redhat.

XDP Infrastructure development

Bold new ideas... that might never happen!

Jesper Dangaard Brouer,
Principal Engineer, Red Hat Inc.

NetConf 2017 part-2
South Korea, Seoul, November 2017

Background for these slides

Presented at “closed” invite-only conference NetConf

Presentation is primarily XDP progress, missing features and issues

- As a ground for discussion
- Discussing bold new ideas
  - … that might NEVER be implemented

This presentation is mostly relevant to:

- Infrastructure developers of XDP

What is XDP really

New layer in the kernel network stack

Basically: New layer in the kernel network stack

- Before allocating the SKB
- Means: Competing at the same “layer” as DPDK / netmap

Together with other eBPF hooks (like TC), opens for

- User programmable networking
  - Powerful flexibility offered to userspace

Progress

Follow Up since NetConf in Montreal April 2017
See previous presentation: Here

Driver: `ixgbe+i40e`

- killed: 1-page per packet restriction
- Tie us into refcnt based page model
- Be careful this doesn't kill new memory model for networking

New XDP_REDIRECT action

- Limited driver support :-(
- Innovative part: Redirect using maps - helper: bpf_redirect_map()
- Indirectly got RX bulking, via redirect maps

Remaining issues

The headaches not cured yet

Currently avoiding memory layer

- XDP_{DROP,TX} avoid calling memory layer
  - Driver contained, free to perform local page recycle tricks

XDP_REDIRECT cannot avoid memory layer

- Remote driver need to free/return page at DMA TX completion
- Currently based on page_frag_free()/put_page() refcnt
  - Recycling only works for ixgbe 2-pkts per page trick
  - Easily demonstrate ixgbe recycle size it too small
- Danger: Destroy opportunity for new memory model for RX-rings
  - Need fixed/known page-return point, not racing on refcnt
- Edward Cree: Suggested driver NDO for returning (XDP) pages

XDP_REDIRECT what got implemented?

Related to ifindex vs port-table discussions in Montreal

XDP redirect with maps

- Is the port-table idea from Montreal

XDP_REDIRECT via direct ifindex did happen

- Concerns addressed via
  - Egress ifindex also need to have loaded an XDP program
  - Monitor activity via tracepoints
- Using ifindex is significantly slower than using redirect maps
  - Non-map ifindex 8 Mpps -> devmap 13 Mpps
  - Devmap faster due bulking via delayed tailptr write

XDP_REDIRECT via maps

Why is XDP_REDIRECT via maps innovative?

The last XDP driver action return code (hopefully?)

- New types of redirect can be introduced without driver changes

Can be used for dynamic adaptive RX bulking

- Method of adding bulking without introducing additional latency
- Bulk only frames available in driver NAPI poll loop
- Via (driver) flush operation after napi_poll

Could be used for priority queuing

- Between frames available during NAPI poll

New map types for redirect

What is currently implemented?

Map types for redirect:

- devmap - BPF_MAP_TYPE_DEVMAP
  - Redirect to net_devices, require new driver NDO
  - Bulk effect via delaying HW tail/doorbell (like xmit_more)
- cpumap - BPF_MAP_TYPE_CPUMAP
  - Redirect raw XDP frames to remote CPUs, that alloc SKBs
  - Much more on next slide...

Next slides
XDP_REDIRECT issues

Common mistakes and issues with XDP_REDIRECT

XDP_REDIRECT pitfalls

Common mistakes and issues with XDP_REDIRECT

XDP prog return XDP_REDIRECT

- BUT have no knowledge if packet is dropped
- Sample xdp_redirect{_map} report RX-packets
  - People misguide think this equal forward TX pkts
  - Default setting will RX=13Mpps TX=6.3Mpps
    - Partly fixed via ixgbe adaptive TX DMA cleanup interval

Knowledge of drops via tracepoints

- Drop also occur due to misconfig
- (Currently) different ERRNO return code used to distinguish
- Tracepoint cost ~25ns, which affect XDP performance (split into _err)

XDP_REDIRECT API issues

API issues with

```
int ndo_xdp_xmit(struct net_device *dev, struct xdp_buff *xdp)
```

Issue: ndo_xdp_xmit queue single xdp_buff frame

- Driver must provide/alloc dedicated XDP TX queues per CPU
  - AFAIK holding back driver adoption
- Simple solution: Bulk enqueue (like cpumap)
  - Relevant xmit to VMs (queue almost empty case, cache-bounce)

Issue: no page return method or handle

- De Facto enforced refcnt based model for page return
- Info/ref to RX-device is lost, thus no later return API possible

Next slides
Describe CPUMAP current state

How the merged code works!

XDP_REDIRECT + cpumap

What is cpumap redirect?

Basic cpumap properties

- Enables redirection XDP frames to remote CPUs
- Moved SKB allocation outside driver (could help simplify drivers)

Scalability and isolation mechanism

- Allows isolating/decouple driver XDP layer from network stack
  - Don't delay XDP by deep call into network stack
- Enables DDoS protection on end-hosts (that run services)
  - XDP fast-enough to avoid packet drops happen in HW NICs

Cpumap redirect CPU scaling

Tricky part getting cross CPU delivery fast-enough

Cpumap architecture: Every slot in array-map: dest-CPU

- MPSC (Multi Producer Single Consumer) model: per dest-CPU
  - Multiple RX-queue CPUs can enqueue to single dest-CPU
- Fast per CPU enqueue store (for now) 8 packets
  - Amortized enqueue cost to shared ptr_ring queue via bulk-enq
- Lockless dequeue, via pinning kthread CPU and disallow ptr_ring resize

Important properties from main shared queue ptr_ring (cyclic array based)

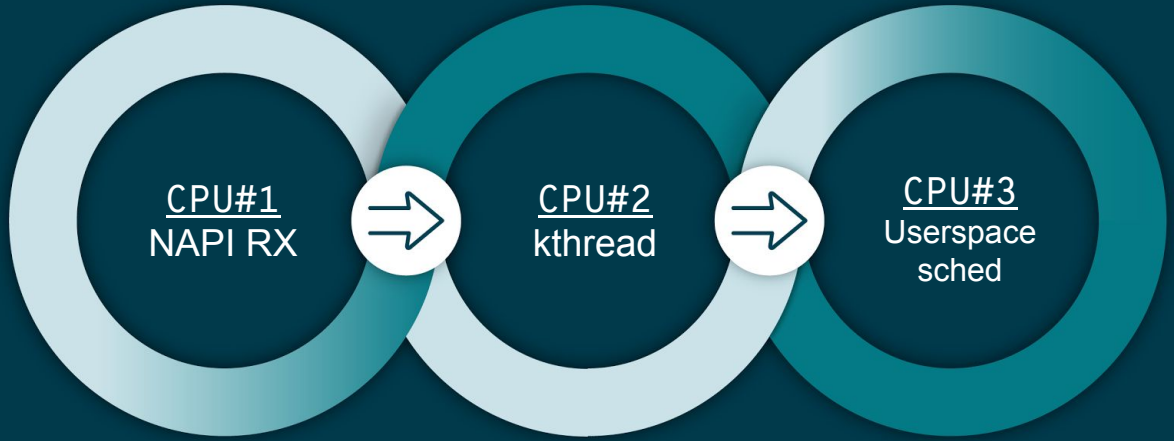
- Enqueue+dequeue don't share cache-line for synchronization
  - Synchronization happen based on elements
  - In queue almost full case, avoid cache-line bouncing
  - In queue almost empty case, reduce cache-line bouncing via bulk-enq

CPU scheduling via cpumap

Queuing and scheduling in cpumap

Hint: Same CPU sched possible

- But adjust /proc/sys/kernel/sched_wakeup_granularity_ns



Next slides
Benchmark results for cpumap

Related to page refcnt and recycle tricks

Sample/bpf xdp_redirect_cpu

Program used for benchmarking XDP cpumap, while developing

The program sample/bpf: `xdp_redirect_cpu`

- Have several XDP progs to choose between via --prog
  - Prog_num 0: Redir 1 CPU - no-touch data
  - Prog_num 1: Redir 1 CPU - touch data
  - Prog_num 2: Round-Robin between avail CPUs
  - Prog_num 3: Separate in proto UDP/TCP/ICMP , need 3 CPUs
  - Prog_num 4: Like prog3, but drop UDP dest port 9 in XDP-RX CPU
- CPUs are added via --cpu (specify multiple times, depend on prog usage)

Simply redirect from CPU-1 to CPU-2

Generator ./pktgen_sample03_burst_single_flow.sh -t 1

- Sending single UDP flow with 7.1Mpps
- Packets dropped due to "UdpNoPorts" listener

```
# ./xdp_redirect_cpu --dev ixgbe1 --prog 1 --cpu 2

Running XDP/eBPF prog_num:1
XDP-cpumap      CPU:to  pps            drop-pps    extra-info
XDP-RX          1       7,139,086      0           0
XDP-RX          total   7,139,086      0
cpumap-enqueue    1:2   7,138,979      3,123,418   8.00        bulk-average
cpumap-enqueue  sum:2   7,138,979      3,123,418   8.00        bulk-average
cpumap_kthread  2       4,015,615      0           101         sched
cpumap_kthread  total   4,015,615      0           101         sched-sum
redirect_err    total   0              0
xdp_exception   total   0              0
```

Impressive 4Mpps getting forwarded to remote CPU

- SKB alloc, netstack, kfree_skb
- Refcnt cost on CPU-2: 9.01% page_frag_free()

Increase load - same test

Same test: Simple redirect from CPU-1 to CPU-2

- Generator sending 14.88Mpps

```
Running XDP/eBPF prog_num:1
XDP-cpumap      CPU:to  pps            drop-pps    extra-info
XDP-RX          1       10,312,899     0           0
XDP-RX          total   10,312,899     0
cpumap-enqueue    1:2   10,312,890     7,283,321   8.00       bulk-average
cpumap-enqueue  sum:2   10,312,890     7,283,321   8.00       bulk-average
cpumap_kthread  2       3,029,580      0           9          sched
cpumap_kthread  total   3,029,580      0           9          sched-sum
redirect_err    total   0              0
xdp_exception   total   0              0
```

What happened? - ixgbe page recycle trick fails!

- CPU redirect drop from 4Mpps to 3Mpps
- XDP input limited to 10.3Mpps

Ethtool stats counter: 2,229,493 <= alloc_rx_page /sec (x2 for PPS)

- Perf show free_one_page() stall on page-alloc spinlock

Same test - increase page-recycles size

Need more pages to recycle

- Ixgbe recycle tied to RX-ring size, increase from 512 to 1024
- ethtool -G ixgbe1 rx 1024 tx 1024

```
Running XDP/eBPF prog_num:1
XDP-cpumap      CPU:to  pps            drop-pps    extra-info
XDP-RX          1       13,510,564     0           0
XDP-RX          total   13,510,564     0
cpumap-enqueue    1:2   13,510,564     9,490,196   8.00       bulk-average
cpumap-enqueue  sum:2   13,510,564     9,490,196   8.00       bulk-average
cpumap_kthread  2       4,020,367      0           8          sched
cpumap_kthread  total   4,020,367      0           8          sched-sum
```

Solved problem: again 4 Mpps redirect to CPU-2

- 13.5Mpps limit might be related to HW limit in NIC (MPC HW counter)

Start userspace UDP consumer

Generator pktgen 14.88 Mpps

- UDP sink pinned on CPU 4
- Result: Userspace delivery 2,545,429 pps

```
Running XDP/eBPF prog_num:1
XDP-cpumap      CPU:to  pps            drop-pps    extra-info
XDP-RX          1       10,269,035     0           0
XDP-RX          total   10,269,035     0
cpumap-enqueue    1:2   10,269,023     6,302,826   8.00       bulk-average
cpumap-enqueue  sum:2   10,269,023     6,302,826   8.00       bulk-average
cpumap_kthread  2       3,966,197      0           9          sched
cpumap_kthread  total   3,966,197      0           9          sched-sum
redirect_err    total   0              0
xdp_exception   total   0              0
```

Picture change: XDP RX again reduced to 10 Mpps

- Again limited by page-allocator
- ethtool stats show 1,894,833 <= alloc_rx_page /sec
- Even-though RX-ring size is 1024
- Cpumap_kthread not affected, because not touching page refcnt

Same UDP consumer - larger recycle cache

Increase RX-ring queue to 2048 to increase page recycle trick

- Result: Userspace delivery 3,321,521 pps
- Before: 2,545,429 pps
- Without XDP, udp_sink saw 3,026,201 pps (unconnected UDP 2,755,547 pps)

```
Running XDP/eBPF prog_num:1
XDP-cpumap      CPU:to  pps            drop-pps    extra-info
XDP-RX          1       13,481,008     0           0
XDP-RX          total   13,481,008     0
cpumap-enqueue    1:2   13,481,008     9,530,719   8.00       bulk-average
cpumap-enqueue  sum:2   13,481,008     9,530,719   8.00       bulk-average
cpumap_kthread  2       3,950,296      0           9          sched
cpumap_kthread  total   3,950,296      0           9          sched-sum
redirect_err    total   0              0
```

Performance improvement:

- 3,321,521 - 2,545,429 = +776 Kpps
- (1/3321521 - 1/2545429)*10^9 = -91.79 ns saved
- Cross CPU page free/alloc is expensive

Next slides
Missing features for XDP_REDIRECT

Not too many crazy ideas, they come later

XDP_REDIRECT for Generic-XDP

How close can/should we bring Generic-XDP to Native-XDP...

Current state:

- Generic-XDP redirect via devmap works (without map flush)
  - BUT no bulking occurs, estimate 20-30% gain adding bulking!
  - Cannot intermix with Native-XDP
- Generic-XDP redirect via cpumap disabled

Generic-XDP bulking needed by both devmap and cpumap

- Simply introduce flush point in net_rx_action (like RPS have) ?

Generic-XDP for cpumap challenges

- Require extending/changing queue structures, to support SKBs
- Will lose effect of remote-SKB allocation
  - Option: could free, and allow new SKB on remote CPU(?)

Next slides
Missing features for cpumap

More crazy ideas...

Cpumap packet structure xdp_pkt

On enqueue: store info in packet data headroom

- Convert xdp_buff into struct xdp_pkt
- For now, cpumap internal data structure, consider generalizing

XDP structure layout

```
struct xdp_buff {
        void *data;
        void *data_end;
        void *data_meta;
        void *data_hard_start;
};
```

Packet structure layout

```
struct xdp_pkt {
        void *data;
        u16 len;
        u16 headroom;
        u16 metasize;
        struct net_device *dev_rx;
};
```

Cpumap - missing descriptor info

Getting access to info in HW descriptor

Missing info: On (remote) SKB creation: **RX-hash** + **HW-csum**

XDP prog on RX need to see (+ change) RX-hash

- Needed for (cpumap) redirect decision
- Modifying RX-hash (could be) used for GRO steering

How to export/get this info??? (in vendor agnostic way)

Crazy idea:

1. On enqueue, run BPF prog, somehow read info
   - Populate new xdp_pkt fields like layer4_offset, rx_hash, csum
2. OR on dequeue, run BPF prog, that read info via data_meta
   - Update new (on stack) struct, transfer to SKB (if marked avail)

Cpumap missing GRO integration

GRO integration fairly simple: BUT lets do something better!

Multi level partial packet sorting and priority queuing

- Principal: intermixed packet flows become less-intermixed

Levels:

1. First level sorting, already happen on enqueue selecting dest-CPU
2. Extend enqueue, with 8 pkts * 8 buckets (percpu)
    - Select bucket based on RX-hash,
    - or new helper xdp_redirect_cpu(..., prio, flag_flush)
        i. Crazy idea: Bucket zero could mean high prio, queue immediately
3. Dequeue watch when RX-hash changes, then build batch for GRO

Cpumap dynamic load-balancing

Problem statement, why this is difficult...

Advanced use-case: on demand activate more CPUs

- When safe to switch a flow to a new CPU?
  - When no packets are in-flight

cpumap pushed responsibility to BPF programmer

- Currently: in-flight detecting only works on dest-CPU level
  - XDP + tracepoints + map-counter, deduct queue is empty
- Flow level in-flight packet detection not possible
  - Proposal next slide...

Cpumap dynamic load-balancing

Features needed to support flow-level OoO-safety

Extensions to support: flow level OoO-safety require

- Add two BPF progs: to enqueue and dequeue (attached to cpumap)

Cooperating BPF programs, that see packet content/RX-hash

1. Normal XDP-RX, increment **enq**-flow-counter
2. Enqueue BPF prog, detect drops then decrement **enq**-flow-counter
3. Dequeue BPF prog, increment **deq**-flow-counter

- Packets in-flight = enq-flow-counter - deq-flow-counter

Crazy ideas: Use dropped packets

Make something useful out of packets about to be dropped...

Packet drops on enqueue happens when consumer is too slow

- Drop indicate producer is sending too fast
- or consumer CPU is overloaded

Use dropped packet for something

- Allow (enqueue) BPF prog to take new decision?

If protocol is flow-control or congestion aware

- Priority queue packet-drop indication to app/socket

Would allow implementing: https://youtu.be/BO0QhaxBRr0

- Paper: "Re-architecting datacenter networks and stacks for low latency and high performance"

redhat.

End slide

… well sort of, lots of benchmarks as extra slides


plus.google.com/+JesperDangaardBrouer


facebook.com/brouer


linkedin.com/in/brouer


twitter.com/JesperBrouer


youtube.com/channel/UCSyplUCgtl42z63soRMONng

Thanks to all contributors

XDP + BPF combined effort of many people

- Alexei Starovoitov
- Daniel Borkmann
- Brenden Blanco
- Tom Herbert
- John Fastabend
- Martin KaFai Lau
- Jakub Kicinski
- Jason Wang
- Andy Gospodarek
- Thomas Graf
- Edward Cree
- Michael Chan (bnxt_en)
- Saeed Mahameed (mlx5)
- Tariq Toukan (mlx4)
- Björn Töpel (i40e)
- Yuval Mintz (qede)
- Sunil Goutham (thunderx)
- Jason Wang (VM)
- Michael S. Tsirkin (ptr_ring)

Next slides
Benchmark results for cpumap

Related to scaling cpumap

Scaling cpumap, 1 RX queue + 4 dest-CPUs

Generator 14.88 Mpps, RX-ring size 2048, 1 RX-queue (ethtool -L ixgbe1 combined 1)

- Prog_num 2: Round-Robin between available CPUs

```
# ./xdp_redirect_cpu --dev ixgbe1 --prog 2 --cpu 1 --cpu 2 --cpu 3 --cpu 4
Running XDP/eBPF prog_num:2
XDP-cpumap      CPU:to  pps            drop-pps    extra-info
XDP-RX          0       10,042,078     0           0
XDP-RX          total   10,042,078     0
cpumap-enqueue    0:1   2,510,519      39          8.00       bulk-average
cpumap-enqueue  sum:1   2,510,519      39          8.00       bulk-average
cpumap-enqueue    0:2   2,510,521      0           8.00       bulk-average
cpumap-enqueue  sum:2   2,510,521      0           8.00       bulk-average
cpumap-enqueue    0:3   2,510,518      179         8.00       bulk-average
cpumap-enqueue  sum:3   2,510,518      179         8.00       bulk-average
cpumap-enqueue    0:4   2,510,520      115         8.00       bulk-average
cpumap-enqueue  sum:4   2,510,520      115         8.00       bulk-average
cpumap_kthread  1       2,510,480      0           80,880     sched
cpumap_kthread  2       2,510,520      0           85,999     sched
cpumap_kthread  3       2,510,344      0           95,320     sched
cpumap_kthread  4       2,510,460      0           125,950    sched
cpumap_kthread  total   10,041,807     0           388,150    sched-sum
```

Analysis: cpumap_kthread have idle time: more overhead in wake_up_process

- Few page allocs: 642 <= alloc_rx_page /sec

Interesting: kthreads sched=queue-sometimes-empty + can bulk enqueue at same time

Scaling cpumap, 2 RX queue + 4 dest-CPUs

Generator 14.88 Mpps, RX-ring size 2048, 2 RX-queue (ethtool -L ixgbe1 combined 2)

```
Running XDP/eBPF prog_num:2
XDP-cpumap      CPU:to  pps            drop-pps    extra-info
XDP-RX          0       9,135,210      0           0
XDP-RX          1       5,479,637      0           0
XDP-RX          total   14,614,847     0
cpumap-enqueue    0:2   2,283,807      401         7.70       bulk-average
cpumap-enqueue    1:2   1,369,906      320         6.80       bulk-average
cpumap-enqueue  sum:2   3,653,713      722         7.34       bulk-average
cpumap-enqueue    0:3   2,283,809      736         7.70       bulk-average
cpumap-enqueue    1:3   1,369,909      472         6.80       bulk-average
cpumap-enqueue  sum:3   3,653,718      1,209       7.34       bulk-average
cpumap-enqueue    0:4   2,283,811      327         7.70       bulk-average
cpumap-enqueue    1:4   1,369,906      195         6.80       bulk-average
cpumap-enqueue  sum:4   3,653,718      522         7.34       bulk-average
cpumap-enqueue    0:5   2,283,809      305         7.70       bulk-average
cpumap-enqueue    1:5   1,369,909      199         6.80       bulk-average
cpumap-enqueue  sum:5   3,653,719      505         7.34       bulk-average
cpumap_kthread  2       3,652,994      0           4,885      sched
cpumap_kthread  3       3,652,507      0           5,225      sched
cpumap_kthread  4       3,653,191      0           4,887      sched
cpumap_kthread  5       3,653,205      0           4,786      sched
cpumap_kthread  total   14,611,899     0           19,785     sched-sum
```

Basically handling wirespeed 10G, and system have idle cycles

- Only need 2 RX queues to serve+re-distribute 10G/14.6Mpps

Scaling cpumap, 3 RX queues + 1 dest-CPU

RX-ring size 2048, 4 RX-queue but 3 used (ethtool -L ixgbe1 combined 4)

- New pktgen generator: pktgen_sample05_flow_per_thread.sh -t 4
- Hash in NIC result in uneven distribution of flows, only 3 RX queues used

```
Running XDP/eBPF prog_num:1
XDP-cpumap      CPU:to  pps            drop-pps    extra-info
XDP-RX          0       3,650,261      0           0
XDP-RX          2       7,301,400      0           0
XDP-RX          3       3,650,512      0           0
XDP-RX          total   14,602,174     0
cpumap-enqueue    0:5   3,650,267      2,693,478   7.77        bulk-average
cpumap-enqueue    2:5   7,301,415      5,233,577   7.89        bulk-average
cpumap-enqueue    3:5   3,650,521      2,731,520   7.77        bulk-average
cpumap-enqueue  sum:5   14,602,204     10,658,576  7.83        bulk-average
cpumap_kthread  5       3,943,628      0           1           sched
cpumap_kthread  total   3,943,628      0           1           sched-sum
```

XDP-RX CPUs have many idle cycles

- Good results: cpumap_kthread handle approx 4 Mpps (as in other tests)
- XDP-RX total 14.6Mpps basically wirespeed

Scaling cpumap, 4 RX queues + 1 dest-CPU

RX-ring size 2048, 4 RX-queue (ethtool -L ixgbe1 combined 4)

- Generator: pktgen_sample05_flow_per_thread.sh -t 8
- Hash in NIC result in uneven distribution of flows (now using 4 RX-queues)

```
Running XDP/eBPF prog_num:1
XDP-cpumap      CPU:to  pps            drop-pps    extra-info
XDP-RX          0       3,584,590      0           0
XDP-RX          1       1,795,150      0           0
XDP-RX          2       5,383,807      0           0
XDP-RX          3       3,582,162      0           0
XDP-RX          total   14,345,711     0
cpumap-enqueue    0:5   3,584,590      2,636,202   7.77        bulk-average
cpumap-enqueue    1:5   1,795,150      1,334,663   7.71        bulk-average
cpumap-enqueue    2:5   5,383,832      3,930,381   7.79        bulk-average
cpumap-enqueue    3:5   3,582,157      2,662,313   7.78        bulk-average
cpumap-enqueue  sum:5   14,345,730     10,563,561  7.77        bulk-average
cpumap_kthread  5       3,782,177      0           4           sched
cpumap_kthread  total   3,782,177      0           4           sched-sum
```

XDP-RX CPUs have many idle cycles

- Still good results: cpumap_kthread handle approx 3.8 Mpps
- XDP-RX total 14.3Mpps

Scaling, 4 RX queues + 1 dest-CPU + udp_sink

RX-ring size 2048, 4 RX-queue (ethtool -L ixgbe1 combined 4)

- Userspace delivery
- UDP sink pinned on CPU 4: Performance: 2,650,814 pps
  - Cannot used connected UDP sockets, thus lower perf base expected 2.7Mpps

```
Running XDP/eBPF prog_num:1
XDP-cpumap      CPU:to  pps            drop-pps    extra-info
XDP-RX          0       3,585,477      0           0
XDP-RX          1       1,794,914      0           0
XDP-RX          2       5,375,309      0           0
XDP-RX          3       3,588,839      0           0
XDP-RX          total   14,344,540     0
cpumap-enqueue    0:5   3,585,473      2,924,183   7.77        bulk-average
cpumap-enqueue    1:5   1,794,895      1,471,639   7.70        bulk-average
cpumap-enqueue    2:5   5,375,309      4,367,503   7.79        bulk-average
cpumap-enqueue    3:5   3,588,846      2,929,602   7.77        bulk-average
cpumap-enqueue  sum:5   14,344,525     11,692,929  7.77        bulk-average
cpumap_kthread  5       2,651,601      0           0
cpumap_kthread  total   2,651,601      0           0
```

Notice: cpumap_kthread: limited by UDP enqueue

- top#1 - 13.31% __udp_enqueue_schedule_skb()

Next slides
Benchmark results for cpumap
DDoS protection on the end-host (running services)

DDoS protecting end-host

Generator 12.3Mpps (below wirespeed to avoid wire/HW drops)

- Worse-case: Force traffic to share same RX-ring queue
- Prog_num 4: Like prog3, but drop UDP dest port 9 on XDP-RX CPU

```
# ./xdp_redirect_cpu --dev ixgbe1 --prog 4 --cpu 1 --cpu 2 --cpu 3
Running XDP/eBPF prog_num:4
XDP-cpumap      CPU:to  pps            drop-pps    extra-info
XDP-RX          0       12,245,665     12,210,629  0
XDP-RX          total   12,245,665     12,210,629
cpumap-enqueue    0:1   35,036         0           1.00        bulk-average
cpumap-enqueue  sum:1   35,036         0           1.00        bulk-average
cpumap_kthread  1       35,036         0           35,036      sched
cpumap_kthread  total   35,036         0           35,036      sched-sum
redirect_err    total   0              0
xdp_exception   total   0              0
```

Notice: XDP-RX CPU had 44% idle cycles (while dropping 12.2Mpps)

- Netperf TCP_RR test shows 35K trans/sec, during DDoS attack
  - Normal 40K trans/sec, Limit in NIC-HW cause this
    - using separate RXq the same result
    - Reducing load to 6.4Mpps then 40K trans/sec

DDoS protecting end-host

Generator reduced to 6.2 Mpps

- Worse-case: Force traffic to share same RX-ring queue
- Prog_num 4: Like prog3, but drop UDP dest port 9 on XDP-RX CPU

```
# ./xdp_redirect_cpu --dev ixgbe1 --prog 4 --cpu 2 --cpu 3 --cpu 4
Running XDP/eBPF prog_num:4
XDP-cpumap      CPU:to  pps            drop-pps    extra-info
XDP-RX          1       6,277,030      6,236,628   0
XDP-RX          total   6,277,030      6,236,628
cpumap-enqueue    1:2   40,402         0           1.00       bulk-average
cpumap-enqueue  sum:2   40,402         0           1.00       bulk-average
cpumap_kthread  2       40,402         0           40,402     sched
cpumap_kthread  total   40,402         0           40,402     sched-sum
redirect_err    total   0              0
xdp_exception   total   0              0
```

Reducing load to 6.4Mpps

- Netperf TCP_RR now shows 40K trans/sec, during DDoS attack
- AFAIK limit in ixgbe HW handing out descriptors

Generator

- sss

```
Running XDP/eBPF prog_num:1
```

Imp

- S