eBPF for networking

Stefano Salsano – University of Rome Tor Vergata

stefano.salsano@uniroma2.it

Ackowledgements

- **We have reused public material available in several presentations authored by:**

  Fulvio Risso, Thomas Graf, Michael Kehoe, Fabian Ruffy, Suchakrapani Sharma, Sebastiano Miano

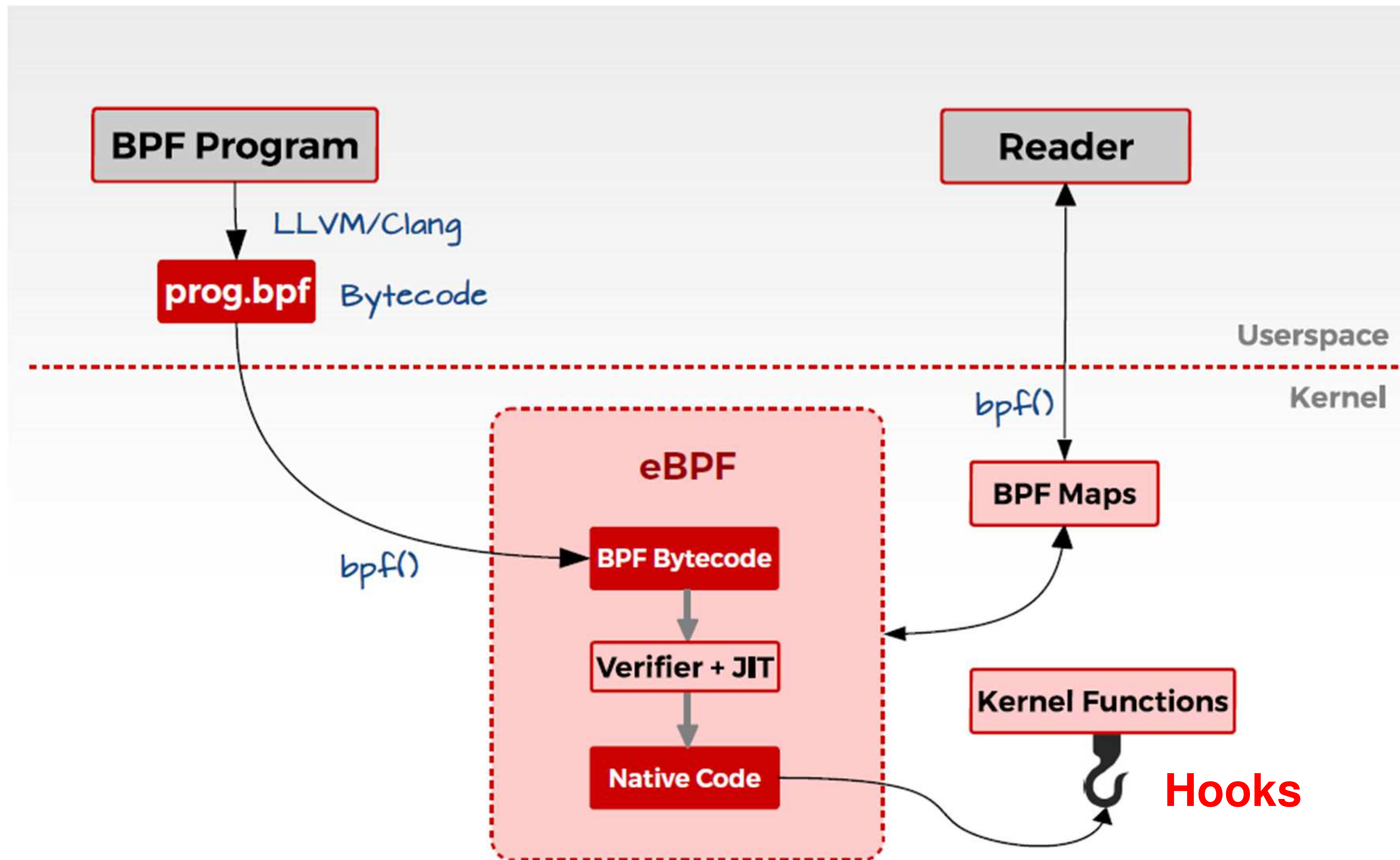
  (full references at the end)

eBPF Day

eBPF Day

- **Recall on some eBPF features**

eBPF architecture



https://hsdm.dorsal.polymtl.ca/system/files/eBPF-5May2017%20(1).pdf

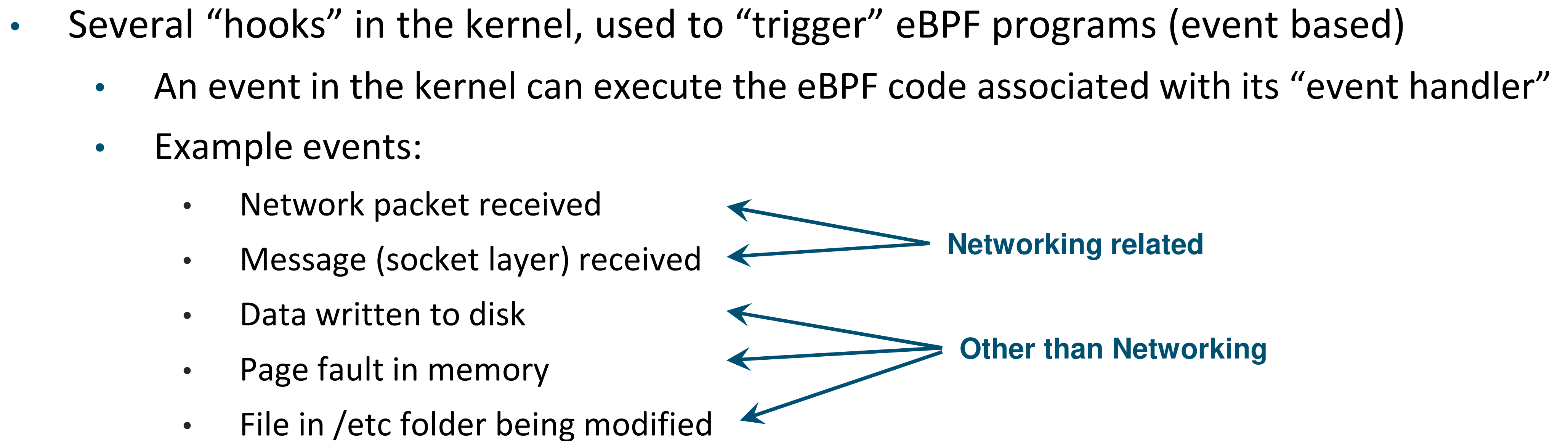
eBPF main features

- The eBPF VM implements a RISC-like assembly language in kernel space: User-defined, “sandboxed” bytecode executed by the kernel
- The eBPF Linux module enables arbitrary code to be dynamically injected and executed in the Linux kernel
- eBPF provides hard safety guarantees in order to preserve the integrity of the system (e.g. eBPF does not allow unbounded loops) – eBPF Verifier
- Several “hooks” in the kernel, used to “trigger” eBPF programs (event based)
- All interactions between kernel / user space are done through eBPF “maps”

eBPF main features – safety / verifier

- eBPF provides hard safety guarantees in order to preserve the integrity of the system (e.g. eBPF does not allow unbounded loops) – eBPF Verifier
  - “Sandbox” approach => no invalid memory access
  - The Verifier checks that the program has a maximum number of instructions (no unbounded loops) and that all accesses to memory are valid
  - Main consequences:
    - eBPF cannot execute arbitrary code (it’s not “Turing complete”)
    - the verification is done “statically” by checking all execution paths.
Heuristics needs to be used to speed up the verification
    - “false positives” : programs that are rejected by the verifier although they are valid

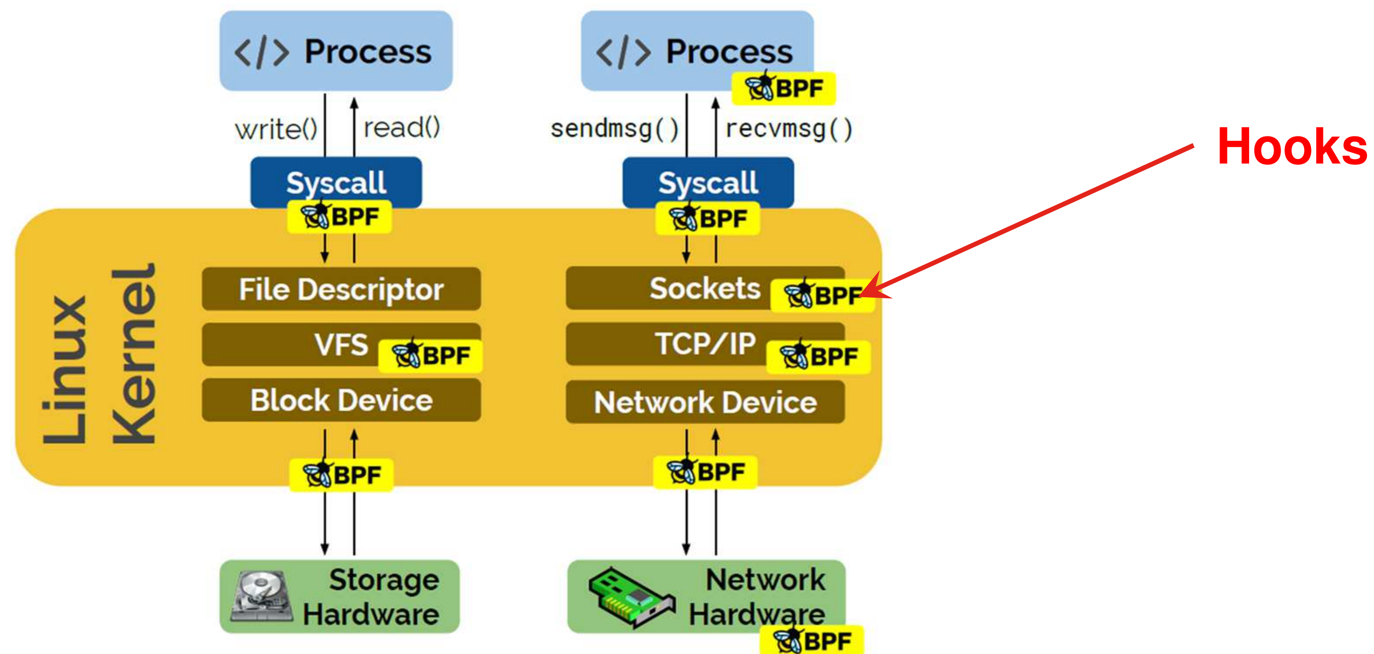
eBPF main features - hooks



http://site.ieee.org/hpsr-2018/files/2018/06/18-06-18-IOVisor-HPSR.pdf

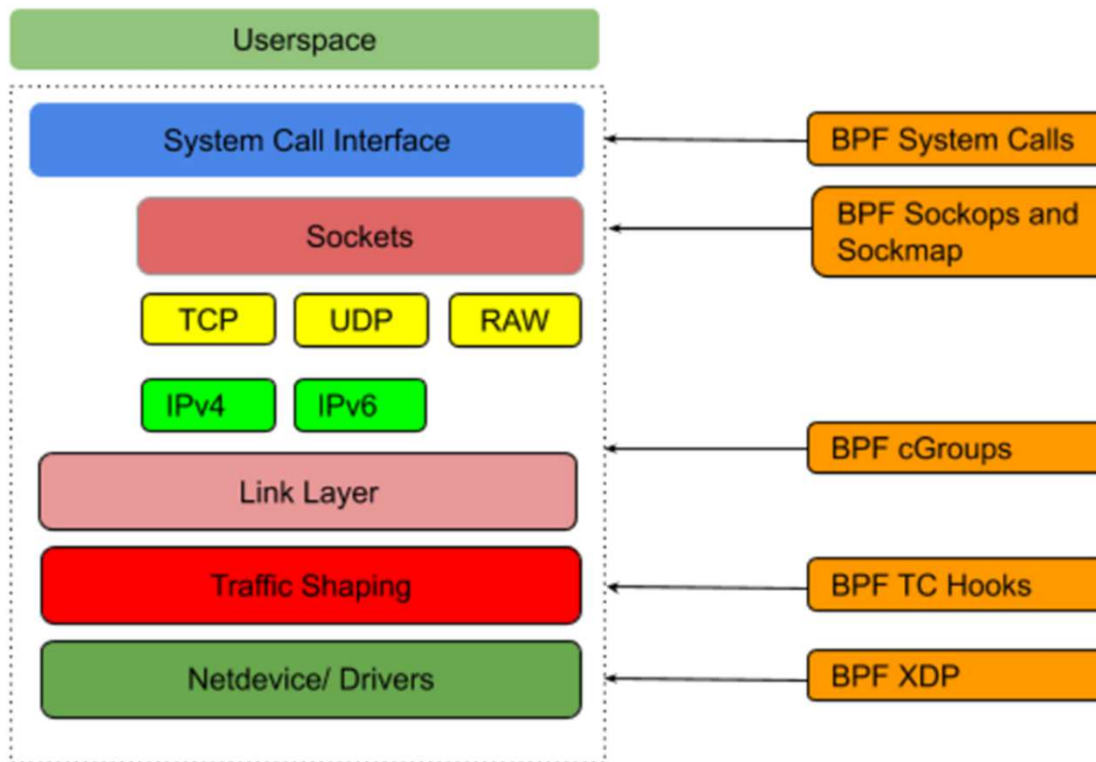
eBPF main features - hooks

- Several “hooks” in the kernel are used to “trigger” eBPF programs (event based)
  - not only for networking!



https://www.slideshare.net/ThomasGraf5/ebpf-rethinking-the-linux-kernel

eBPF hooks and program types



- For a given hook, a specific eBPF "program type" can be invoked.
- The "context" which is passed to the eBPF program depends on the hook.
- The capabilities of the eBPF program depend on the hook, i.e. different interactions with kernel (helper functions) can be invoked.

https://cyral.com/blog/how-to-ebpf-accelerating-cloud-native/

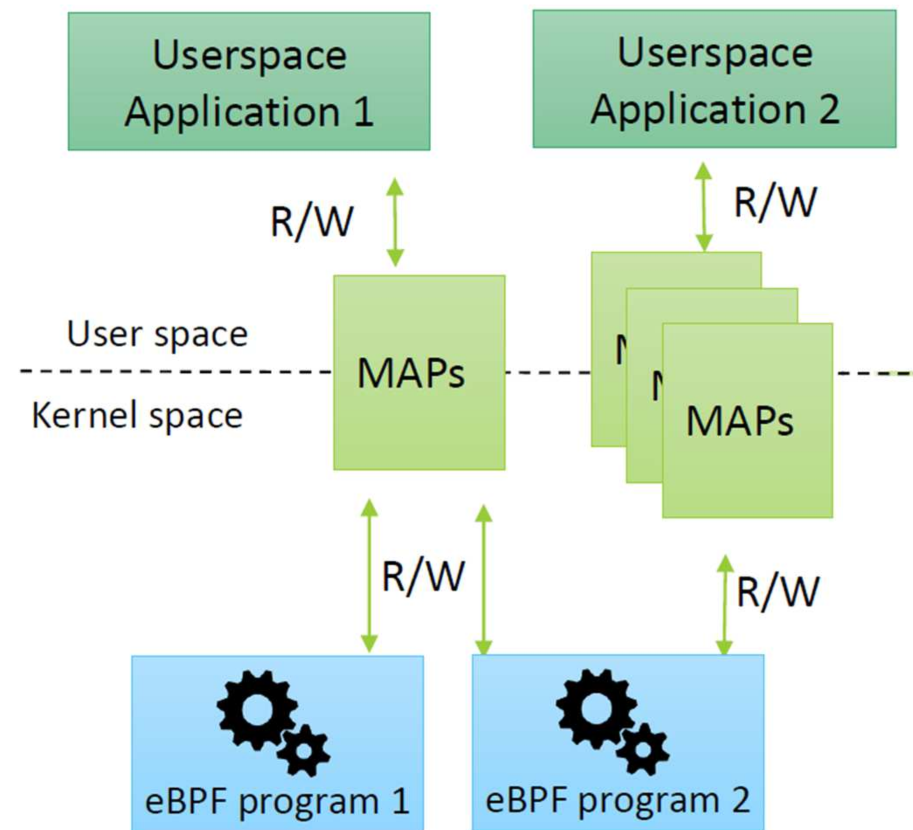
eBPF program types

```
enum bpf_prog_type {
    BPF_PROG_TYPE_UNSPEC,
    BPF_PROG_TYPE_SOCKET_FILTER,
    BPF_PROG_TYPE_KPROBE,
    BPF_PROG_TYPE_SCHED_CLS,
    BPF_PROG_TYPE_SCHED_ACT,
    BPF_PROG_TYPE_TRACEPOINT,
    BPF_PROG_TYPE_XDP,
    BPF_PROG_TYPE_PERF_EVENT,
    BPF_PROG_TYPE_CGROUP_SKB,
    BPF_PROG_TYPE_CGROUP_SOCK,
    BPF_PROG_TYPE_LWT_IN,
    BPF_PROG_TYPE_LWT_OUT,
    BPF_PROG_TYPE_LWT_XMIT,
    BPF_PROG_TYPE_SOCK_OPS,
    BPF_PROG_TYPE_SK_SKB,
};
```

https://www.slideshare.net/MichaelKehoe3/ebpf-basics-149201150

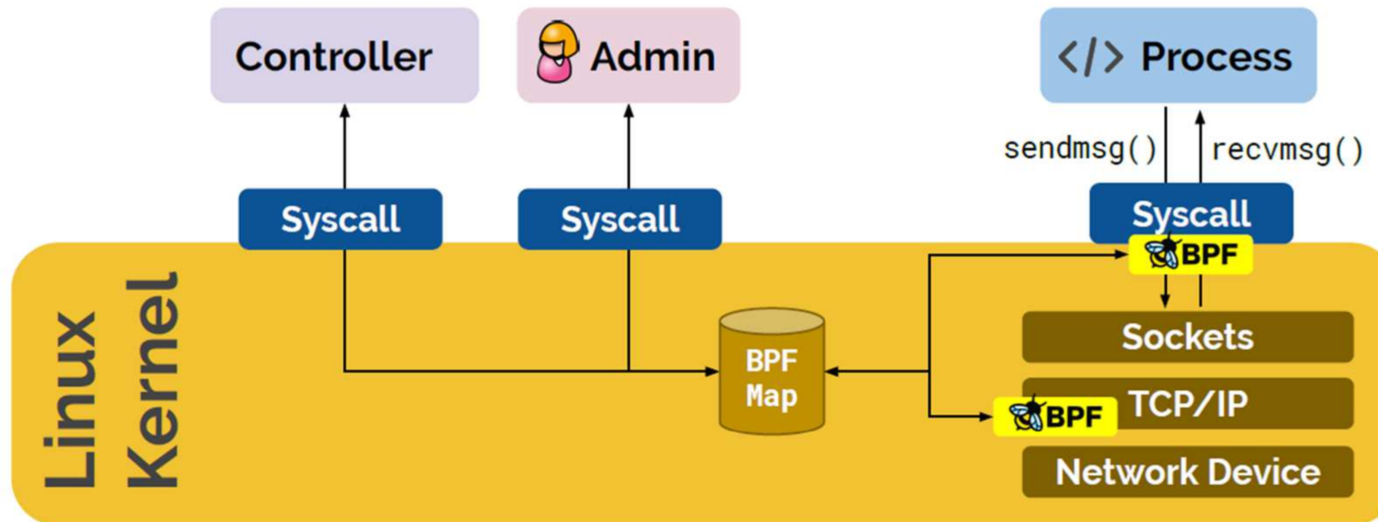
eBPF main features - maps

- All interactions between kernel / user space are done through eBPF “maps”
  - The maps can be shared with user space applications and among eBPF programs
  - The eBPF programs are “stateless”: the state is stored in the maps
  - The maps helps dealing with concurrency
  - Per-CPU maps



http://site.ieee.org/hpsr-2018/files/2018/06/18-06-18-IOVisor-HPSR.pdf

eBPF maps



Map Types:

- Hash tables, Arrays
- LRU (Least Recently Used)
- Ring Buffer
- Stack Trace
- LPM (Longest Prefix match)

What are Maps used for?

- Program state
- Program configuration
- Share data between programs
- Share state, metrics, and statistics with user space

https://www.slideshare.net/ThomasGraf5/ebpf-rethinking-the-linux-kernel

eBPF maps

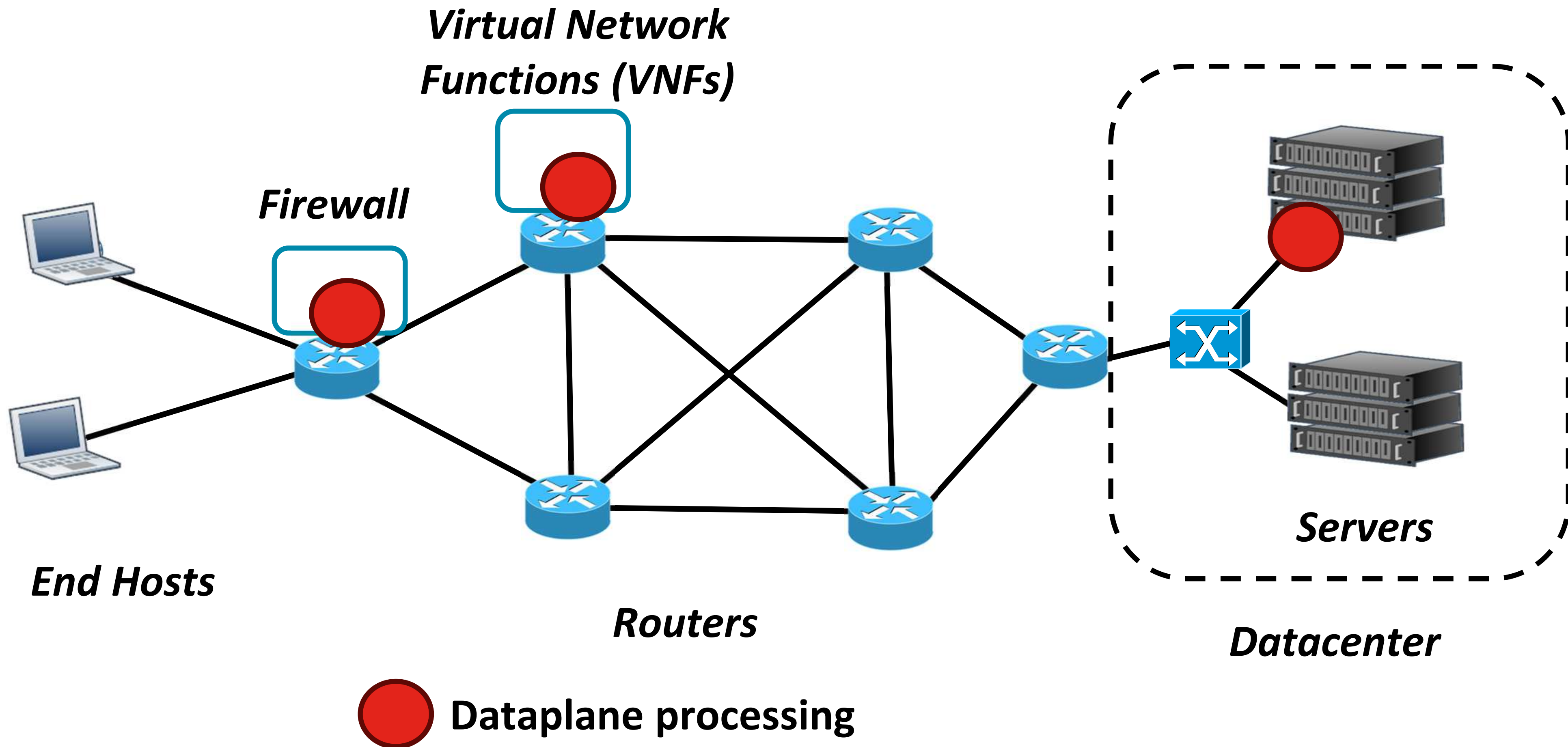
HASH - A hash table
ARRAY- An array map, optimized for fast lookup speeds
PROG_ARRAY - An array of FD's corresponding to eBPF programs
PER_CPU_ARRAY - A per-CPU array, used to implement histograms
PERF_EVENT_ARRAY - Stores pointers to struct perf_event
CGROUP_ARRAY – Stores pointers to control groups
PER_CPU_HASH – A per-CPU hash table
LRU_HASH - A hash table that only retains the most recently used items
LRU_PER_CPU_HASH - A per-CPU hash table that only retains the most recently used items
LPM_TRIE - A longest-prefix match true, good for matching IP addresses
STACK_TRACE - Stores stack traces
ARRAY_OF_MAPS - A map-in-map data structure
HASH_OF_MAPS – A map-in-map data structure
DEVICE_MAP - For storing and looking up network device references
SOCKET_MAP – Stores and looks up sockets and allows redirection

https://www.slideshare.net/MichaelKehoe3/ebpf-basics-149201150

eBPF Day

- **eBPF for networking**

Networking scenarios



Different types of processing

- Routing
- Tunneling (encap/decap, e.g. VXLANs)
- NATs - NAPTs
- Firewalls
- Load Balancers
- Application-level processing
- Deep packet inspection

Dataplane Softwarization

eBPF Day

- **Which are the design choices for a "Software routers" or a "Packet processing device", based on:**
  - Generic purpose processors
  - Linux OS
- **with the (obvious) goal to optimize the performance…**

Different types of processing

eBPF Day

Routers/Universal CPEs etc

```
L2 Switch
VLAN/ Q-inQ
L3 Router
NAT
ACL (mac, ip, port)
```

Broadband Network Gateway

```
L2 Switch
L3 Router
Classification
hQoS
ACL
TM (Policing, Metering)
```

Cloud Load Balancer

```
Bonding
VLAN / Q-in-Q
NAT
ACL (blacklist)
TM (policing, metering) L4
Load Balancer
```

Intrusion Prevention System

```
L2 Switch
L3 Router
Classification
NAT
ACL (mac, ip, port)
```

Key requirements : support for new services & performance

Solutions

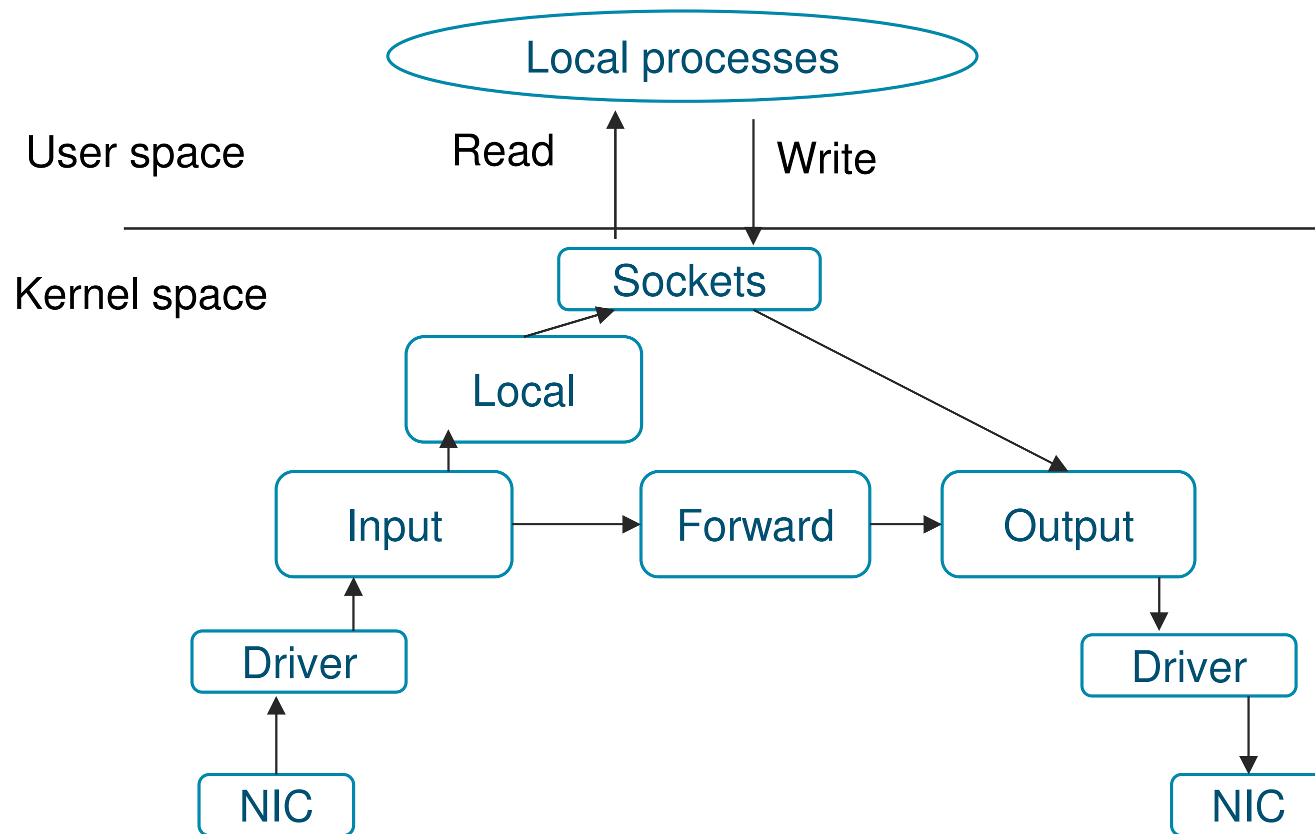
Forwarding and processing framework/tools

- **Linux Kernel**
- **VPP (FD.io)**
- **OvS (Open vSwitch)**
- **Cilium**

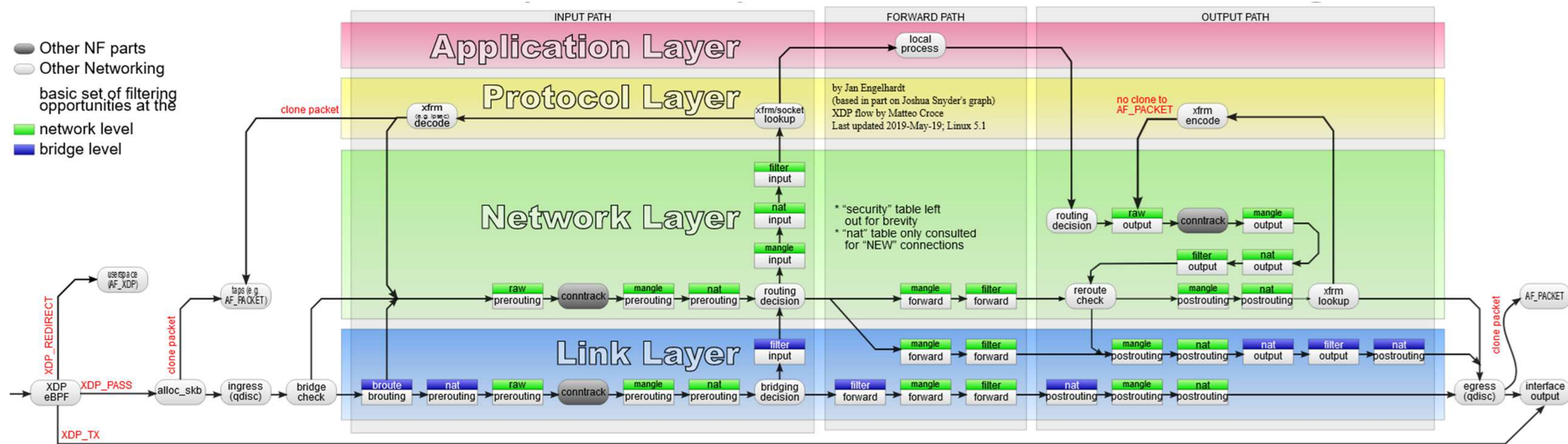
Acceleration frameworks (“fast IO”)

- **DPDK (Hardware acceleration)**
- *Netmap*
- **eBPF**

Packet processing in Linux kernel (very simplified)

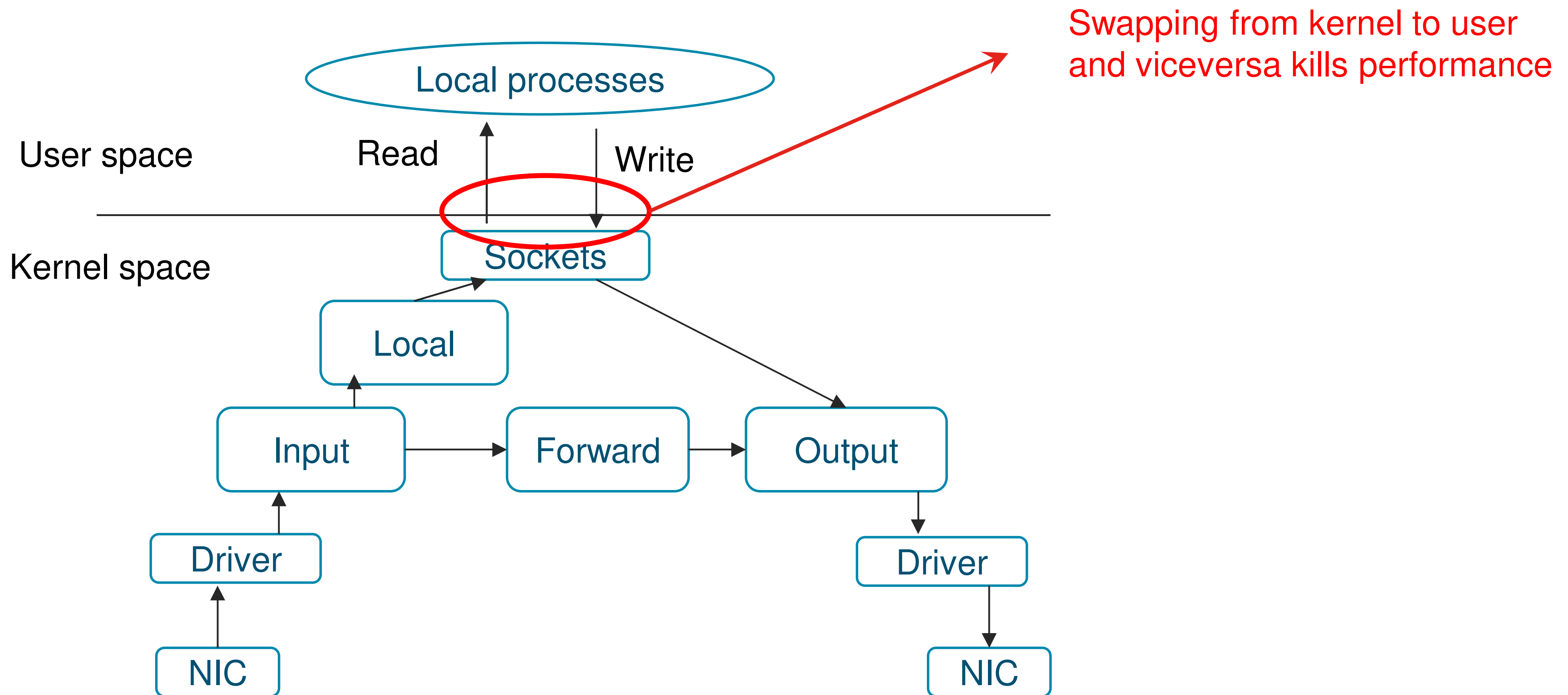


Packet processing in Linux kernel

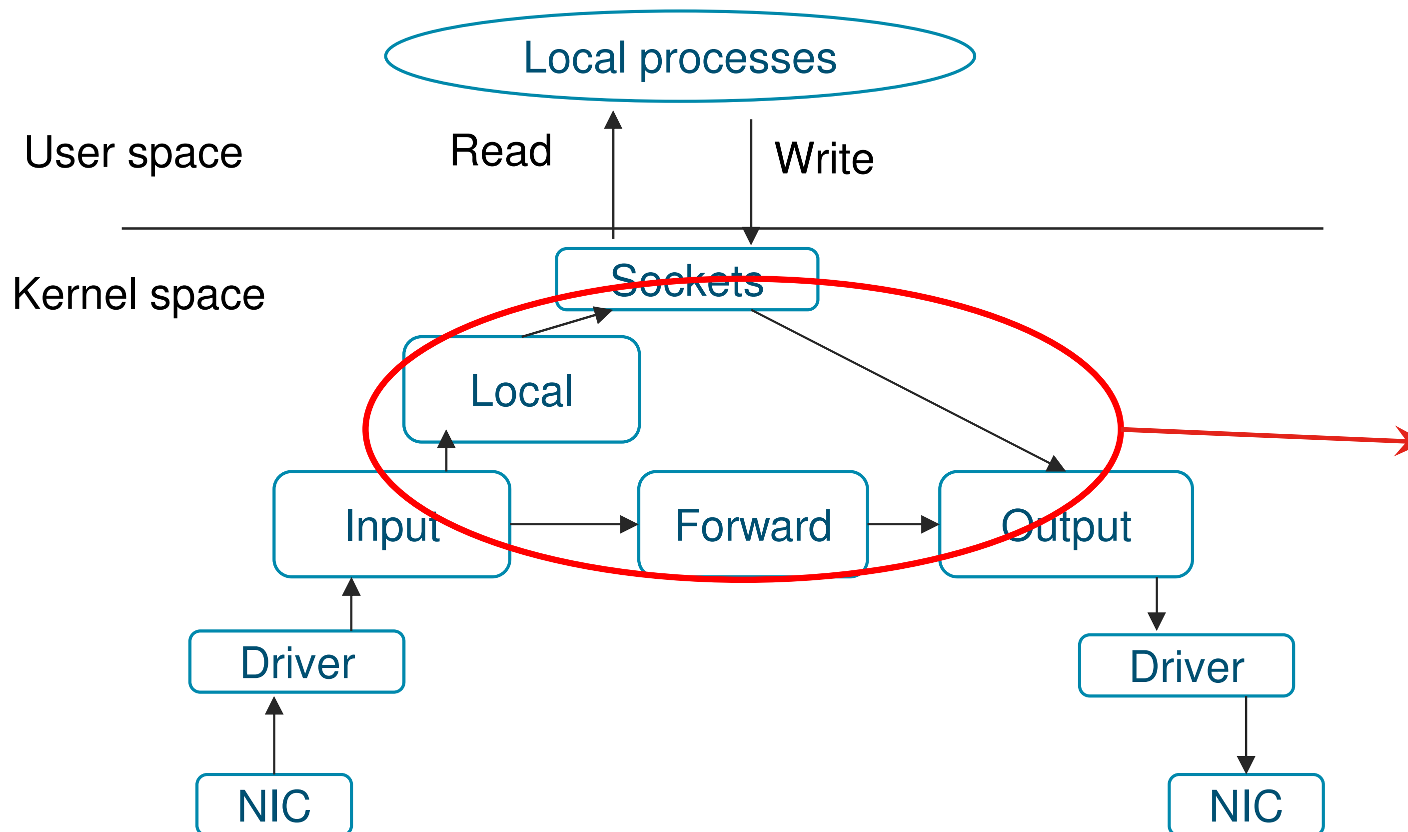


Source: https://commons.wikimedia.org/wiki/File:Netfilter-packet-flow.svg

Key issues

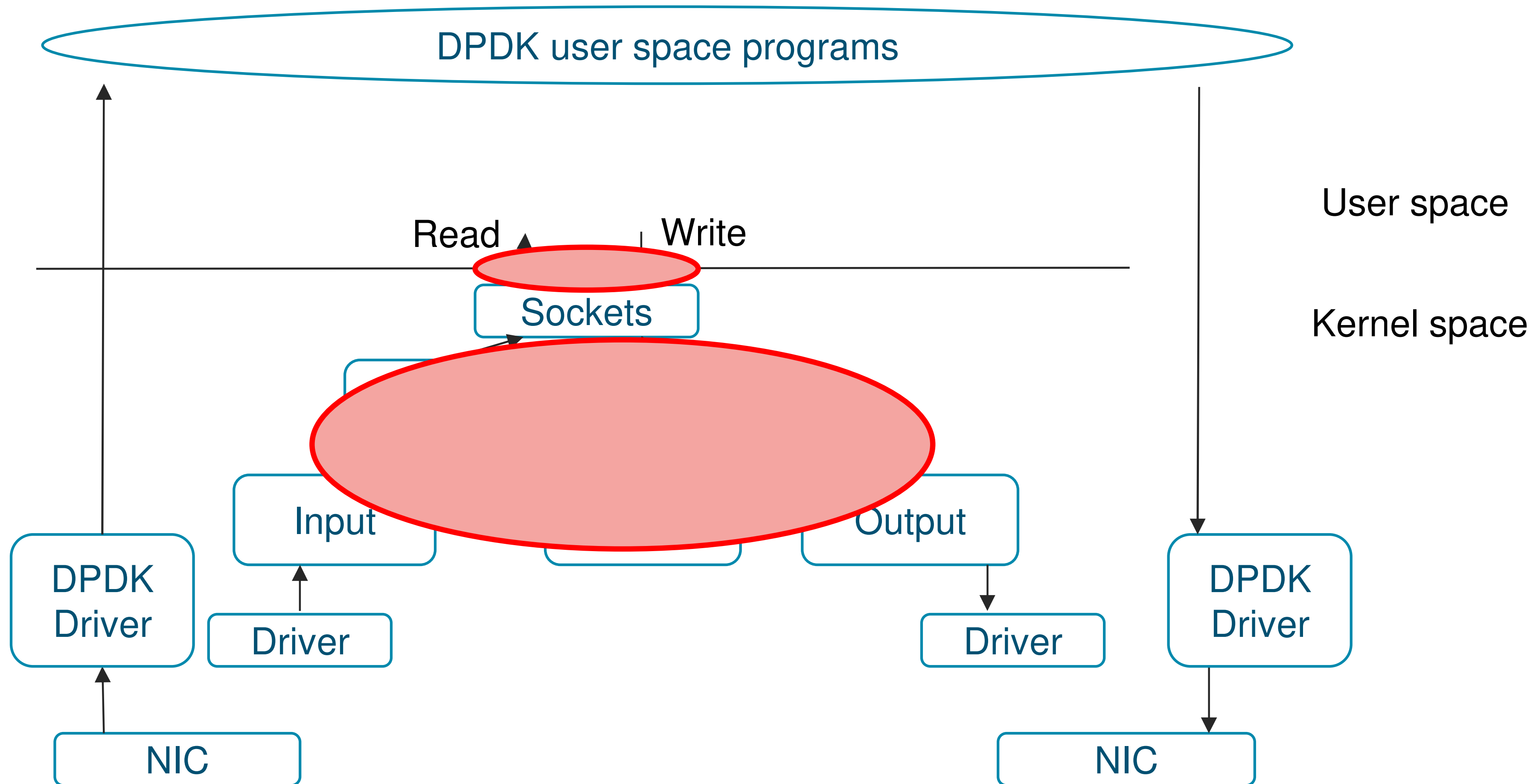


Key issues

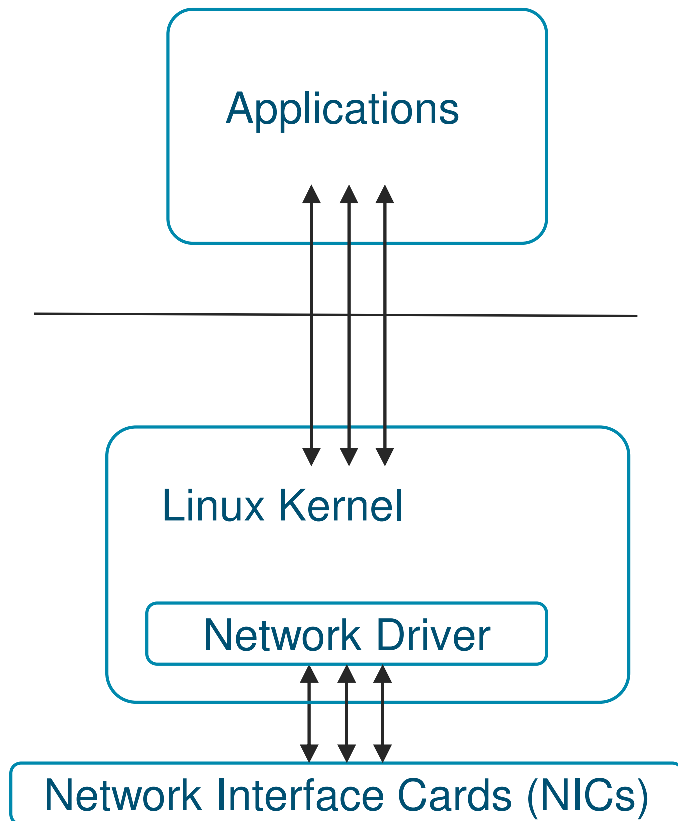


Kernel processing is designed for “generality” and is not optimized for specific use cases

Kernel bypass solution (e.g. DPDK)

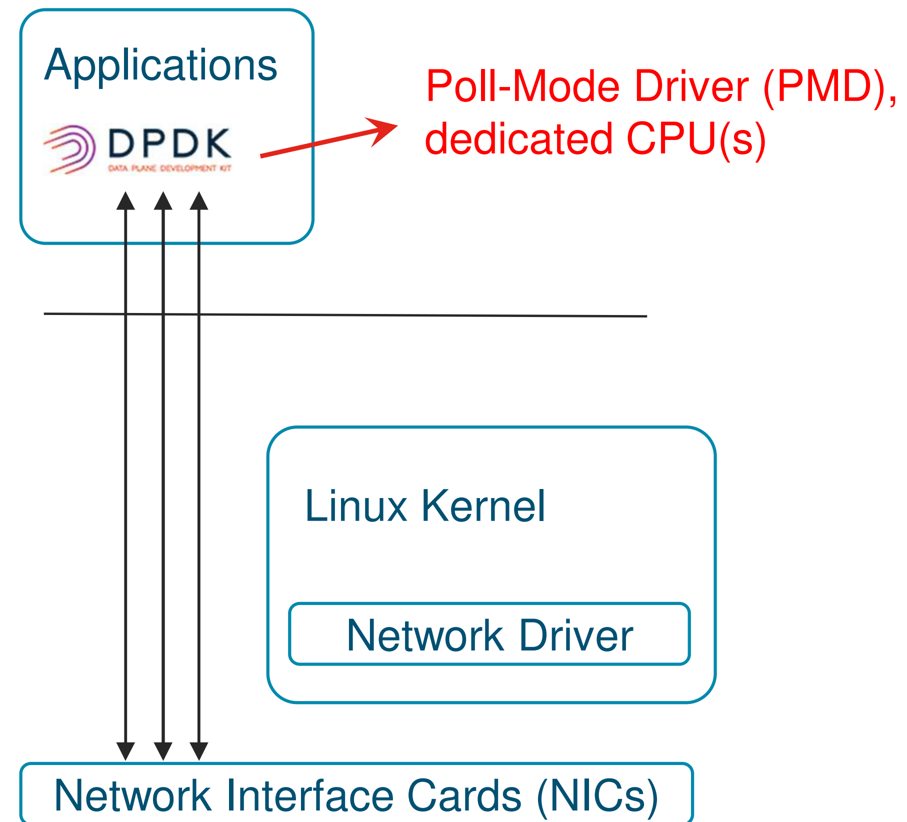


Kernel bypass solution (e.g. DPDK)



User space

Kernel space



Without DPDK

With DPDK

Considerations for in-kernel solutions

eBPF Day

Overall requirements

- **Coexistence/integration with kernel-based processing**
- **Hardware independence**

Issues to be solved…

- **Complexity of interaction with existing features**
- **Security issues / risks of “freezing” the kernel**
- **Performance aspects**

eBPF - extended Berkeley Packet Filter

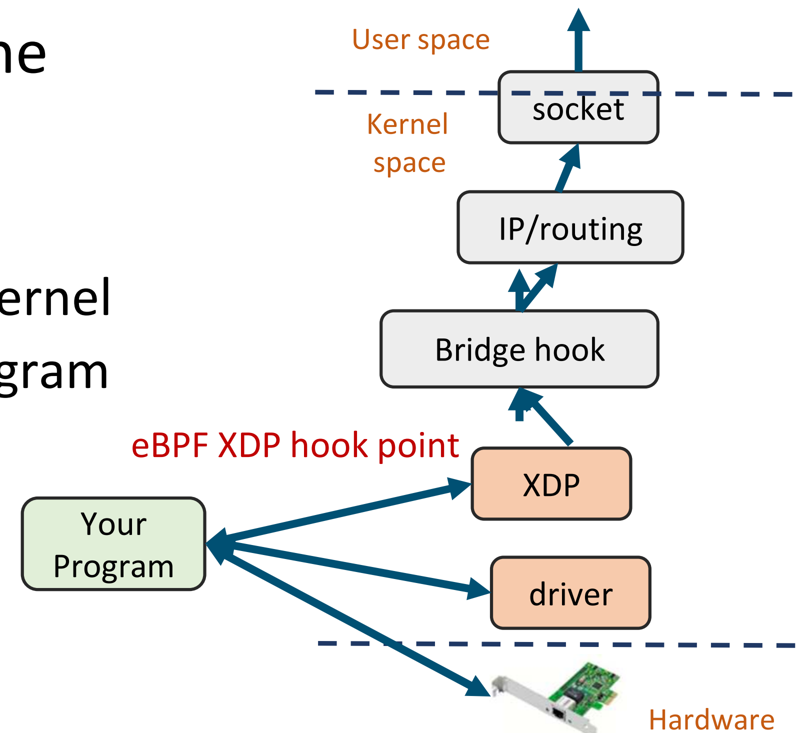
eBPF Day

eBPF provides an "in-kernel" solution addressing the issues

- **Complexity…** — **well defined interaction model with "hooks"**
- **Security / "freezing"** — **"restricted" language, Virtual Machine, verification approach**
- **Performance aspects** — **specific hooks offer "high performance"**

Recall of eBPF features

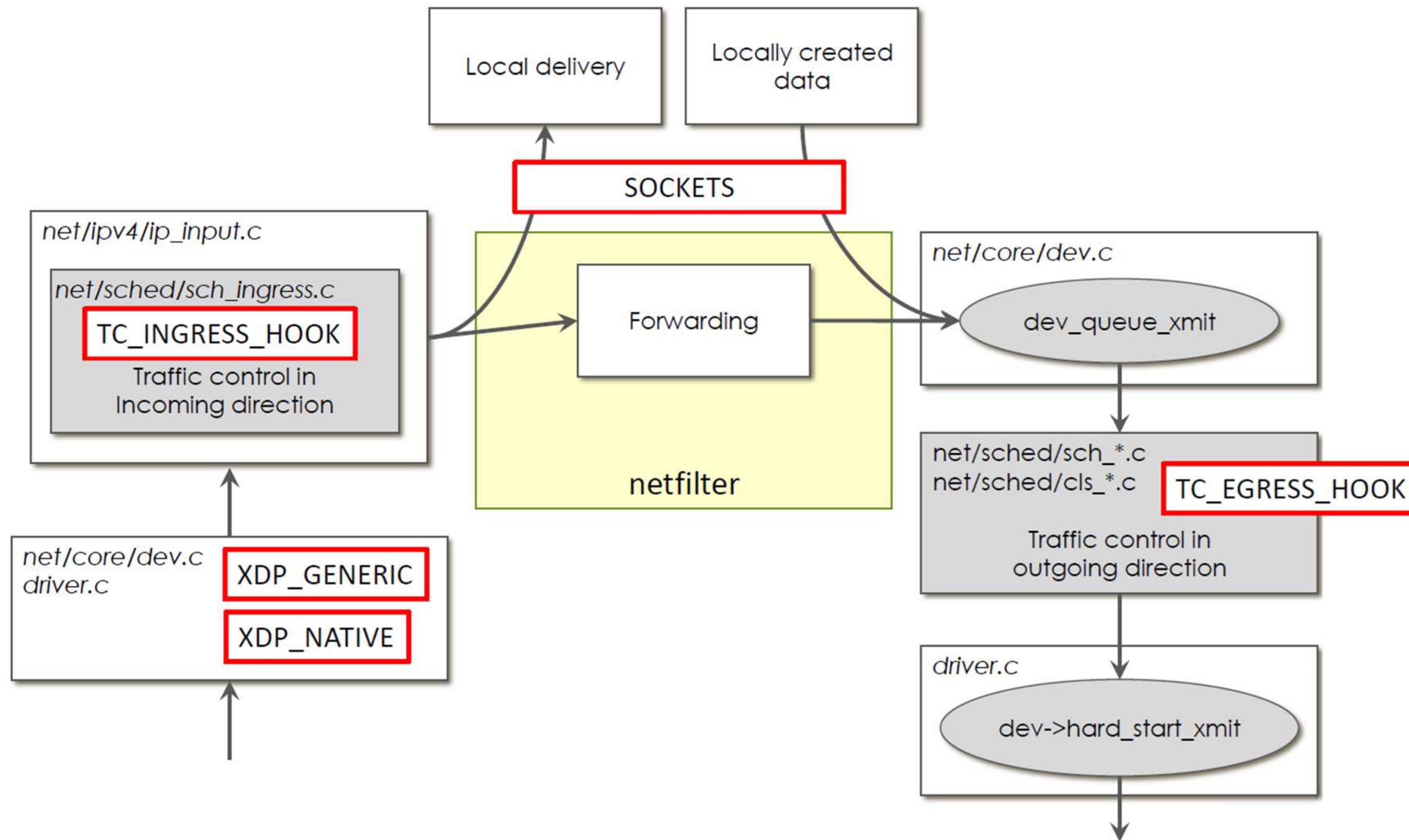
- **Virtual Machine** (or **Virtual CPU**) running in the Linux kernel
- Provides:
  - The ability to write restricted C and run it in the kernel
  - A set of kernel hook points invoking the eBPF program
- Extensible, safe and fast
- Alternative to user-space networking



A programmable data plane in the Linux kernel!

https://ruffy.eu/presentations/p4c-xdp-lpc18-presentation.pptx

eBPF hooks (for networking)



http://site.ieee.org/hpsr-2018/files/2018/06/18-06-18-IOVisor-HPSR.pdf

eBPF program types (for networking)

SOCKET-RELATED
- SOCKET_FILTER: Filtering actions (e.g. drop packets)
- SK_SKB: Access SKB and socket details with a view to redirect SKB's
- SOCK_OPS – Catch socket operations

CGROUPS
- CGROUP_SKB – Allow or deny network access on IP egress/ ingress
- CGROUP_SOCK – Allow or deny network access at various socket-related events

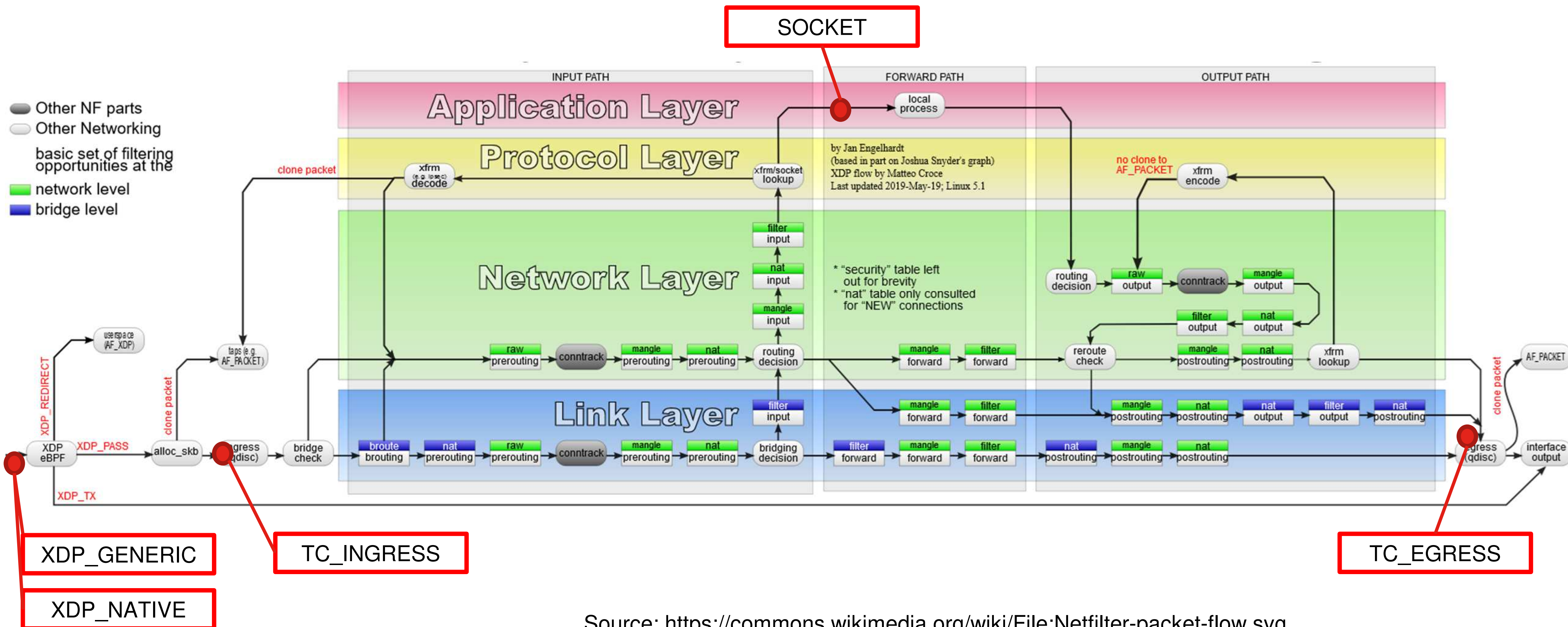
LIGHTWEIGHT TUNNELS
- LWT_IN – Examine inbound packets for lightweight tunnel deencapsulation
- LWT_OUT – Implement encapsulation tunnels for specific destination routes
- LWT_XMIT – Allowed to modify content and prepend a L2 header

TRAFFIC CONTROL
- SCHED_CLS: A network traffic-control classifier
- SCHED_ACT: A network traffic-control action

XDP
- XDP: Allows access to packet data as early as possible (DDoS mitigation/ Load-balancing)

https://www.slideshare.net/MichaelKehoe3/ebpf-basics-149201150

Packet processing in Linux kernel



Source: https://commons.wikimedia.org/wiki/File:Netfilter-packet-flow.svg

Socket buffers (sk_buff)

eBPF Day

- **The socket buffer (abbreviated as sk_buff) is the fundamental data structure used to represent network packets**

```
struct sk_buff {
  struct   sk_buff          *next;     // next sk_buff in the list
  struct   sk_buff          *prev;     // previous sk_buff
  struct   sk_buff_head     *list;     //list we are on
  struct   sock             *sk;       // socket we belong to
  struct   timeval          stamp;     //arrival timestamp
  struct   net_device       * dev;     // "output" device
  ... many other!!!
```

Socket buffers (sk_buff)

eBPF Day

- **sk_buffs contains the attributes and metadata associated with a packet, allowing the kernel to handle packet processing, routing, and transmission**

```
union {
        struct tcphdr    *th;
        struct udphdr    *uh;
        struct icmphdr  *icmph;
        …
} h;  //transport level header
        //(tcp, udp, icmp….)
```

```
union {
        struct iphdr    *iph;
        struct ipv6hdr  *ipv6h;
        struct arphdr   *arph;
        …
} nh; //network level header
        //(ip, ipv6, arp….)
```

eBPF helper functions ("BPF-helpers")

eBPF Day

BPF-helpers are "offered" by the Linux kernel and can be called from eBPF programs. For example, they can be used to:

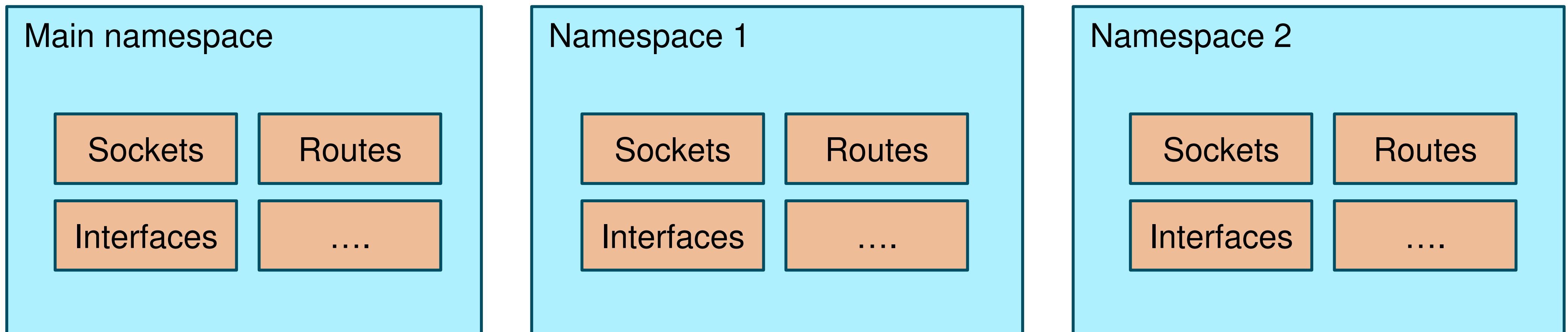
- print debugging messages
- get the time since the system was booted
- interact with eBPF maps
- manipulate network packets

More that 150 BPF-helpers are listed in the man page:
https://man7.org/linux/man-pages/man7/bpf-helpers.7.html

Caveat: "This manual page is an effort to document the existing eBPF helper functions. But as of this writing, the BPF sub-system is under heavy development. New eBPF program or map types are added, along with new helper functions. Some helpers are occasionally made available for additional program types. So in spite of the efforts of the community, this page might not be up-to-date."

Network namespaces

- **Linux kernel keeps networking resources separated into “namespaces”**



Thank you. Questions?

Contacts

Stefano Salsano
University of Rome Tor Vergata
stefano.salsano@uniroma2.it

References

- **Fabian Ruffy, Linux Network Programming with P4**
  https://ruffy.eu/presentations/p4c-xdp-lpc18-presentation.pptx
- Michael Kehoe, (c|e)BPF Basics
  https://www.slideshare.net/MichaelKehoe3/ebpf-basics-149201150
- Michael Kehoe, eBPF Workshop
  https://www.slideshare.net/MichaelKehoe3/ebpf-workshop
- Fulvio Risso, Toward Flexible and Efficient In Kernel Network Function Chaining with IOVisor, IEEE HPSR 2018,
  http://site.ieee.org/hpsr-2018/files/2018/06/18-06-18-IOVisor-HPSR.pdf
- https://www.iovisor.org/technology/ebpf (quite old)

References

- Suchakrapani Sharma, “Trace Aggregation and Collection with eBPF” (2017) https://hsdm.dorsal.polymtl.ca/system/files/eBPF-5May2017%20(1).pdf
- How to use eBPF for accelerating Cloud Native applications https://cyral.com/blog/how-to-ebpf-accelerating-cloud-native/
- Thomas Graf, “Cilium - BPF & XDP for containers” (2016) https://www.slideshare.net/Docker/cilium-bpf-xdp-for-containers-66969823
- Thomas Graf, “eBPF - Rethinking the Linux Kernel” (2020) https://www.slideshare.net/ThomasGraf5/ebpf-rethinking-the-linux-kernel
- Thomas Graf, “BPF & Cilium - Turning Linux into a Microservices-aware Operating System” (2018) https://www.slideshare.net/ThomasGraf5/bpf-cilium-turning-linux-into-a-microservicesaware-operating-system