XDP ACCELERATION USING NIC META DATA

Neerav Parikh, PJ Waskiewicz (Intel Corporation, Networking Division)
Saeed Mahameed (Mellanox)

Linux Plumbers Conference, Nov. 2018
Vancouver, BC, Canada

Network Division

Overview

- XDP Acceleration – Netdev 2.1 Recap
- XDP Performance Results
  - L4 Load Balancer
  - xdp_tx_ip_tunnel
- XDP NIIC Rx Metadata Requirements
- XDP NIC Rx Metadata Programming Model
- Next steps

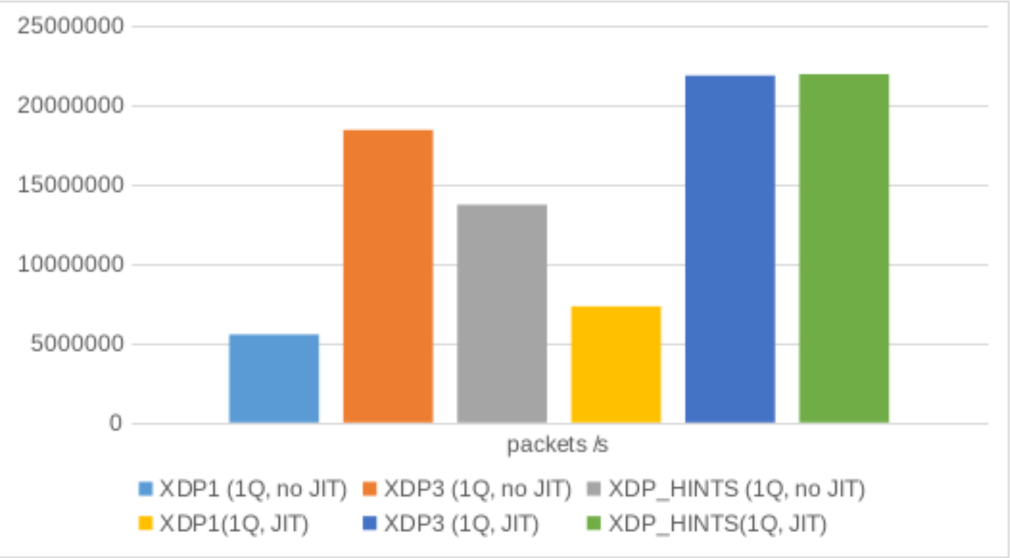
XDP Acceleration – Netdev 2.1 Recap

- What can present-day NIC HW 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 HW agnostic as much as possible
  - Best effort acceleration
  - A framework that can change with changing needs of packet processing
- Expose the flexibility provided by programmable packet processing pipeline to adapt to XDP program needs
- Help design the next generation hardware to take full advantage of XDP and the kernel framework

- How do you dynamically program the Hardware to get the XDP program the right kind of packet parsing help?

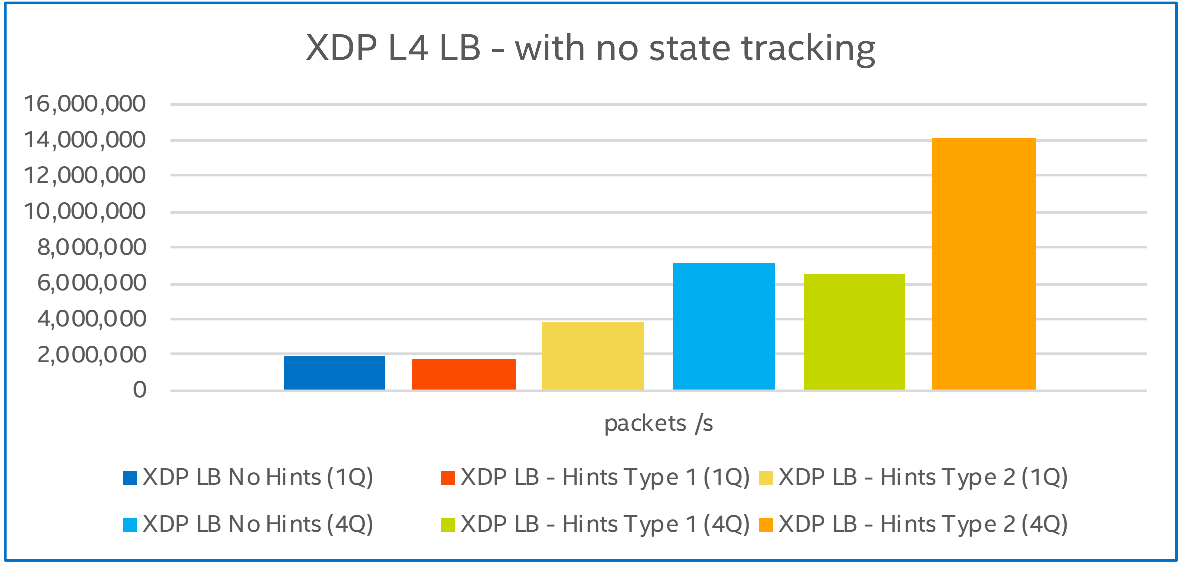
- How to pass the packet parsing/map lookup hints that the HW provides with every packet into the XDP program so that it can benefit from it?

Netdev 2.1 Recap - Performance data



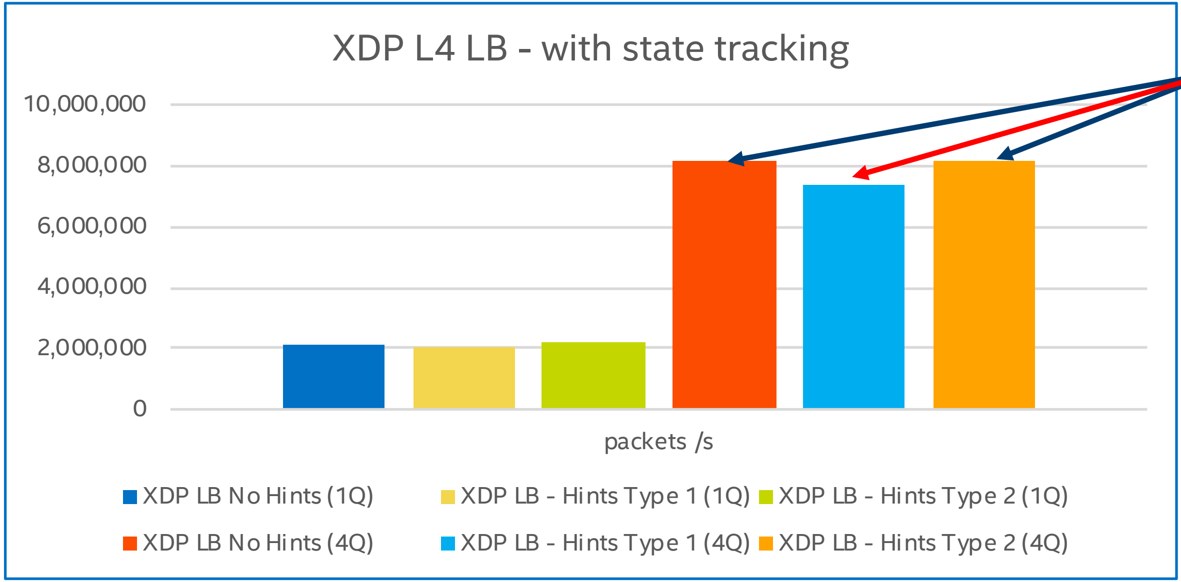
- XDP1: Linux kernel sample, parses packet to identify protocol, count and drop
- XDP3: Zero packet parsing (best case scenario), just drop all packets
- XDP_HINTS: Uses packet type (IPv4/v6, TCP/UDP, etc.) provided by driver as meta data, no packet parsing, count and drop

L4 Load balancer Performance



- L4 LB: L4 Load Balancer sample application with multiple Virtual IP tunnels, forwarding packets to destination based on hash calculations and lookup
- 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

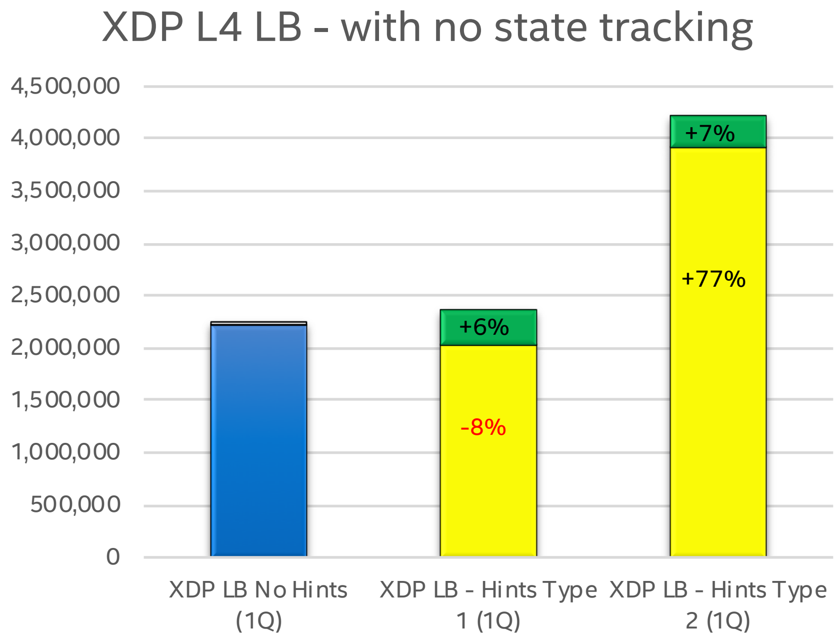
L4 Load balancer Performance

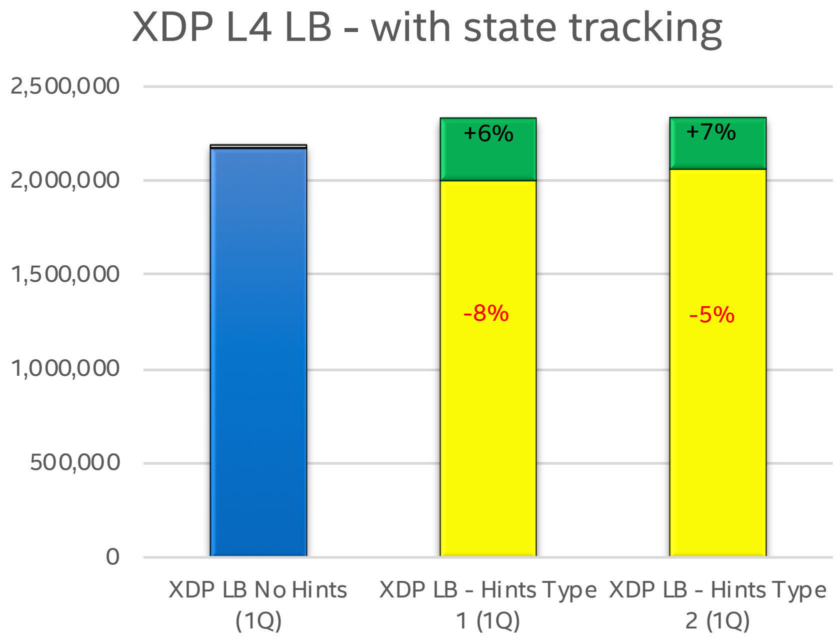


No visible advantage in performance with just packet parsing hints when XDP application is doing state tracking and connection management.

https://git.kernel.org/pub/scm/linux/kernel/git/jkirsher/next-queue.git/log/?h=XDP-hints-EXPERIMENTAL

L4 Load balancer Performance Analysis Projected



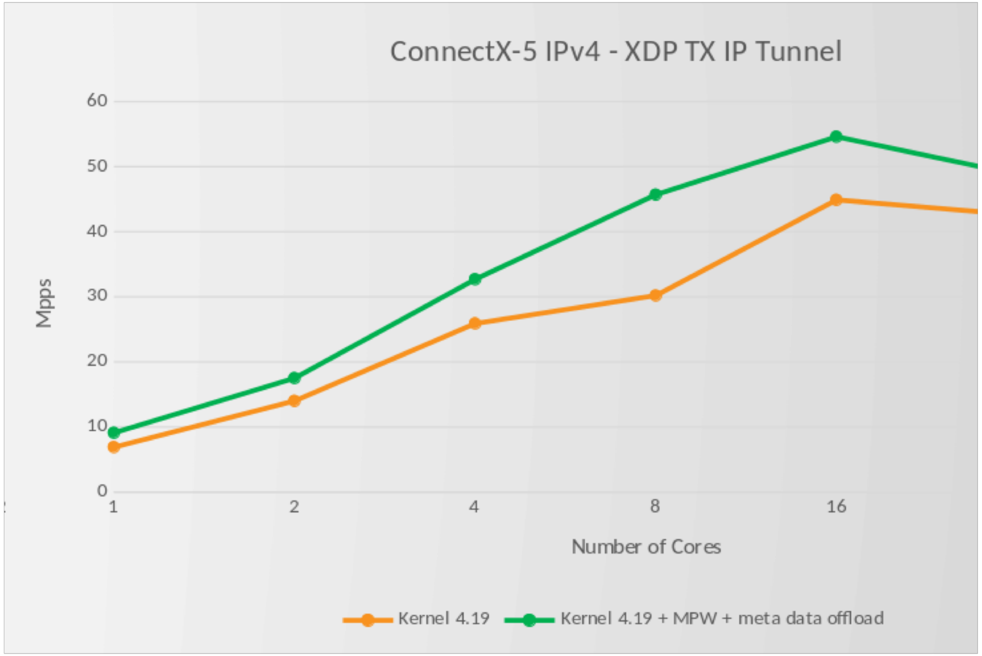


PPS without any Hints

% Improvement in PPS with inline HW Hints

%Change in PPS with SW (driver) generated hints

xdp_tx_ip_tunnel with HW Flow Mark



- Modified xdp_tx_iptunnel kernel sample
- Need an extra map flow2tnl similar to vip2tnl
- Setup a TC rule to mark packets with the well-known VIP (dst ip protocol and ds port) with a unique flow mark
- XDP Rx Meta data includes a flow_mark to fetch the tunnel from flow2tnl map

* Saeed Mahameed (Mellanox)

XDP and Rx metadata Requirements

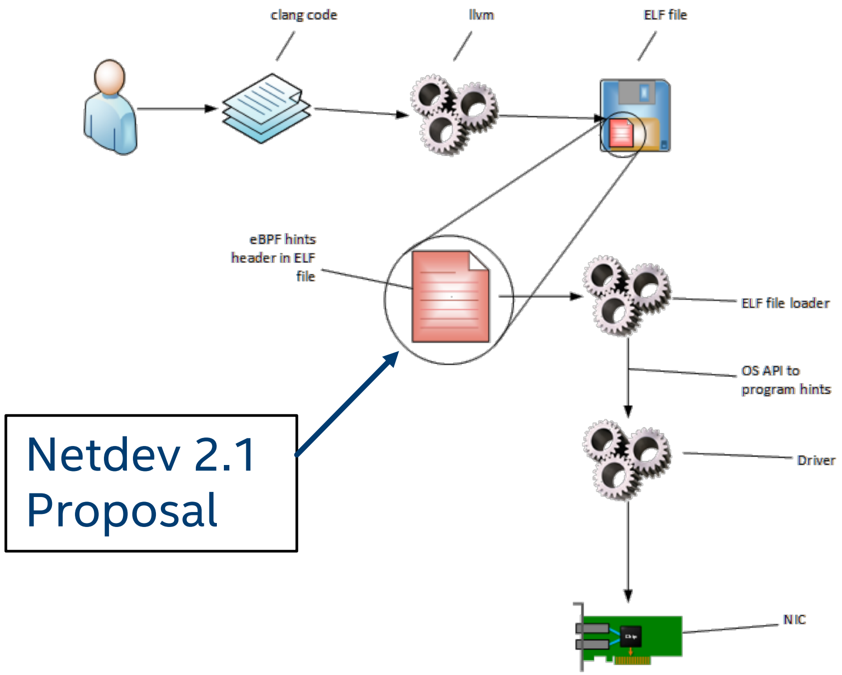
XDP program to Rx metadata type selections:

- Legacy NICs: Fixed vendor specific meta data structures provided as Rx descriptors or completions – Intel 82599(ixgbe), 7xx Series (i40e)
- Programmable NICs: Flexible Rx descriptors allows customization of Rx meta data based on use-cases – Intel E800 Series (ice)

Association of Rx meta data type to Rx Queues:

- XDP Programs should run regardless of Rx meta-data enabling
  - Legacy Programs should run without requiring meta data
- Granularity of configuration
  - All Rx Queues - Same fixed or flexible format meta data
  - Per Rx Queue – Fixed or Flexible metadata for different Rx queues for example XDP program may need different information in terms of Rx meta-data v/s AF_XDP based application on a given Rx queue may need different information

XDP meta data programming model



- Need mechanism to allow meta data types or Generic type information exchange between SW driver and XDP programs
- Supported XDP meta data configured at XDP program at load time or either at compile time

XDP meta data programming model – Solution Options

Option #1 (Fields Offset Array)

Well known XDP meta data types, defined by the kernel

A program can request any subset of well-known meta data fields from driver

Offset array
- The driver will fill meta data buffer with a pre-defined order according to the requested meta data fields (ascending order by the field enum)

- The user program will access the specific field via the pre-defined (calculated offset array)

```
flow_mark = xdp->data_meta + offset_array[XDP_META_FLOW_MARK];
```

Option #2 (BTF)

- BTF support added in 4.15+ by Facebook to provide eBPF program and maps meta data description.

2(a)

- Extend that to provide NIC meta data programming to describe meta data formats with the ndo_bfp() callback of the driver to determine if the HW can offload/provide such a meta data or not

2(b)

- Optionally Driver + firmware keep layout of the metadata in BTF format; that a user can query the driver and generate normal C header file based on BTF in the given NIC
- During sys_bpf(prog_load) the kernel checks (via supplied BTF)
- Every NIC can have their own layout of metadata and its own meaning of the fields, Standardize at least a few common fields like hash

*Inputs from Saeed Mahameed (Mellanox)

XDP meta data programming model – Pros v/s Cons of Option #2 (BTF) compared to Options #1(Fields Offset Array)

Pros

- Allows vendor defined or specific offloads to be enabled without requiring kernel support
- Meta data layout is well known to the BPF program at load time and doesn't need to use offsets at run-time

Cons

- XDP program has to be compile/recompiled with the correct meta data type for given SW+FW+HW
- Standardizing some fields is up to naming conventions of fields between different NIC vendors and overlap of these fields across vendors may create issues

*Input from Saeed Mahameed (Mellanox)

XDP Acceleration using NIC HW: Current Status

- Rx meta data WIP/RFC level patches:
  - Intel (WIP):
    - https://git.kernel.org/pub/scm/linux/kernel/git/jkirsher/next-queue.git/commit/?h=XDP-hints-EXPERIMENTAL
  - Mellanox:
    - [RFC bpf-next 0/6] XDP RX device meta data acceleration (WIP) https://www.spinics.net/lists/netdev/msg509814.html
    - [RFC bpf-next 2/6] net: xdp: RX meta data infrastructure https://www.spinics.net/lists/netdev/msg509820.html
    - https://git.kernel.org/pub/scm/linux/kernel/git/saeed/linux.git/commit/?h=topic/xdp_metadata&id=5f2908515bf64d72684b2bf902acb1a8d9af2d44
  - Alexei and Daniel proposal in netdev mailing list
    - https://www.spinics.net/lists/netdev/msg509820.html

XDP Acceleration using NIC HW: Next Steps

- Community need to agree on the approach on Rx meta data programming model to provide flexibility for a user across various use-cases and applications
- Chaining, Meta data placement in the xdp buffer
  - Chaining can be easily achieved by calling bpf_xdp_adjust_meta helper from the chained programs
  - Having the meta data fields sitting exactly before the actual packet buffer (xdp→data) is ok, BUT !
  - When bpf_xdp_adjust_head is required (header rewrite), and meta data buffer is filled, memmove(meta_data) will be required (performance hit)
    - Invalidate meta data once consumed, this will break chaining
    - Place meta data starting at xdp_buff.data_hard_start, complicated

*Input from Saeed Mahameed (Mellanox)

XDP Acceleration using NIC HW: Next Steps

- Tx metadata and processing hints
  - Same as Rx need way to configure/consume Tx meta data from applications to HW via SW drivers.
  - Provide hints to take advantage of HW offloads/accelerations like checksums, packet processing/forwarding, QoS, etc.

- Programming Rules in NIC HW to accelerate flow look-ups and actions:
  - Advantage of taking actions prior to Rx in software (e.g. drop or forwarding to a Rx queue)
  - Currently tc u32/flower or ethtool based model for enabling HW offloads and match-action rules. Programming model not suitable for XDP.
  - Not all NICs have eBPF map-table like semantics

Questions?

Backup

Performance improvements

- Internal testing yielded promising results
- Test setup:

  Target: Intel Xeon E5-2697v2 (Ivy Bridge)

  Kernel: 4.14.0-rc1+ (net-next)

  Network device: XXV710, 25GbE NIC, driver version 2.1.14-k

  Configuration: Single Rx queue, pinned interrupt

  XDP3: Zero packet parsing (best case scenario)

  XDP_HINTS: Uses ptype provided by driver, no packet parsing

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

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

intel®