FOSDEM'19 • Brussels, 2019-02-02

Unifying network filtering rules for the Linux kernel with eBPF

Quentin Monnet

<quentin.monnet@netronome.com>
@qeole

NETRONOME

Outline

Several network filtering mechanisms in the Linux kernel

- What are they, and what do they do?
- How are they used?

Latest addition: eBPF

- What does it bring to filter networking?

Increasing number of convergence leads between the different models

- What are the objectives?
- How can they be unified?

Some network filtering mechanisms in the Linux kernel

Netfilter (iptables/nf_tables)

Framework for packet filtering (firewall), NAT

- Often the default choice for dropping flows
- Several front-end components (ebtables, arptables, iptables, ip6tables, nf_tables, conntrack)
- Back-end: Netfilter
- nf_tables successor to iptables: more flexible, more efficient

TC framework for Traffic Control in the kernel: traffic shaping, scheduling, policing, dropping

- “Queueing disciplines” (qdisc), possibly applied to “classes”
- Filters are used to dispatch packets into the different classes (Traffic control mostly applies to egress traffic, but filters also usable for ingress)
- Framework actually using a variety of filters:
  - basic (ematch, “extended match”)
  - flow
  - flower
  - u32
  - [bpf]
  - Specific filters: fw, route, rsvp, tcindex

Hardware filters (ethtool)

"Receive network flow classification": Hardware filters

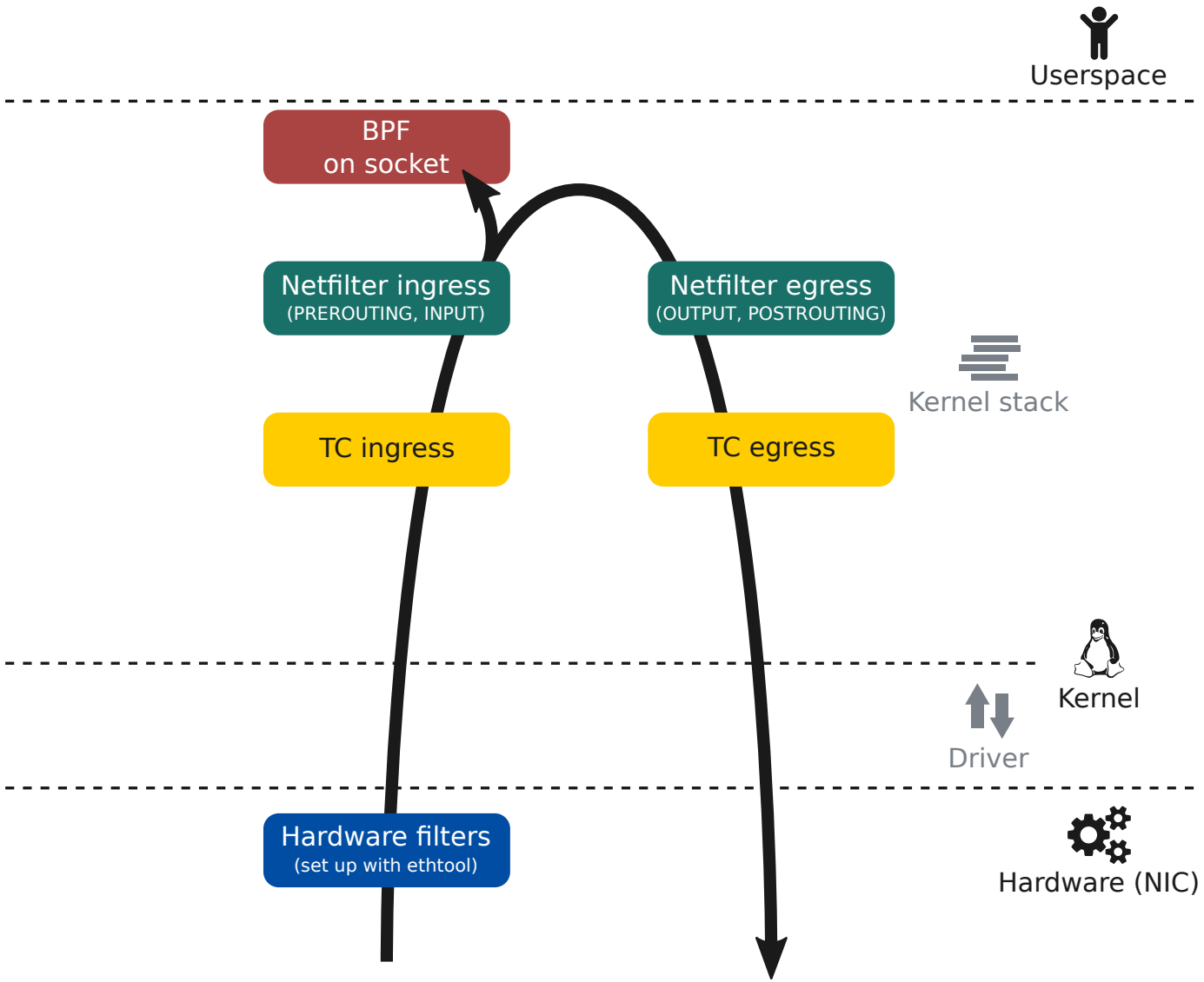
- Main objective: flow steering, but able to drop flows
- Needs hardware support, not all NICs have it
- Rules set with `ethtool -U` (ioctl)

pcap-filters, cBPF (e.g. for tcpdump)

Facility from the libpcap library

- Takes an expression and turns it into a filter
- Output is legacy BPF (cBPF), attached to sockets in the kernel (or run in user space if not on Linux)
- Used by tcpdump (see `tcpdump -i eth0 -d <expr>`)

Filtering hooks



Many rule syntaxes

Example rule: Drop incoming IP(v4) HTTP packets

```
# iptables -A INPUT -i eth0 \
        -p tcp --dport 80 -j drop

# nft add rule ip filter input iif eth0 \
        tcp dport 80 drop

# tcpdump -i eth0 \
        ip and tcp dst port 80

# tc filter add dev eth0 ingress flower \
        ip_proto tcp dst_port 80 action drop

# ethtool -U eth0 \
        flow-type tcp4 dst-port 80 action -1
```

Many other ways to filter packets

NETRONOME

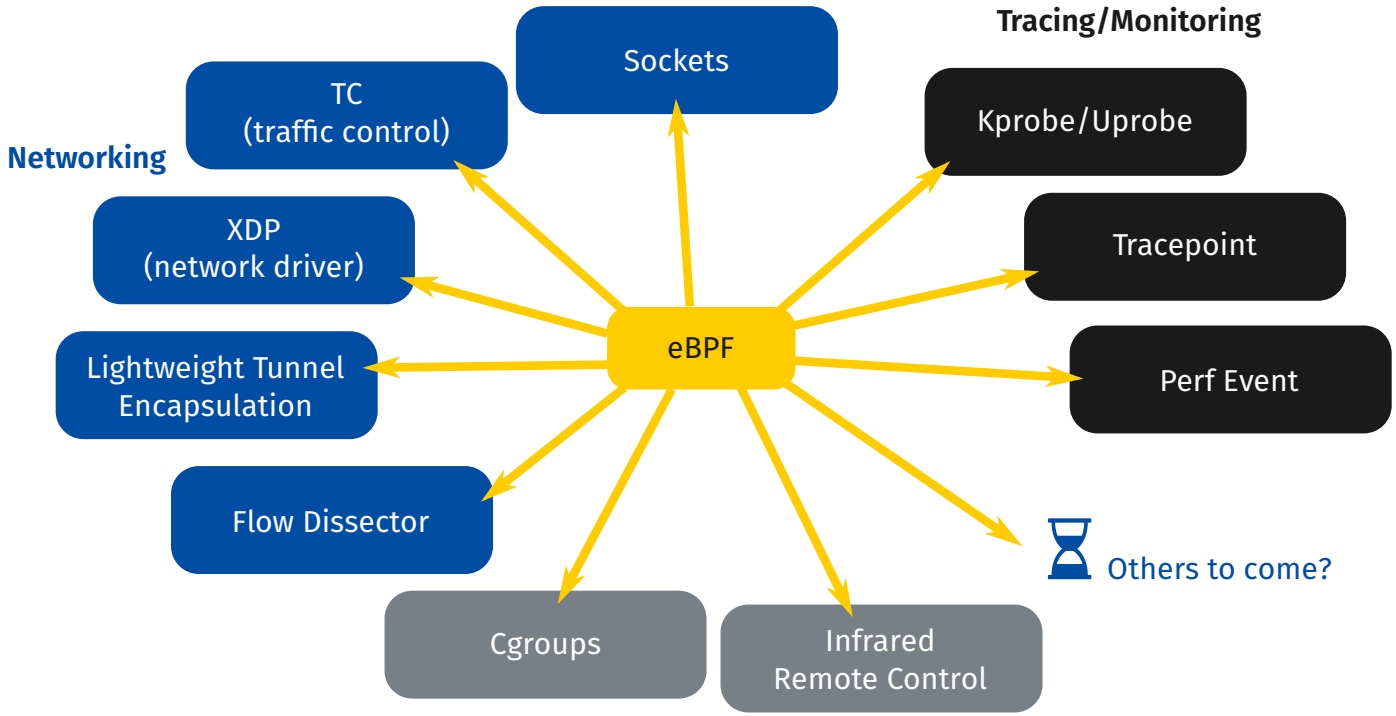
The list is not exhaustive
Other frameworks are available (many of them out of kernel space)

- Software switches: Open vSwitch, etc.
- User space processing: DPDK (rte-flows), firewall apps, etc.
- P4 as another way to implement switches/filters, compile to target
- ...

Enter eBPF

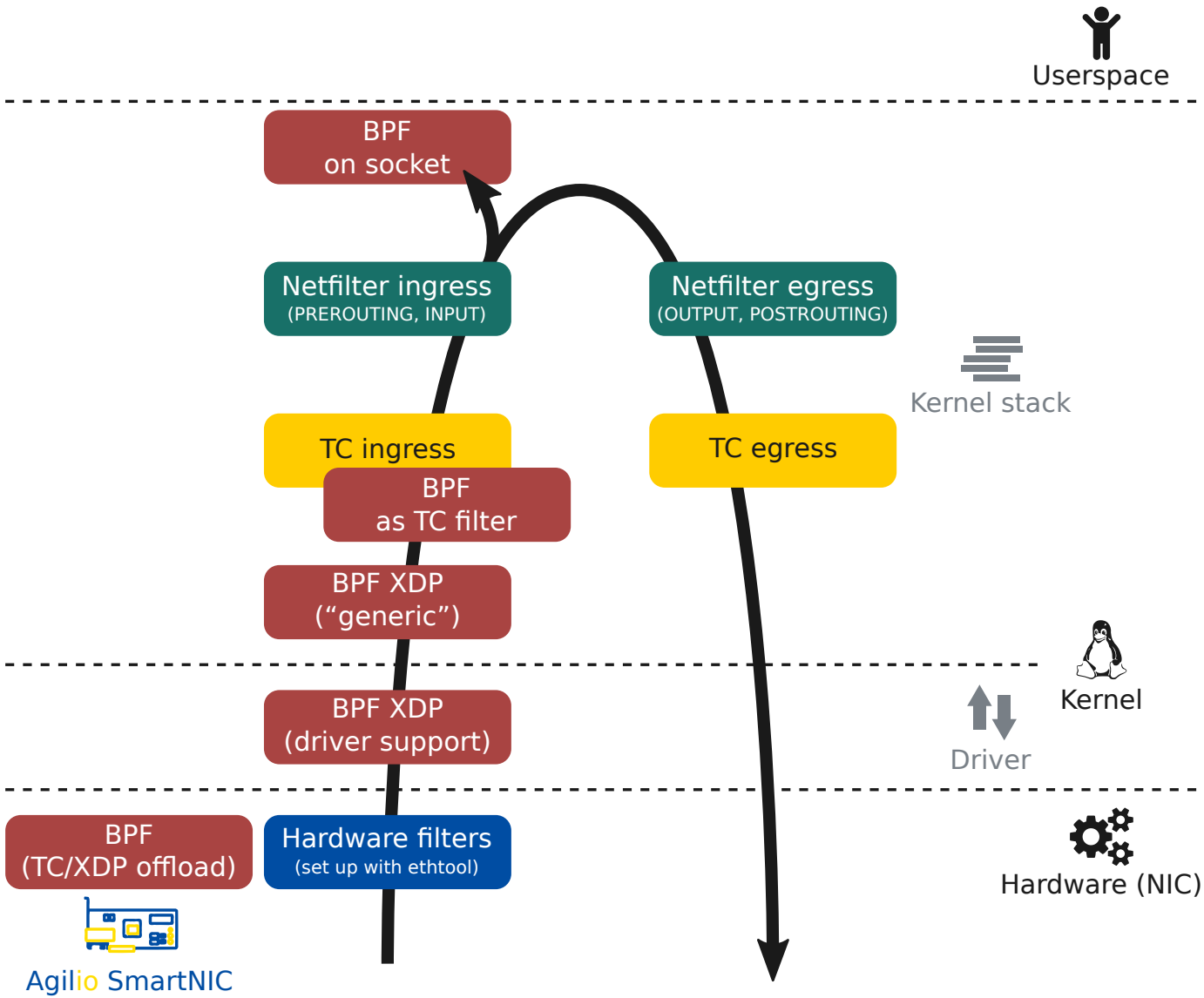
Generic, efficient, secure in-kernel (Linux) virtual machine

Event-based programs injected, verified and attached in the kernel



Specific features: Maps, tail calls, helper functions

In the rest of the presentation: "BPF" means "eBPF"


Userspace
BPF
on socket
Netfilter ingress
(PREROUTING, INPUT)
Netfilter egress
(OUTPUT, POSTROUTING)
Kernel stack
TC ingress
TC egress
BPF
as TC filter
BPF XDP
(“generic”)
Kernel
BPF XDP
(driver support)
Driver
BPF
(TC/XDP offload)
Hardware filters
(set up with ethtool)
Hardware (NIC)
Agilio SmartNIC

BPF is POWER!

- Programmability (change network processing at runtime)
- In-kernel verifier: safety, security of the programs
- JIT (Just-in-time) compiler available for main architectures: speed!
- Low-level (driver hooks): speed!!
- Hardware offload: speed!!!

Also:

- Headaches, long nights spent rewriting the filters
- Additional pain to pass the verifier

But keep in mind: BPF is self-contained, well defined, flexible
Maybe a good intermediate representation to represent filters?

Convergence of the models

Why unifying?

User side:

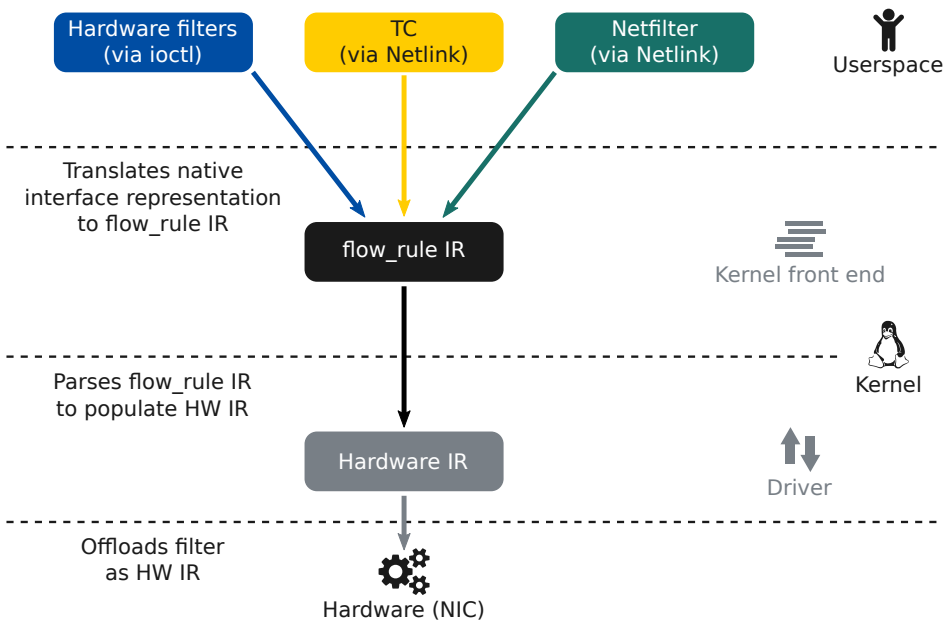
- Transparently reuse existing set of rules
- Benefit from the best of each world: flexibility, ease of use, performance

Developer side:

- Easier to work on a common intermediate representation rather than on a variety of distinct back-ends
- Better uncoupling of the front- and back-ends

Work in progress from Pablo Neira Ayuso—No BPF in this one

- Intermediate representation for ACL hardware offloads
- Based on Linux flow dissector infrastructure and TC actions
- Can be used by different front-ends such as HW filters, TC, Netfilter

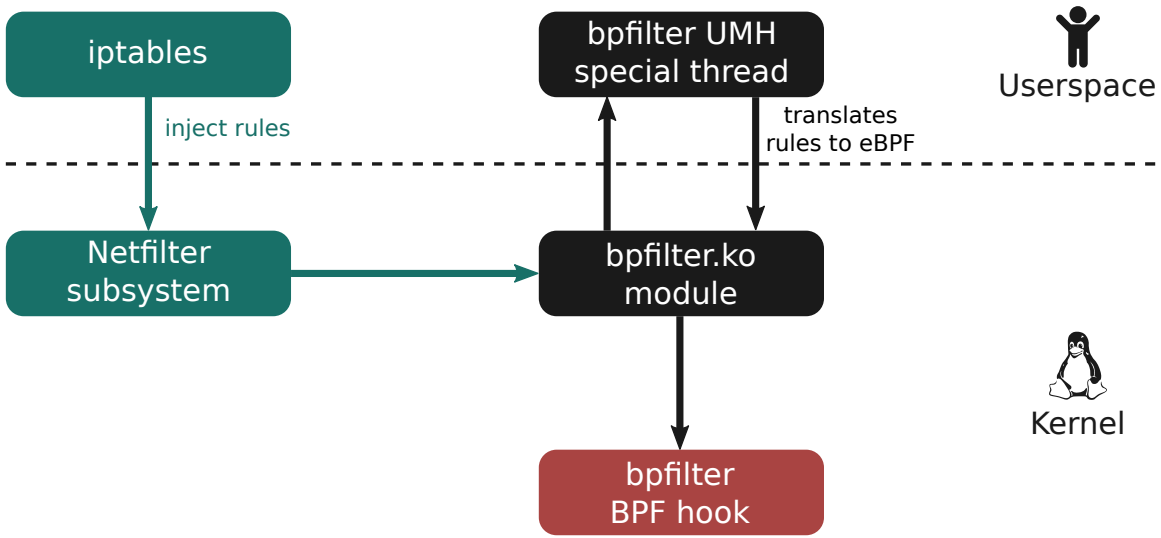


Motivation:

- Unified IR passed to the driver: avoid having one parser for each ACL front-end
- Stop exposing TC front-end details to drivers (easier to add features to TC)

bpfilter: new back-end for iptables in Linux, based on BPF

- The `iptables` binary is left untouched
- Rules are translated into a BPF program
- Uses a special kernel module launching an ELF executable in a special thread in user space, for rule translation
- Also: proposal for nf_tables to BPF translation on top of bpfilter

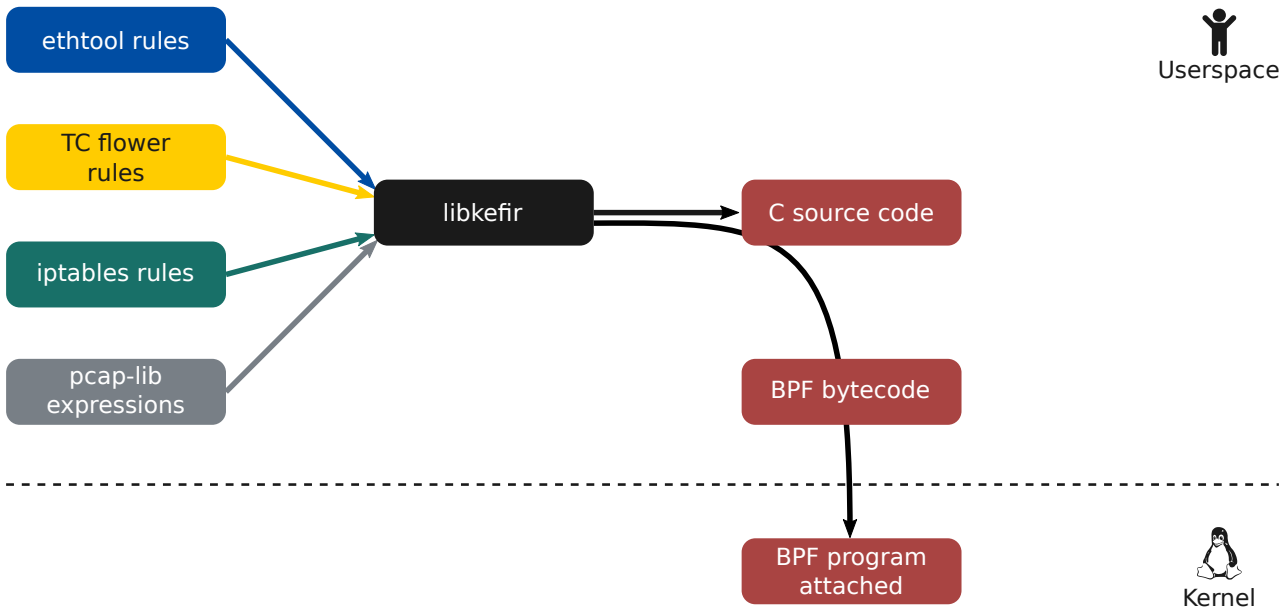


Motivation:

- Reuse rules from iptables
- Improve performance (JIT, offloads)

libkefir: **KE**rnel **FI**ltering **R**ules: Work in progress @ Netronome

- Turn simple ACL rules into hackable BPF programs
- Motivation similar to bpfilter: reuse rules, with improved performance
- But do not try to handle all cases
- And give BPF-compatible C source code to users, so they can hack it
- Comes as a library, for inclusion in other projects



Sorry, not published yet!

Various frameworks for packet filtering in Linux

BPF is one of them, brings new perspectives in terms of programmability, performance, speed, speed and speed

Convergence between different models is beginning to emerge:

- Easier handling of rules for driver developers (flow_rule IR proposal)
- Reuse of existing rules for users (bpfilter, libkefir)
- Better performance for those existing set of rules

Also, consider:

- P4 as another approach for convergence—BPF is one target
- BPF used in other places: Open vSwitch datapath, DPDK
- *eBPF as a heterogeneous processing ABI* (LPC 2018)
- Usage of a DSL for producing BPF programs, but targeted at tracing the Linux kernel: bpftrace

Questions

Additional resources:

Dive into BPF: a list of reading material
https://qmonnet.github.io/whirl-offload/2016/09/01/dive-into-bpf/

Why is the kernel community replacing iptables with BPF?
https://cilium.io/blog/2018/04/17/why-is-the-kernel-community-replacing-iptables/

[PATCH net-next,v6 00/12] add flow_rule infrastructure
https://lwn.net/ml/netdev/20181214181205.28812-1-pablo%40netfilter.org/

BPF comes to firewalls
https://lwn.net/Articles/747551/

Bringing the Power of eBPF to Open vSwitch (William Tu *et al.*, LPC 2018)
http://vger.kernel.org/lpc_net2018_talks/ovs-ebpf-lpc18-presentation.pdf

Using eBPF as a heterogeneous ABI (Jakub Kicinski, LPC 2018)
http://vger.kernel.org/lpc-bpf.html#session-8

DPDK documentation, *Berkeley Packet Filter Library*
http://doc.dpdk.org/guides/prog_guide/bpf_lib.html

Nothing yet on libkefir... Stay tuned!