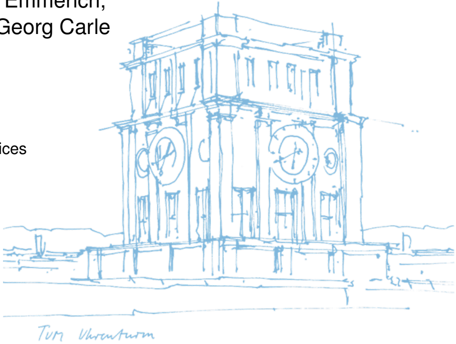
Performance Implications of Packet Filtering with Linux eBPF

Dominik Scholz, Daniel Raumer, Paul Emmerich,
Alexander Kurtz, Krzysztof Lesiak and Georg Carle

Chair of Network Architectures and Services
Department of Informatics
Technical University of Munich



History of Extended Berkeley Packet Filter (eBPF)

Recent Hot Topic

- 1992: BPF developed for UNIX
  - Packet filtering, e.g. *tcpdump*
- 2014: eBPF introduced into Linux Kernel
  - Network monitoring
  - Network traffic manipulation
  - Non-networking purposes
    - Tracing
    - Security auditing
    - ...
- Since then:
  - Continuous (performance) improvements [1]
  - “super powers have finally come to Linux” [2]
  - Offloading support, e.g. Netronome SmartNIC [3]
  - At Host Dataplane Acceleration Tutorial @ SIGCOMM 2018 [4]

[1] A thorough introduction to eBPF, `https://lwn.net/Articles/740157/`
[2] Brendan Gregg, BPF: Tracing and More, `https://www.youtube.com/watch?v=JRFNIKUROPE`
[3] NetroNews, August 2018, `http://hosted.verticalresponse.com/183413/a79f667b58/1413119999/c60e793082/`
[4] ACM SIGCOMM 2018 Morning Tutorial on Host Dataplane Acceleration (HDA), `https://conferences.sigcomm.org/sigcomm/2018/tutorial-hda.html`

Outline

Extended Berkeley Packet Filter (eBPF)

What is it?

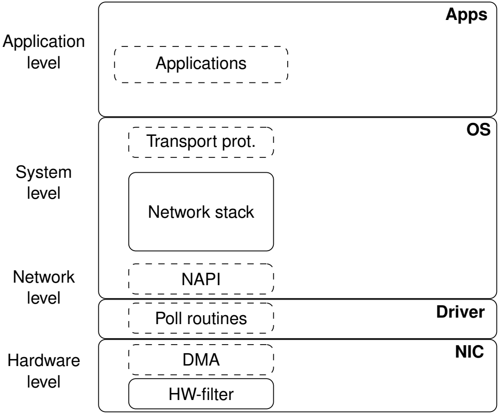
- User space program
- Run in virtual machine in kernel space (“sandboxed”)
- Dynamically interpreted (default) or compiled just-in-time (JIT)

Limitations

- Static verification
  - No backward jumps (loops)
  - Maximum of 4096 instructions
  - ➔ Cannot compromise/block kernel
- Data access: key-value stores (maps)
  - Memory region set up before program is loaded
  - Key size, value type, max. number of entries predetermined
  - ➔ Secure data access between user space and kernel space

Study: Packet Filtering

Layers of Packet Filters

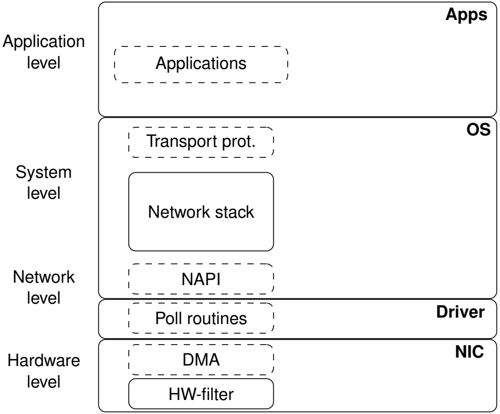


- Hardware offloading and filtering
- Dedicated platforms based on FPGAs or SmartNICs

→ High-performance, ideal for coarse filtering (DoS)

Study: Packet Filtering

Layers of Packet Filters



- Before network stack processing

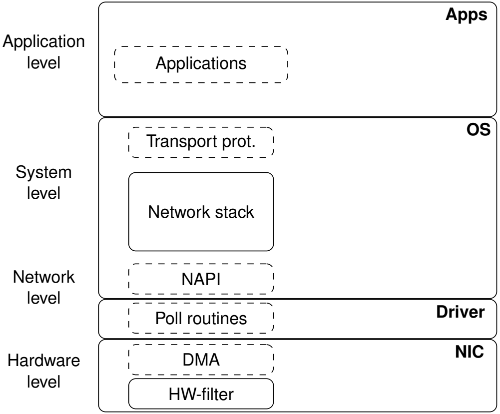
→ Dropping packets with low overhead in software

- Hardware offloading and filtering
- Dedicated platforms based on FPGAs or SmartNICs

→ High-performance, ideal for coarse filtering (DoS)

Study: Packet Filtering

Layers of Packet Filters



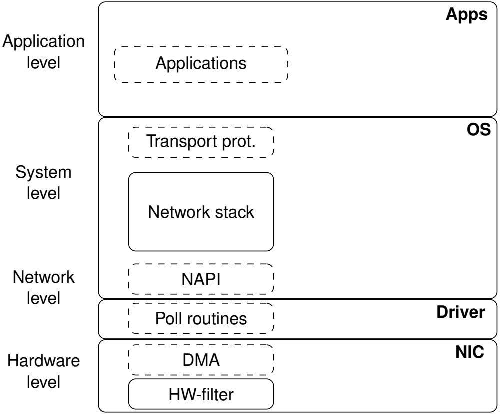
- Hooks into packet processing of network stack
- e.g. iptables or nftables
- → Requires root access, system-specific knowledge

- Before network stack processing
- → Dropping packets with low overhead in software

- Hardware offloading and filtering
- Dedicated platforms based on FPGAs or SmartNICs
- → High-performance, ideal for coarse filtering (DoS)

Study: Packet Filtering

Layers of Packet Filters



- Traffic addressed for a specific application
- → Application “knows best”, high penalty for dropping packets

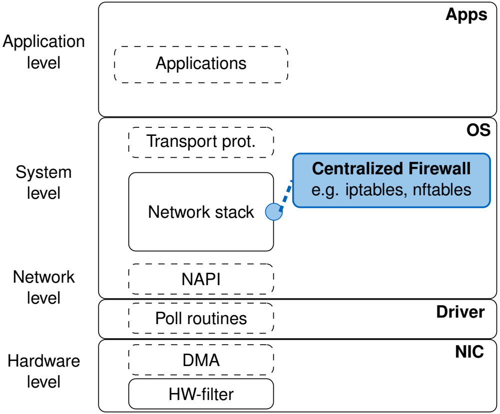
- Hooks into packet processing of network stack
- e.g. iptables or nftables
- → Requires root access, system-specific knowledge

- Before network stack processing
- → Dropping packets with low overhead in software

- Hardware offloading and filtering
- Dedicated platforms based on FPGAs or SmartNICs
- → High-performance, ideal for coarse filtering (DoS)

Use Case: Packet Filtering

Common Scenario – State of the Art



- Traffic addressed for a specific application
- → Application "knows best", high penalty for dropping packets

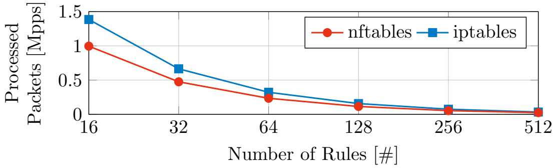
- Hooks into packet processing of network stack
- e.g. iptables or nftables
- → Requires root access, system-specific knowledge

- Before network stack processing
- → Dropping packets with low overhead in software

- Hardware offloading and filtering
- Dedicated platforms based on FPGAs or SmartNICs
- → High-performance, ideal for coarse filtering (DoS)

Use Case: Packet Filtering

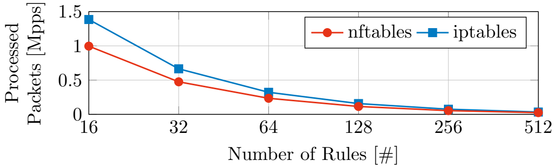
Performance Baseline



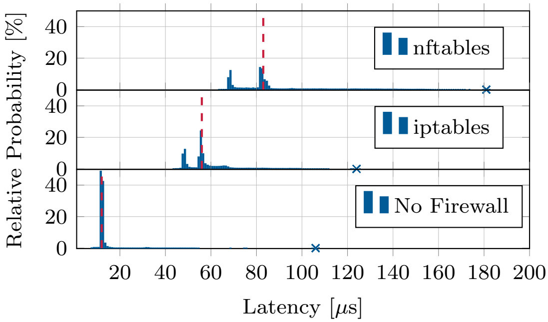
Maximum packet rate

Use Case: Packet Filtering

Performance Baseline



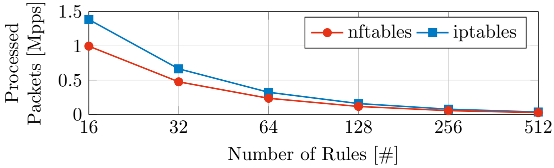
Maximum packet rate



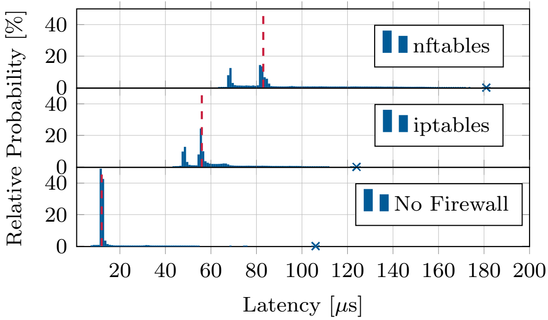
Latency distribution at 0.03 Mpps

Use Case: Packet Filtering

Performance Baseline



Maximum packet rate



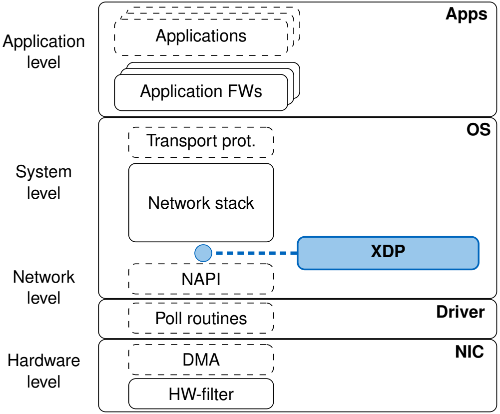
Latency distribution at 0.03 Mpps

Performance sufficient for today's applications?

Limitations: Centralized, complex ruleset, requiring root access

Use Case: Packet Filtering (using Commodity Hardware)

Possibilities with eBPF



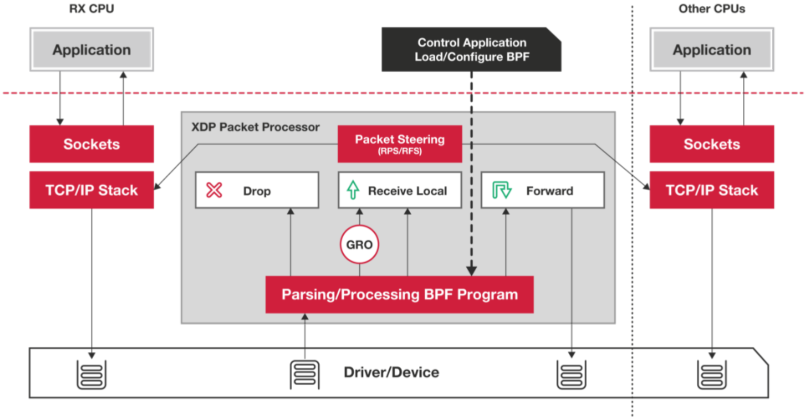
eXpress Data Path

- First line of defense
- Coarse but efficient filtering

→ Protection against DoS attacks

Case Study I: eXpress Data Path (XDP)

Overview



Source: https://www.iovisor.org/technology/xdp

Case Study I: eXpress Data Path (XDP)

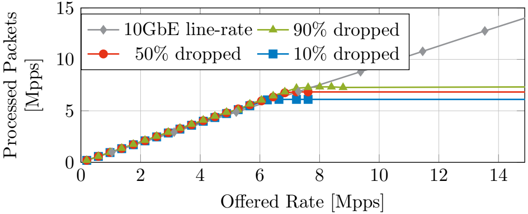
Measurement Setup

- XDP program:
  - Drop if port is blacklisted
  - Otherwise, forward to outgoing interface
  - ➔ Excludes network stack
- Load generator:
  - MoonGen [7]
  - Generates *n* UDP flows
- Just-in-time compiler enabled
- Traffic pinned to single core
- Hyper-threading and Turbo Boost disabled

[7] `https://github.com/emmericp/MoonGen`

Case Study I: eXpress Data Path (XDP)

Performance Baseline

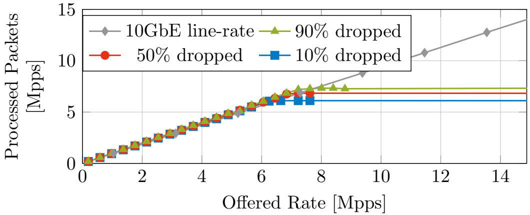


Packet filtering performance

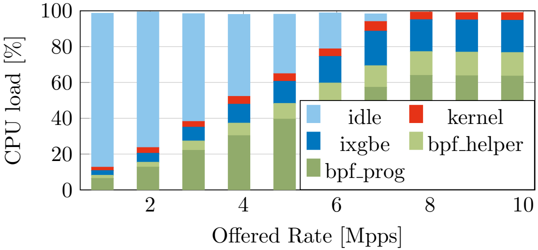
- Drop everything: 10 Mpps
- Drop x-Percent: 6.4 Mpps to 7.2 Mpps

Case Study I: eXpress Data Path (XDP)

Performance Baseline



Packet filtering performance



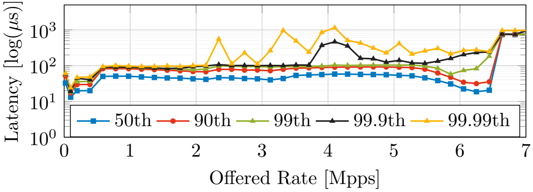
Whitebox measurement – Profiling

- Drop everything: 10 Mpps
- Drop x-Percent: 6.4 Mpps to 7.2 Mpps
- Kernel < 5% CPU time

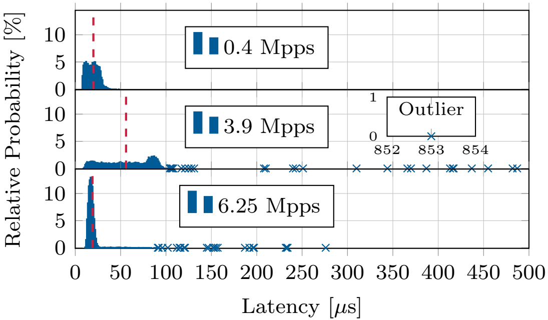
→ 5 to 10 times performance increase compared to in-kernel filtering

Case Study I: eXpress Data Path (XDP)

Latency



Latency percentiles (90 % passing traffic)



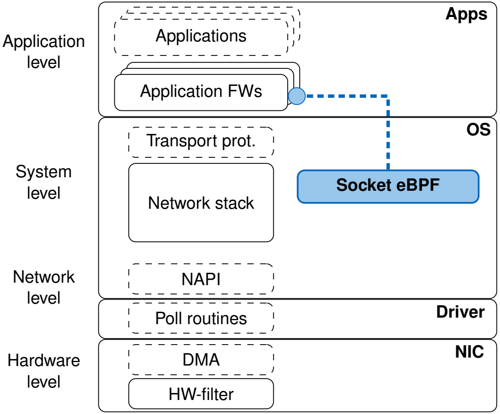
Selected histrograms

Comparison

- iptables: 55μs (median), 110μs (99.99th %ile)
- nftables: 82μs (median), 154μs (99.99th %ile)

Use Case: Packet Filtering (using Commodity Hardware)

Possibilities with eBPF



Application Firewall

- Socket attached filtering
- Application (developer) can extract desired traffic

→ Fine-grained and flexible filtering

Case Study II: Socket Attached Filtering

Application Firewall

Steps

- Write filter in C
- Add eBPF virtual machine to socket
- Run filter in virtual machine

➔ Drop or forward to application

Case Study II: Socket Attached Filtering

Application Firewall

Steps

- Write filter in C
- Add eBPF virtual machine to socket
- Run filter in virtual machine
- ➔ Drop or forward to application

Technical Implementation

- BPF Compiler Collection
- systemd socket activation
- ➔ Added command-line interface and wrapper functions [8]
- ➔ E.g. port-knocking simple to implement

[8] `https://github.com/AlexanderKurtz/alfwrapper`

Case Study II: Socket Attached Filtering

Application Firewall

Steps

- Write filter in C
- Add eBPF virtual machine to socket
- Run filter in virtual machine
- ➔ Drop or forward to application

Technical Implementation

- BPF Compiler Collection
- systemd socket activation
- ➔ Added command-line interface and wrapper functions [8]
- ➔ E.g. port-knocking simple to implement

What have we gained?

- Application developer knows best
- Rule-set shipped with application
- Independent of system administrator
- Small rule-set per application/socket
- Not interfering with rule-set of another socket
- ➔ Optimized for application
- ➔ Reduced complexity, less error-prone

[8] `https://github.com/AlexanderKurtz/alfwrapper`

Case Study II: Socket Attached Filtering

Measurement Setup

Differences

- Application interested in throughput
- (Stateful) applications more difficult to benchmark

Case Study II: Socket Attached Filtering

Measurement Setup

Differences

- Application interested in throughput
- (Stateful) applications more difficult to benchmark

Solution

- iperf as load generator
- Benchmark using loopback device

Case Study II: Socket Attached Filtering

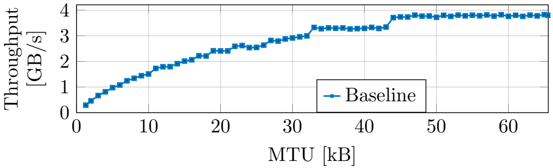
Measurement Setup

Differences

- Application interested in throughput
- (Stateful) applications more difficult to benchmark

Solution

- iperf as load generator
- Benchmark using loopback device



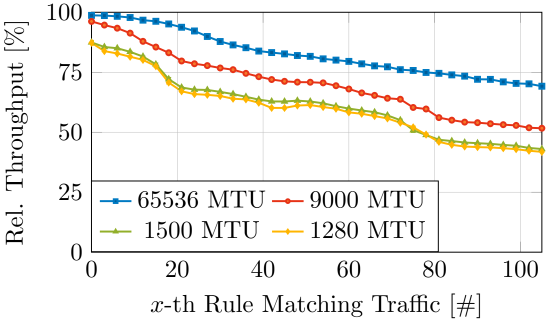
Baseline transmission speeds

Case Study II: Socket Attached Filtering

Performance

Interface filtering

- Simple operation
- Whitelisting: Matching rule at *x*-th position

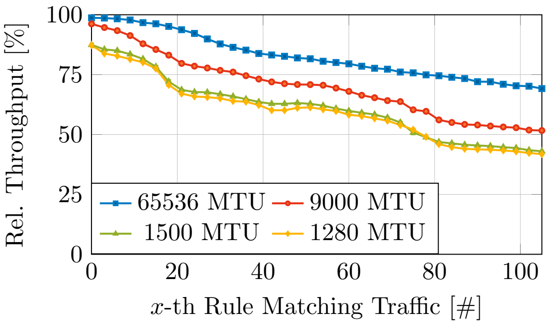


Case Study II: Socket Attached Filtering

Performance

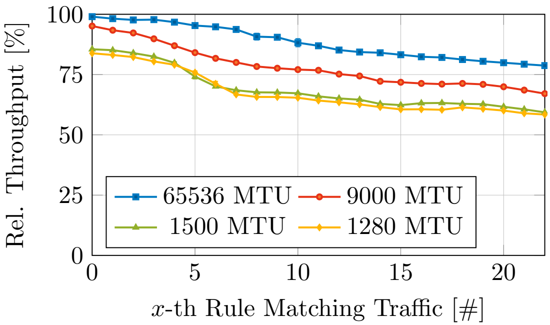
Interface filtering

- Simple operation
- Whitelisting: Matching rule at x-th position



Subnet filtering

- Complex operation
- Limits number of rules

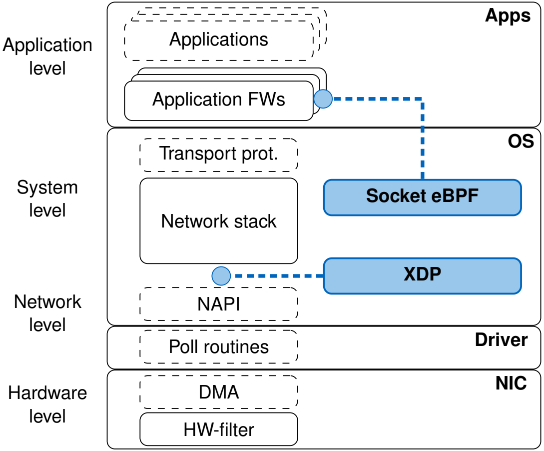


Conclusion

eBPF allows to break up traditional packet filtering

- Adds flexibility
- High-performance
- But: limitations

→ Many applications can benefit from eBPF



Conclusion

eBPF allows to break up traditional packet filtering

- Adds flexibility
- High-performance
- But: limitations

→ Many applications can benefit from eBPF

Future developments

- Improvements for eBPF just-in-time compiler
- Hardware accelerators and offloading capabilities
- P4 programming language