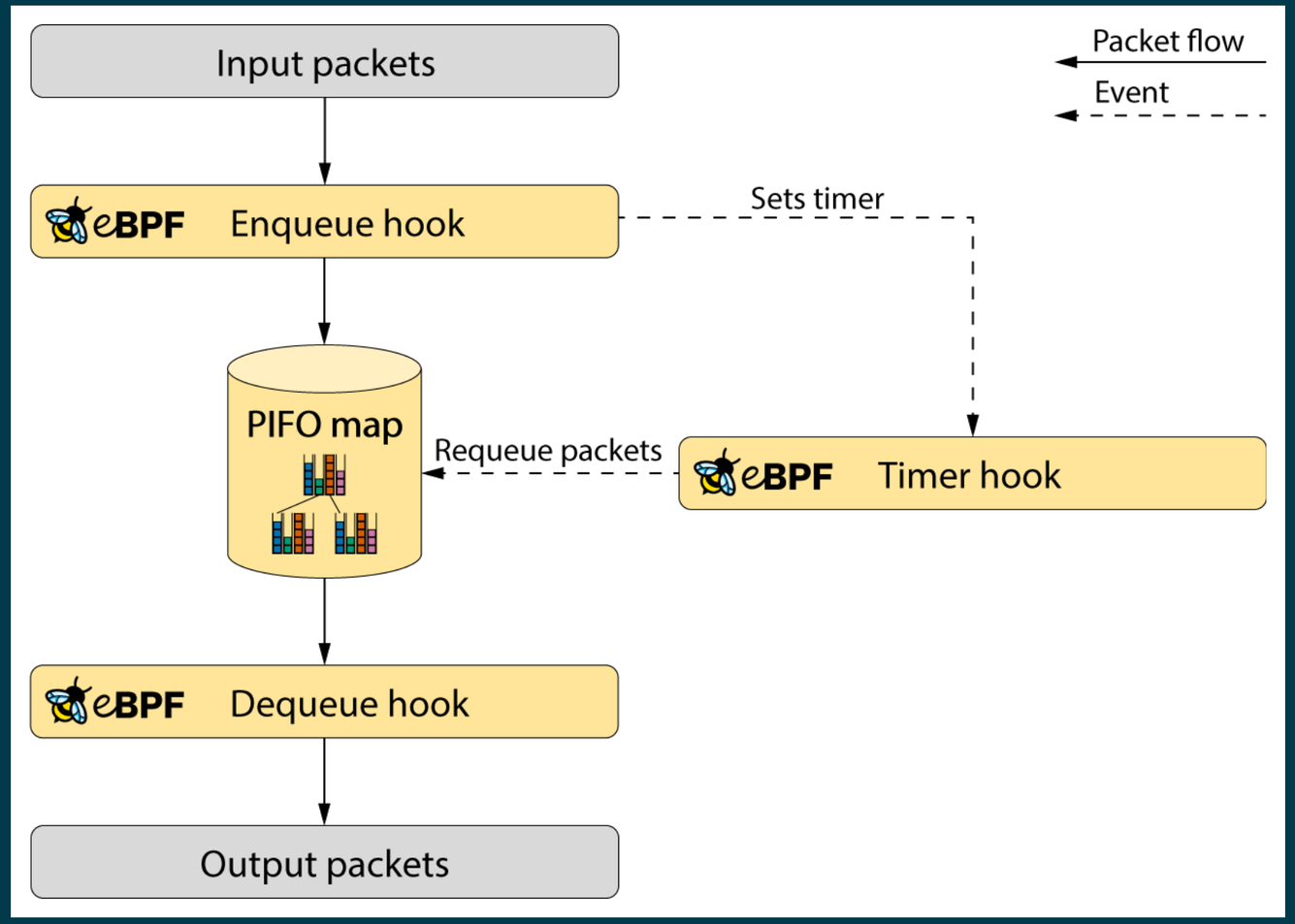
# Research project: Queueing in XDP

# XDP operation – before and after



# Design of XDP queueing mechanism



# Research project: In-band latency measurement

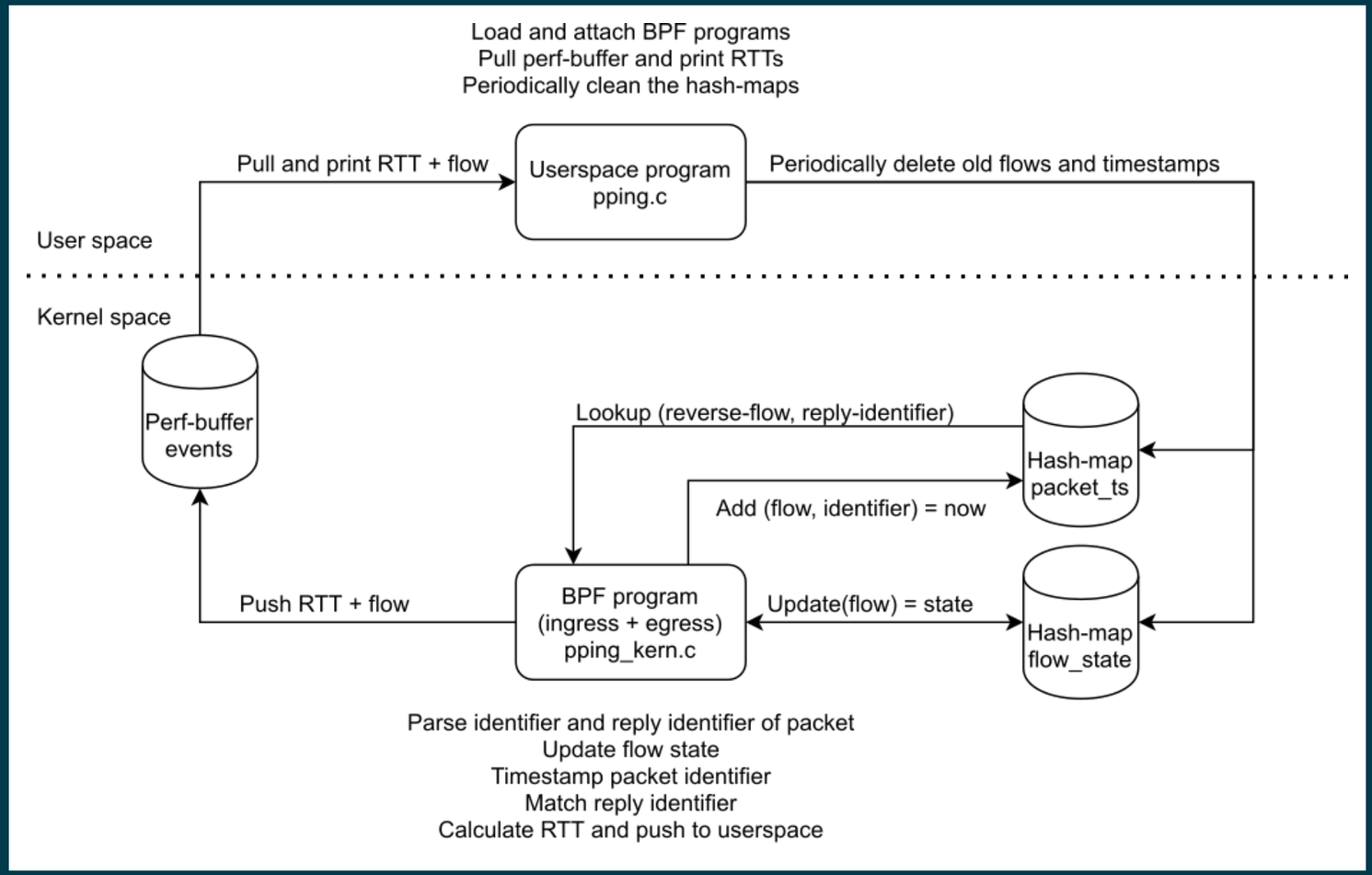
# The ‘passive ping’ utility

Kathy Nichols developed `pping`, the passive ping utility:

https://github.com/pollere/pping

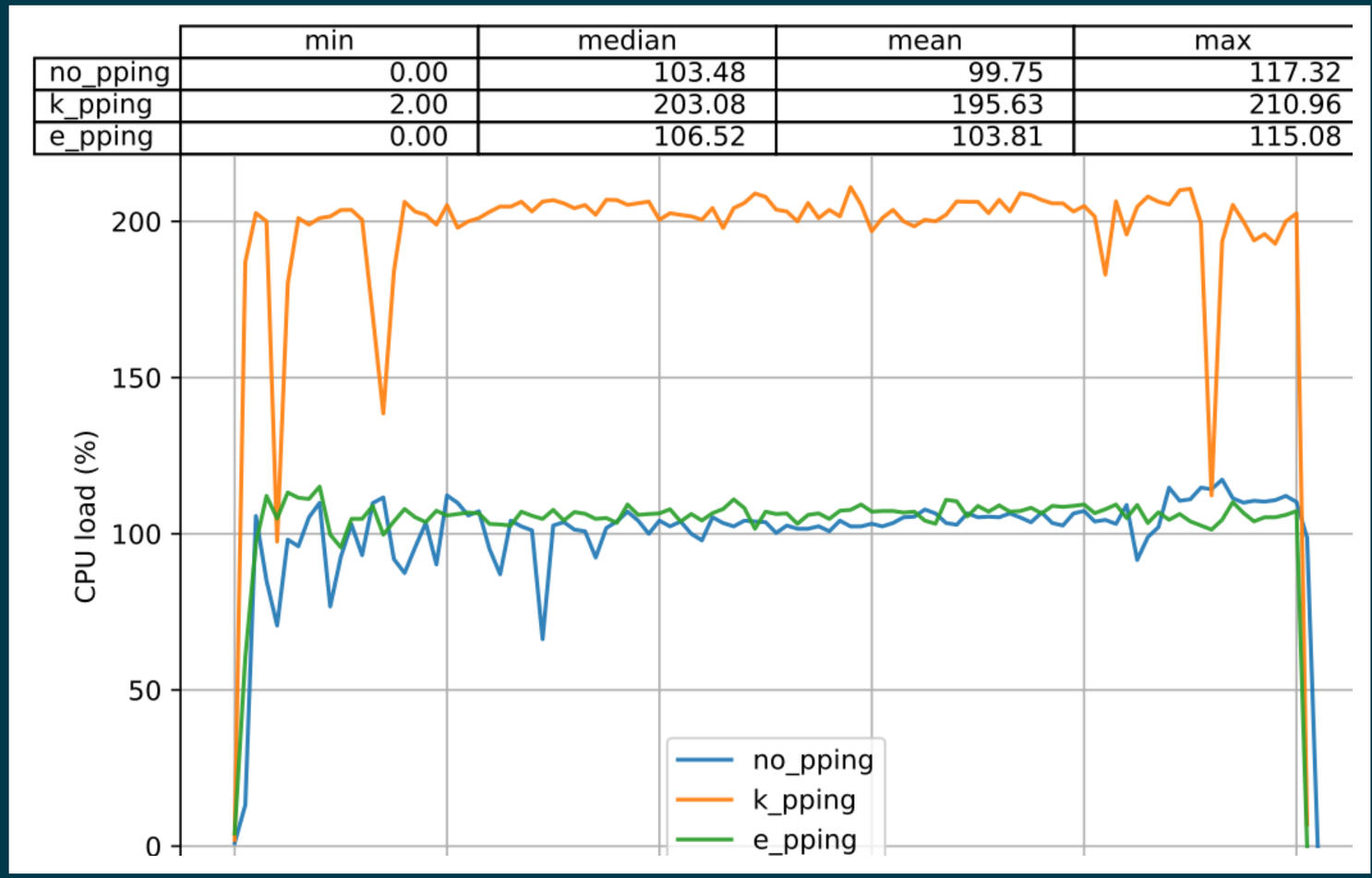
- Measures latency of TCP flows using timestamps.
- Implemented in C++ using libpcap -> quite a bit of overhead

# Using BPF for “always-on” pping



# pping preformance (preliminary!)

| | min | median | mean | max |
|---|---|---|---|---|
| no_pping | 0.00 | 103.48 | 99.75 | 117.32 |
| k_pping | 2.00 | 203.08 | 195.63 | 210.96 |
| e_pping | 0.00 | 106.52 | 103.81 | 115.08 |



# Questions, comments?

Projects are on Github:

- https://github.com/xdp-project/bpf-examples
- https://github.com/xdp-project/bpf-research