OCP SUMMIT

March 20-21
2018
San Jose, CA

OPEN. FOR BUSINESS.

Accelerating Load Balancing programs using HW-Based Hints in XDP

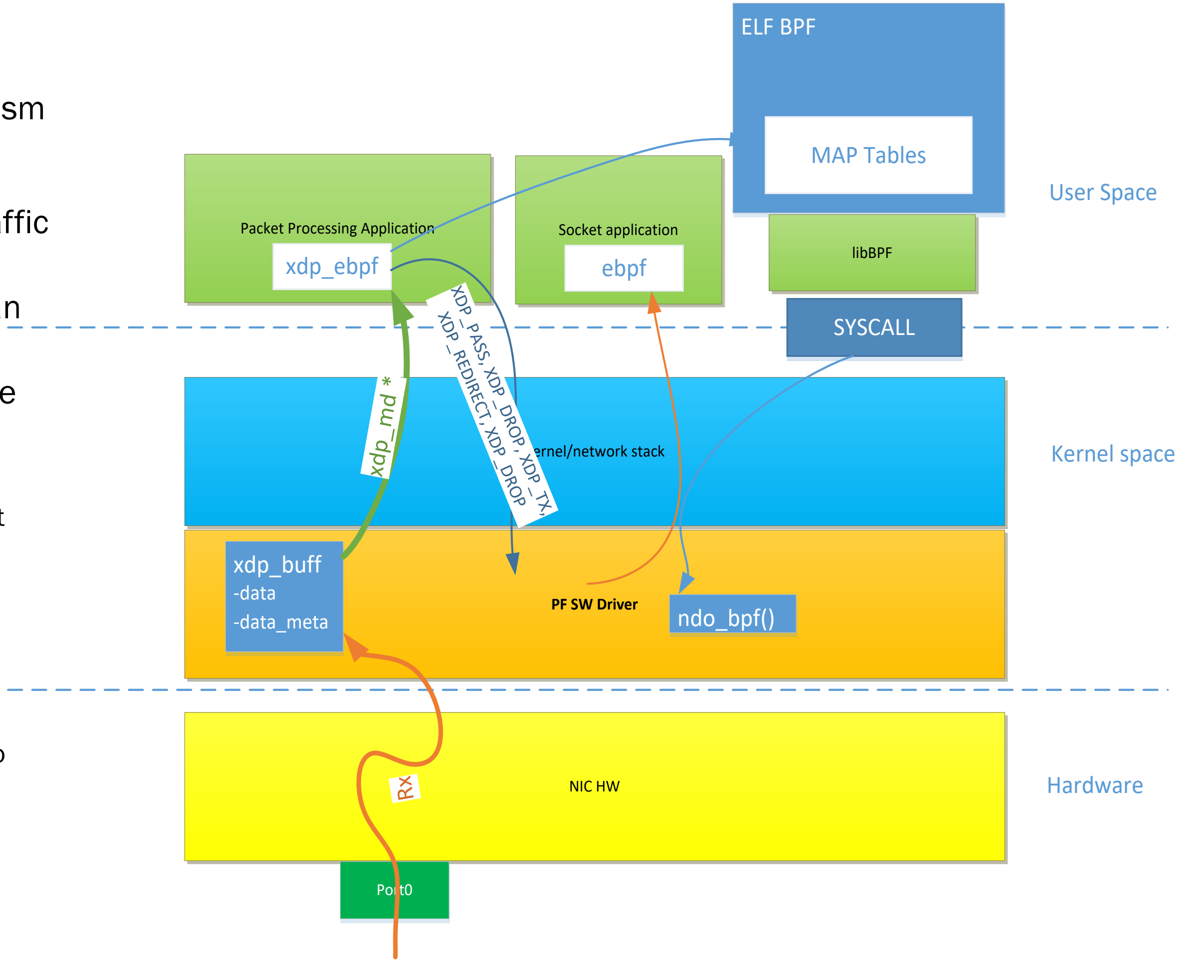
PJ Waskiewicz, Network Software Engineer
Neerav Parikh, Software Architect
Intel Corp.

Agenda

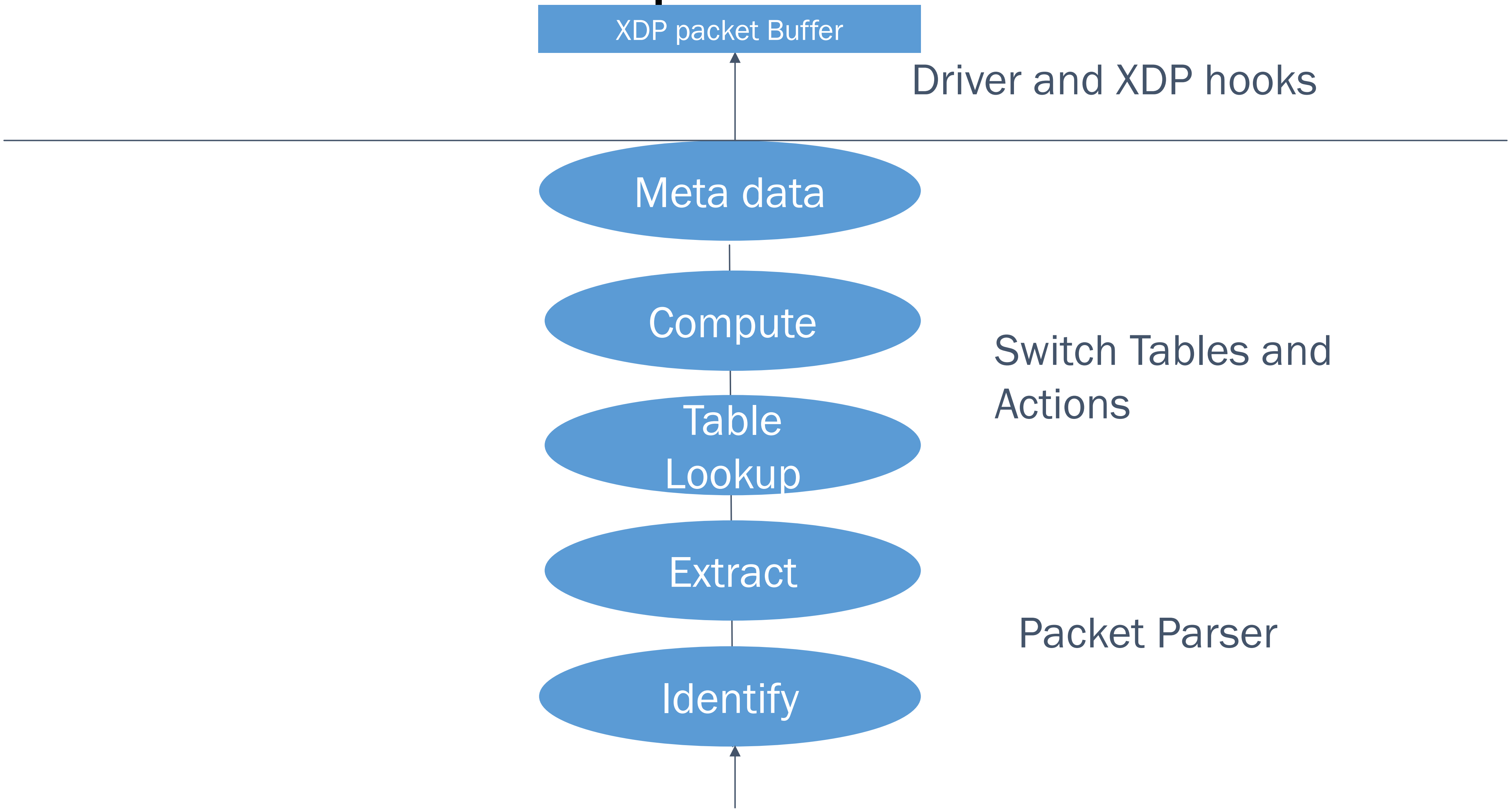
- Overview eXpress Data path (XDP) Software Model
- NIC Hardware Capability
- Our Goals
- HW hints for XDP
- Metadata Passing
- Programming Hardware hints
- Initial Performance Results
- Next-steps
- Questions

XDP Software Model

- eXpress Data Path (XDP) evolved as a Linux in kernel mechanism bypassing regular kernel network stack to allow faster packet processing for certain use-cases
- Typical XDP use-case applications: Firewall, Load balancer, Traffic monitoring, etc.
- XDP utilizes Linux kernel eBPF infrastructure that associates an eBPF program into NIC SW drivers data path
- XDP programs are continuing to evolve and are becoming more complex
- A typical XDP program does following:
  - Packet parsing: Identify the packet type (IPv4/v6, TCP/UDP, etc.) and extract packet header information
  - Based on the use-case then the XDP program
    - may monitor incoming traffic on the network
    - manipulate packets based on incoming traffic
    - compute hash or xsums for modified packets
    - make packet forwarding decisions based on some map table lookups
    - Set up some meta data and return status back to the NIC SW driver to indicate what to do with that packet
      - XDP_PASS: Pass it to regular kernel network stack
      - XDP_DROP: Drop the packet
      - XDP_TX: Tx the packet out
      - XDP_REDIRECT: Redirect the packet to another network device



NIC Hardware Rx Pipeline

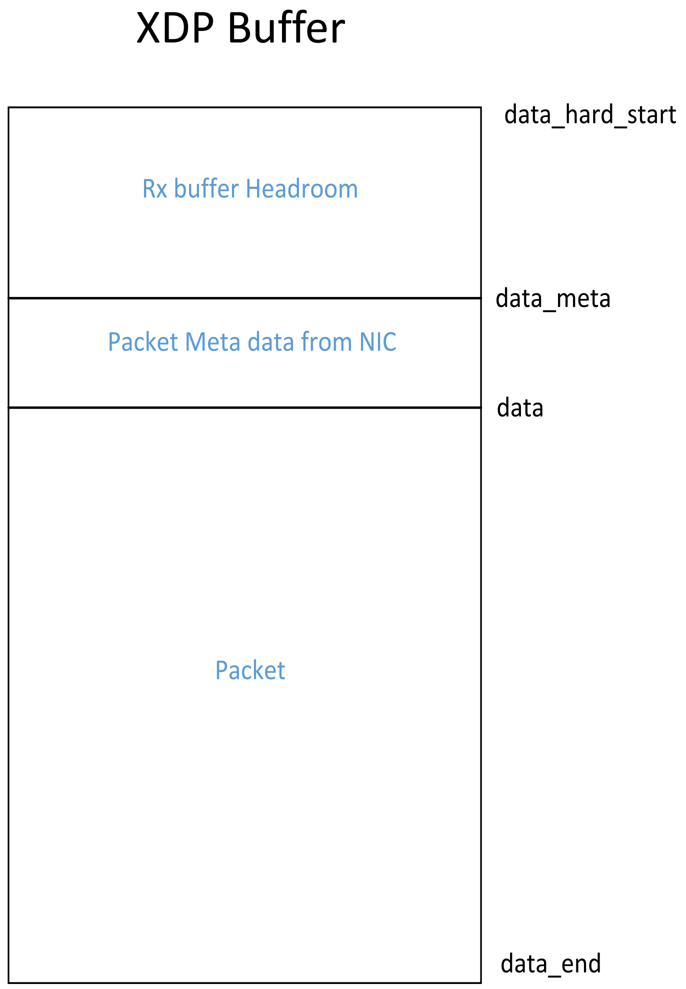


Our Goal

- What can present-day NIC Hardware can do to help:
  - Accelerate what is being done in XDP programs in terms of packet processing
  - Offset some of the CPU cycles used for packet processing
- Keep it consistent with XDP philosophy
  - Avoid kernel changes as much as possible
  - Keep it Hardware agnostic as much as possible
  - Best effort acceleration
  - A frame work that can change with changing needs of packet processing
- Expose the flexibility provided by programmable packet processing pipeline to adapt to XDP program needs

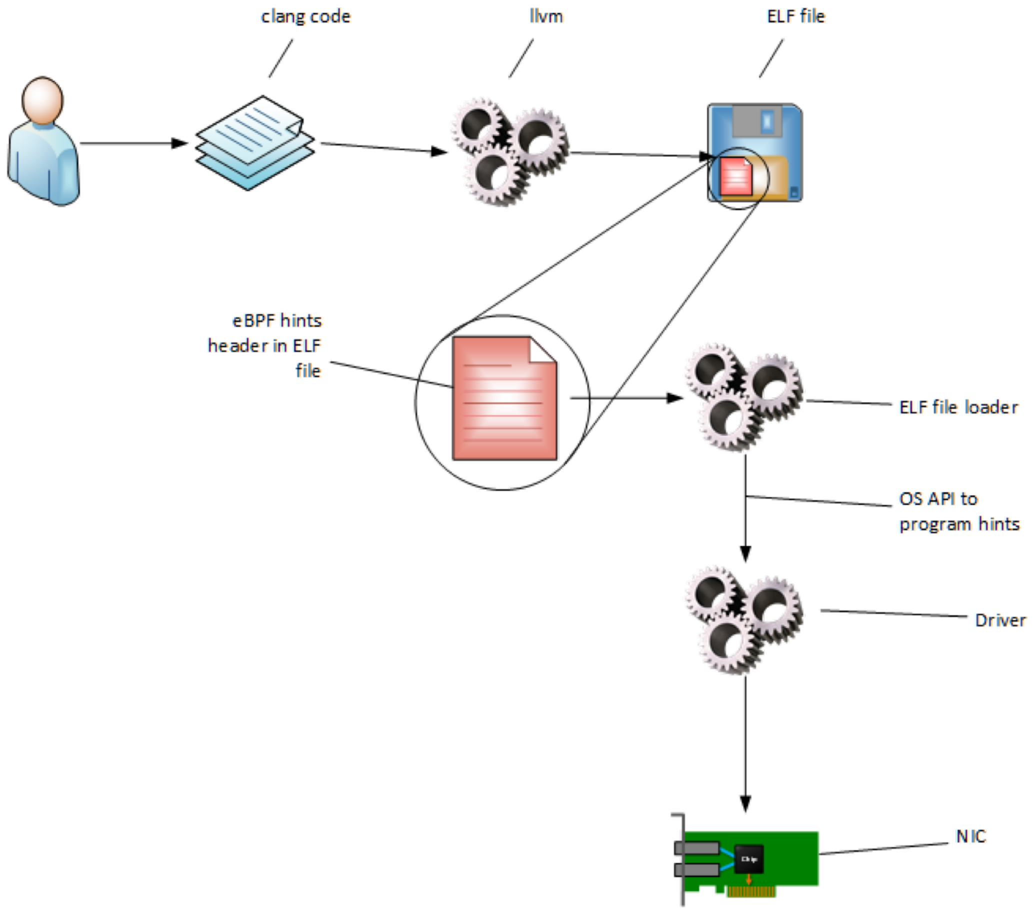
Two problems to solve

- How do you dynamically program the Hardware to get the XDP program the right kind of packet parsing help?
- How to pass the packet parsing/map lookup hints that the HW provides with every packet into the XDP program so that it can benefit from it?



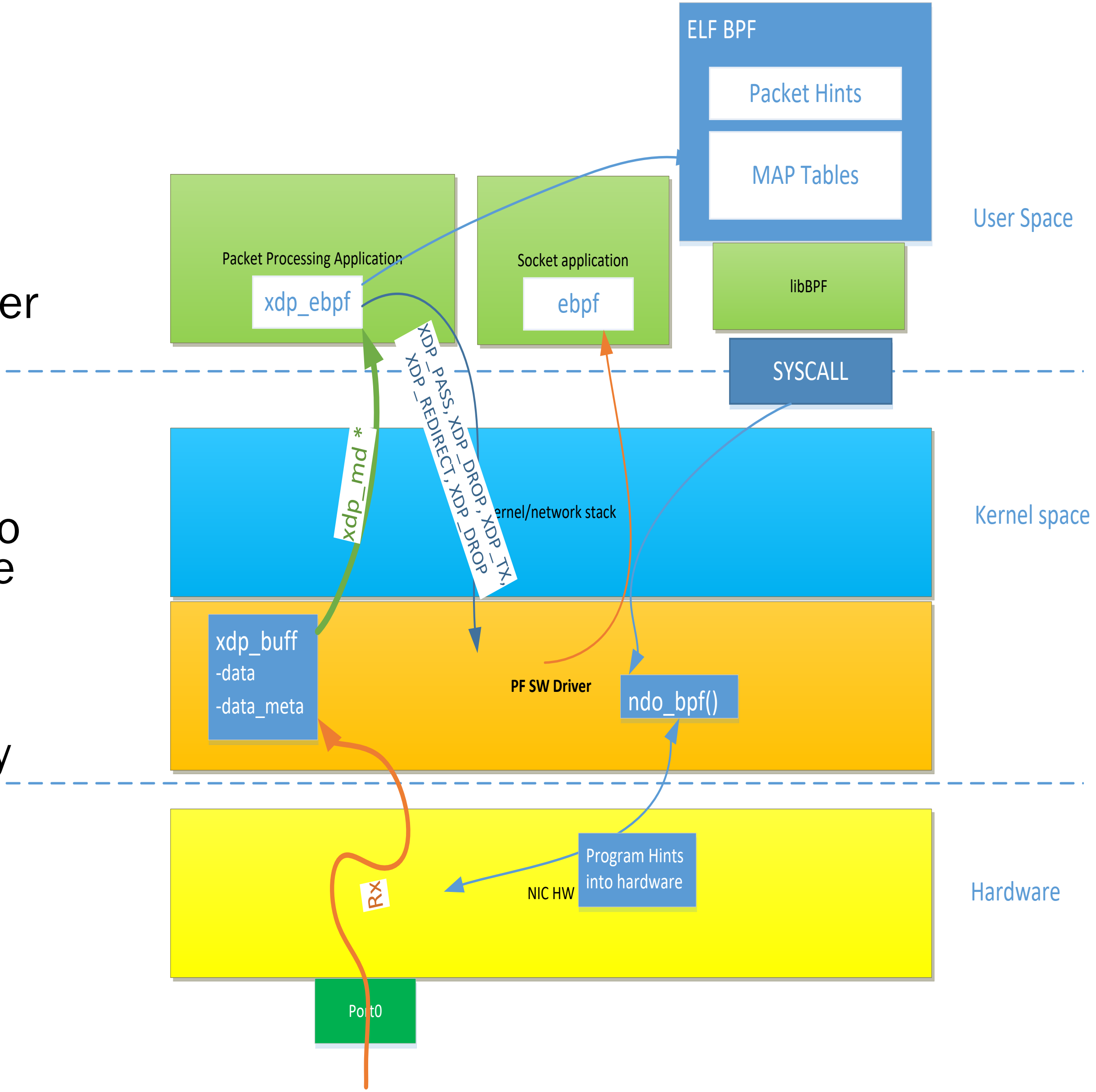
Programming HW hints

- Defining HW hints as ELF sections of eBPF program and program them at time of load
- Example fields to extract for a packet:
  - Packet types: IPv4/IPv6, TCP, UDP, SCTP, ICMP
  - Packet Header data: SMAC/DMAC, SADDR/DADDR, next protocol header offset
  - Processing hints: Rx Hash on packet fields, TCP connection flags (SYN/SYN-ACK/FIN/RST)

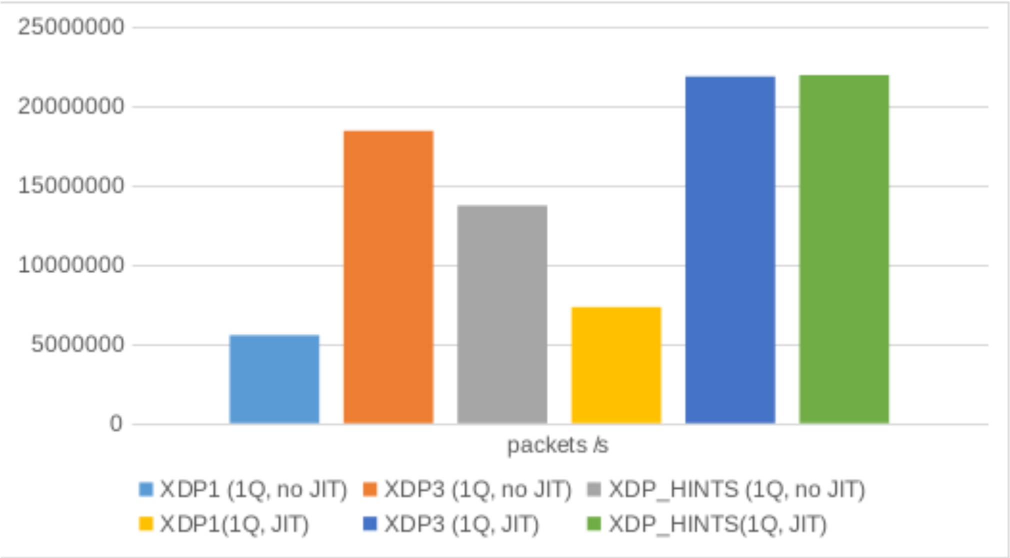


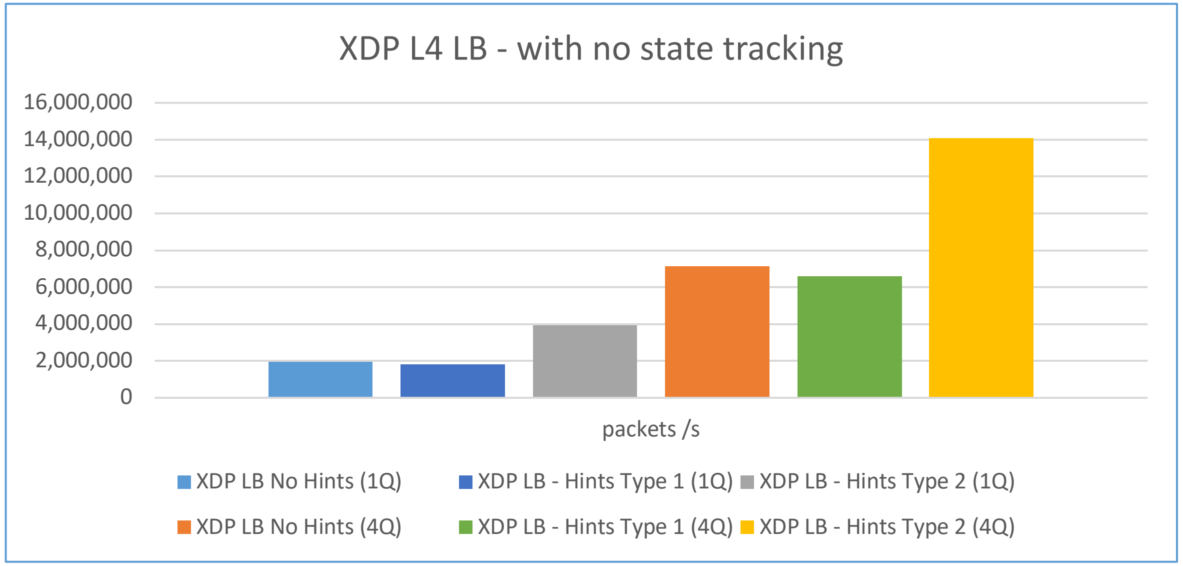
Programming Flow

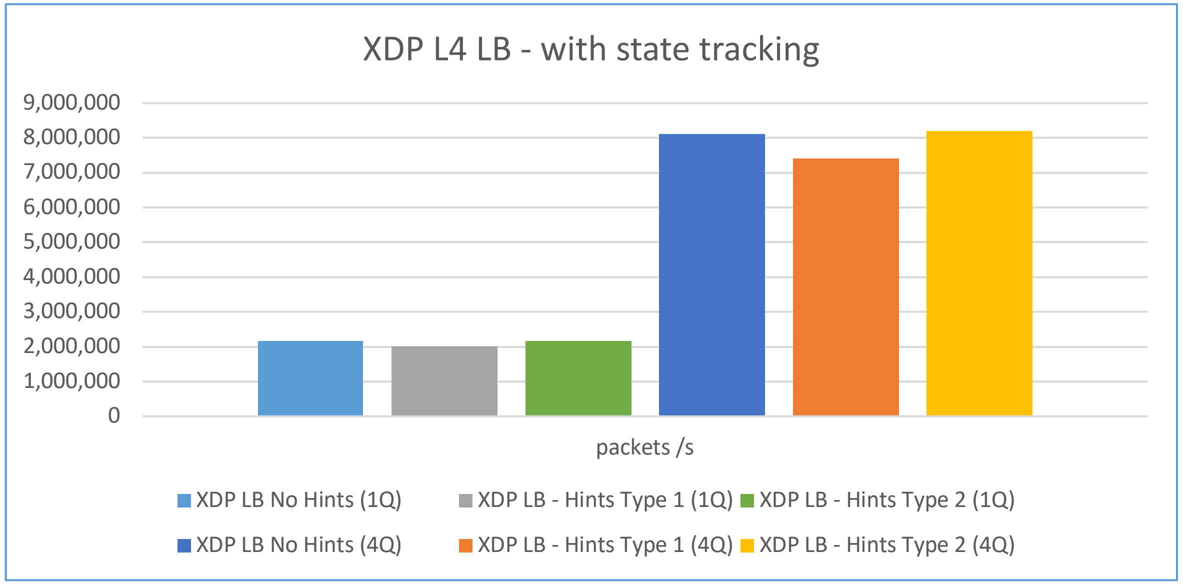
- The ELF sections that carry HW programming hints need to be passed over to the driver in some form so that it can program the HW accordingly
- Introduce some new helper ndo_offload_xdp_hints() or traverse the required hints when ndo_bpf() is called so that the driver can call to extract what the XDP program can use as hints and program the HW accordingly.
- The driver hides all the HW programming details, the hints format is generic for any HW.
- A given HW may or may not be able to provide all the hints.
- It's a best effort mechanism to offload what the HW can support.



Performance with and without hints







- XDP1: Linux kernel sample, parses packet to identify protocol, count and drop
- XDP3: Zero packet parsing (best case scenario), just drop all packets
- L4 LB: L4 Load Balancer sample application with multiple Virtual IP tunnels, forwarding packets to destination based on hash calculations and lookup
- XDP_HINTS: Uses packet type (IPv4/v6, TCP/UDP, etc.) provided by driver as meta data, no packet parsing, count and drop
- Hints Type 1: Protocol Type (IPv4/v6, TCP or UDP, etc.)
- Hints Type 2: Additional hints from type 1 including packet data like source/destination IP addresses, source/destination ports, packet hash index (RSS) generated by hardware

Next steps

- Initial performance results using HW hints with simple XDP programs and programs that don't do much state tracking are promising
- Don't see much benefit with programs that do state tracking
- Continued testing with newer Xeon systems and upstream Linux kernels
- Prototyping of eBPF-based HW hint programming needs to be completed to allow creation of RFC patches to be sent to Linux kernel networking, iovisor.org and eBPF community in general for wider feedback
- Call for action: OCP Networking community involvement? http://www.opencompute.org/projects/networking

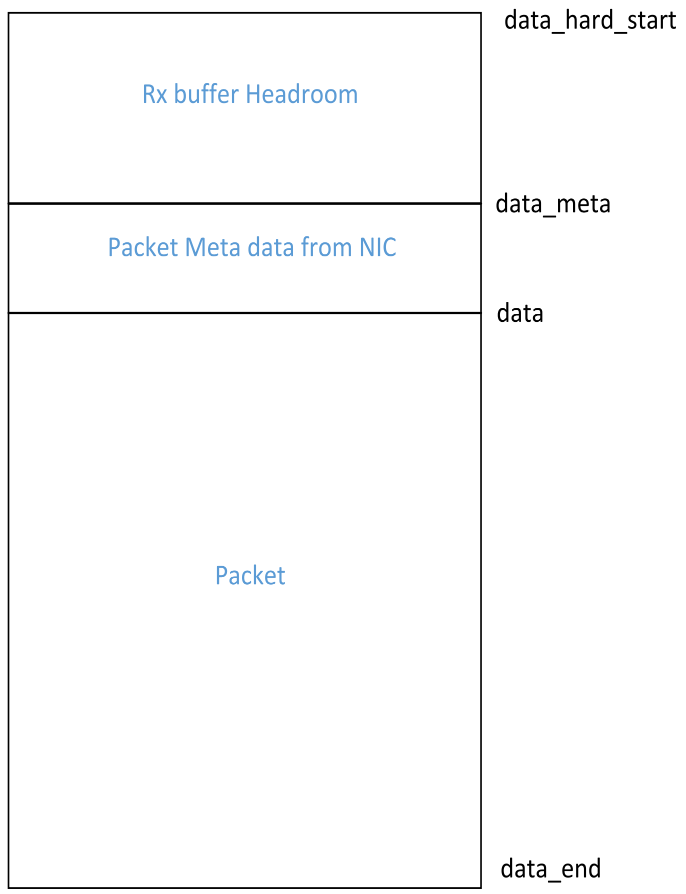
Questions?

OCP
SUMMIT

Backup

Metadata layouts – what to do?

- Approach 1: Common layout independent of underlying HW
  - Requires community agreement on common structures
  - Would be in the UAPI
- Approach 2: Vendor libraries in eBPF libraries
  - Requires XDP/eBPF programs to detect underlying hardware
- Approach 3: Chained XDP programs
  - Lightweight “shim” would contain vendor-specific logic
  - Tail-call larger program with parsed metadata to run rest of logic



HW Hints

Parsing Hints

Type of HW hint	Size	Description
Packet Type	U16	A unique numeric value that identifies an ordered chain of headers that were discovered by the HW in a given packet.
Header offset	U16	Location of the start of a particular header in a given packet. Example start of innermost L3 header.
Extracted Field value	variable	Example Inner most IPv6 address
Hash fields and type	variable	Hash on packet type and selected fields, selected hash type

Map Offload

Match	U32	Match a packet on certain fields and the values, provide a SW marker as a hint if the packet matches the rule

Packet Processing Hints

Checksum	U32	A total packet Checksum
Packet Hash	U32	Hash value calculated over specified fields and a given key for a given packet type
Ingress Timestamp	U64	Packet timestamp as it arrives

ELF Special Headers to request HW hints

```
struct bpf_hw_hints_def SEC("hw hints") rx_offset = {
        .type = PACKET_OFFSET_INNER_L4,
        .size = sizeof(__u16),
        };
```

```
struct bpf_hw_hints_def SEC("hw hints") rx_ptype = {
        .type = PTYPE,
        .size = sizeof(__u16),
        };  /* PTYPE values should be agreed upon between the SW and
the HW providing the hints, the driver may have to do the translation
between the two */
```

```
struct bpf_hw_hints_def SEC("hw hints") rx_match = {
        .type = PACKET_MATCH,
        .fields = {PTYPE, INNER_L3_SRC, INNER_L4_SRC},
        .mask = { 0xff, 0.0.ff.ff, 0xffff},
        .value = { 0x10,  10.10.20.2,  65},
        .result = 25 /* This hints adds a match rule into Hw, which creates a SW defined result when Hw
finds a match */
        .size = sizeof(__u32),
        };
```