

이해력

XDP For the Rest of Us

Jesper Dangaard Brouer - Principal Engineer, Red Hat

Andy Gospodarek - Principal Engineer, Broadcom

Netdev 2.2, November 8th, 2017

South Korea, Seoul

Motivation for this talk

- Follow Up on [NetDev 2.1 tutorial/talk](#)
 - Less time, focus on updates and new tools from XDP ecosystem
- Still motivated to:
 - Demystify XDP and eBPF
 - Help you understand and consume this new technology

What will you learn?

What do *you* get out of this presentation

What will you learn?

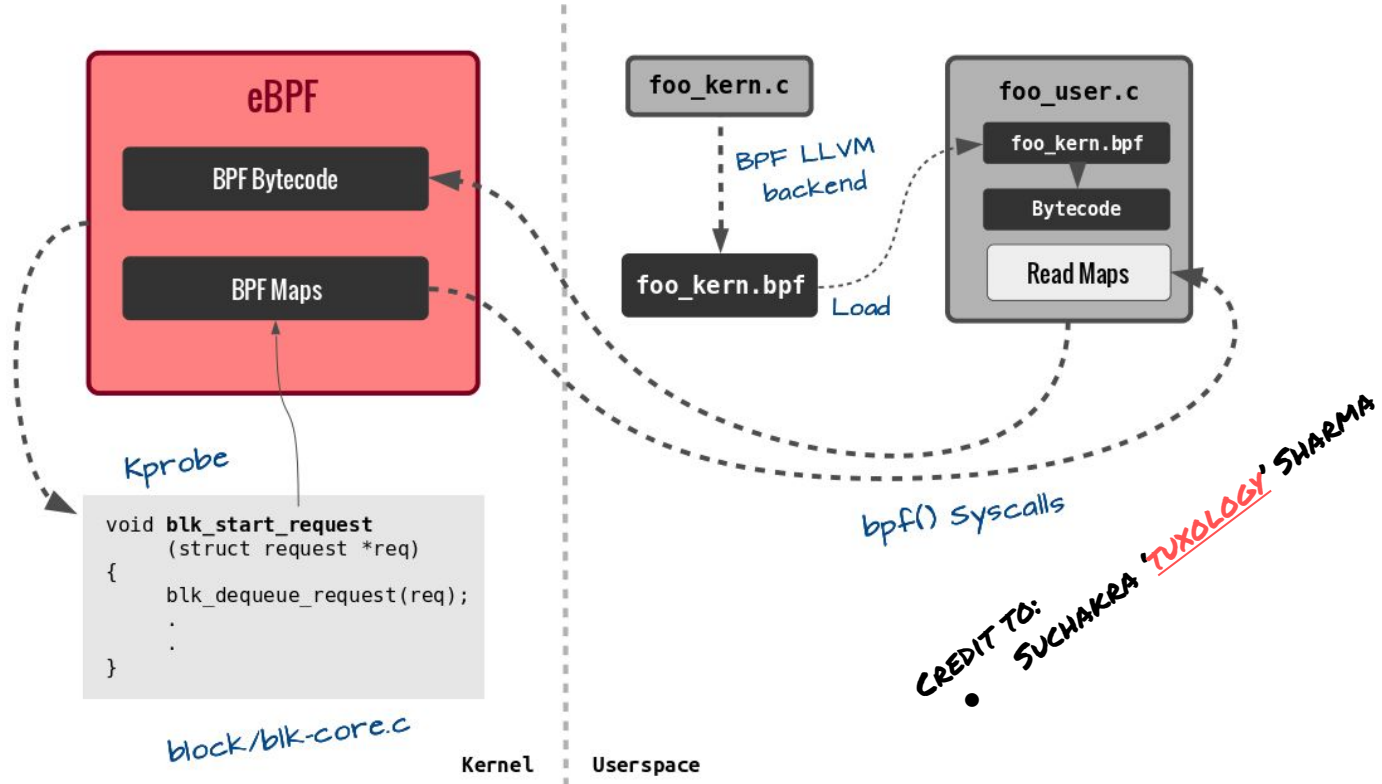
- Bring you up-to-date with the XDP ecosystem
 - highlight subset of recent changes
- We want you in the driver's seat
 - fast, user-programmable networking
- Teach you about some new tools
- Spark new ideas for XDP+BPF use-cases
 - Going beyond DDoS and (bouncing) Load-Balancer use-cases

What will you NOT learn!

- Getting Started with eBPF and XDP
 - Is covered in [Netdev 2.1 talk](#), like:
 - Compiler toolchain LLVM / clang
 - Compiling `kernel/samples/bpf`
 - Source file split `foo_kern.c + foo_user.c`
 - ELF-object containing map-definitions,
 - How handled by BPF loader code
 - Invoking appropriate BPF-syscalls

Want to understand drawing?

- [Watch Netdev 2.1 talk](#) on YouTube ;-)



The XDP technology

A new era with user-programmable networking

Framing: The XDP technology

- XDP a new, lower layer in Linux network stack
 - Programmable hook in drivers can run before allocating full SKB
 - New building block for Linux kernel networking
- Operate at same "layer" as bypass solutions (like DPDK)
 - Operate at same speeds as bypass solutions (low number of CPU instructions per packet)
 - Raw-data access to (Ethernet) frame (before SKB exists)
 - An in-kernel fast-path (XDP core in Linux kernel v4.8)
- The XDP programming language is eBPF
 - eBPF is bigger than XDP, complete compiler toolchain
 - XDP just one-hook using/invoking eBPF
- Real power comes from using more bpf-hooks combined
 - From userspace: Controlling XDP/BPF via maps

XDP + eBPF = User programmable networking

- XDP and eBPF really good combination
 - New era in user programmable networking
- Kernel side: responsible for moving packet fast
- BPF side: maximum flexibility and opt-in
 - User-programmable protocols and policies
 - Administrators can quickly implement something
 - No need to upgrade kernel
 - Only run program code needed for use-case
 - No accumulative feature bloat
- In-kernel solution
 - Maintained by the Linux kernel community
 - New XDP program deployed via atomic swap operation

XDP interface: the basics

- What can XDP do?
 - Can read and modify packet contents
 - Can push and pull headers
- XDP interface: BPF program returns an action-code
 - **XDP_DROP** – very fast drop by recycling (DDoS mitigation)
 - **XDP_PASS** – pass possibly modified packet to network stack
 - **XDP_TX** – Transmit packet back out same interface with or without packet modification
 - **XDP_ABORTED** – also drop, but indicate error condition (catch via tracepoint)
 - **XDP_REDIRECT** – Transmit out other NIC or steer via maps
- All BPF programs interact via
 - Helper function that can lookup or modify kernel state
 - Shared maps that userspace and other bpf-programs can use to track state

Designed to cooperate with network stack

- How to handle new protocol/encapsulation
 - That the kernel doesn't know yet?
 - Without upgrading the running kernel!
- On RX:
 - XDP can adjust packet headers to something kernel understand
 - E.g. steer into VLAN devices
 - XDP can add metadata to data buffer than can be used by other eBPF programs
- On TX:
 - BPF can add back (encapsulation) headers
 - BPF hooks in Traffic Control or Socket filter
 - Restore packet-data based on shared BPF-map, VLAN device or SKB marking

The XDP ecosystem

Where should you start?!?

XDP ecosystem

- Mailing lists:
 - XDP newbies join: xdp-newbies@vger.kernel.org
 - Kernel devel-side: netdev@vger.kernel.org
 - BPF devel-side: iovisor-dev@lists.iovisor.org
- Sample code available:
 - Kernel git-tree: [samples/bpf/](#)
 - Github: [prototype-kernel](#) under [samples/bpf/](#)
 - IOvisor [BCC](#) project (if you prefer Python)
- Documentation:
 - [prototype-kernel.readthedocs.io](#) - plan integrate into [kernel.org/doc](#)
 - Cilium: “[BPF and XDP Reference Guide](#)”

Recent changes of interest

Since last NetDev 2.1 (April 2017, Montreal)

- Only covering constrained subset

Recent changes: BPF introspection

- Visibility into running BPF programs
 - Kernel v4.13: [BPF ID's for loaded progs and maps](#)
 - can be accessed and dumped from userspace
- `bpftool`
 - Part of Kernel tree: [tools/bpf/bpftool/](#)
 - Allows inspection and simple modification of BPF objects
 - Easy to list all programs currently loaded
- `xdp_monitor`
 - Part of kernel tree: [samples/bpf](#)
 - BPF prog monitoring XDP via tracepoints
 - Helps debugging XDP

THANKS TO:
● MARTIN KAFAI LAU (FACEBOOK)
● JAKUB KICINSKI (NETRONOME)

Is an XDP program loaded?

```
$ ip link show

2: enpls0f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 xdp qdisc mq [...]
    link/ether 00:0a:f7:8d:ab:60 brd ff:ff:ff:ff:ff:ff
    prog/xdp id 4
```


bpftool

```
# bpftool
Usage: bpftool [OPTIONS] OBJECT { COMMAND | help }
       bpftool batch file FILE
       bpftool version

OBJECT := { prog | map }
OPTIONS := { {-j|--json} [{-p|--pretty}] }
```

bpftool

```
# bpftool map help
Usage: bpftool map show      [MAP]
      bpftool map dump      MAP
      bpftool map update    MAP key BYTES value VALUE [UPDATE_FLAGS]
      bpftool map lookup    MAP key BYTES
      bpftool map getnext   MAP [key BYTES]
      bpftool map delete    MAP key BYTES
      bpftool map pin       MAP FILE
      bpftool map help

MAP := { id MAP_ID | pinned FILE }
PROGRAM := { id PROG_ID | pinned FILE | tag PROG_TAG }
VALUE := { BYTES | MAP | PROGRAM }
UPDATE_FLAGS := { any | exist | noexist }
OPTIONS := { {-j|--json} [{-p|--pretty}] }
```

bpftool

```
# bpftool program help
Usage: bpftool prog show [PROG]
       bpftool prog dump xlated PROG [{ file FILE | opcodes }]
       bpftool prog dump jited  PROG [{ file FILE | opcodes }]
       bpftool prog pin    PROG FILE
       bpftool prog help

PROG := { id PROG_ID | pinned FILE | tag PROG_TAG }
OPTIONS := { {-j|--json} [{-p|--pretty}] }
```

Running xdp_ddos01_blacklist

```
# xdp_ddos01_blacklist --dev enp1s0f0
```

Documentation:

XDP: DDoS protection via IPv4 blacklist

This program loads the XDP eBPF program into the kernel.
Use the cmdline tool for add/removing source IPs to the blacklist
and read statistics.

```
- Attached to device:enp1s0f0 (ifindex:2)
- Export bpf-map:blacklist to file:/sys/fs/bpf/ddos_blacklist
- Export bpf-map:verdict_cnt to file:/sys/fs/bpf/ddos_blacklist_stat_verdict
- Export bpf-map:port_blacklist to file:/sys/fs/bpf/ddos_port_blacklist
- Export bpf-map:port_blacklist_drop_count_tcp to file:/sys/fs/bpf/ddos_port_blacklist_count_tcp
- Export bpf-map:port_blacklist_drop_count_udp to file:/sys/fs/bpf/ddos_port_blacklist_count_udp
blacklist_modify() IP:198.18.50.3 key:0x33212C6
blacklist_port_modify() dport:80 key:0x50
```

bpftool inspecting xdp_ddos01_blacklist

```
# bpftool prog show
4: xdp tag 575d0fd6aa6dde66
   loaded_at Oct 25/15:04 uid 0
   xlated 864B jited 566B memlock 4096B map_ids 5,6,7,8,9
# bpftool map show
5: percpu_hash flags 0x1
   key 4B value 8B max_entries 100000 memlock 14897152B
6: percpu_array flags 0x0
   key 4B value 8B max_entries 4 memlock 4096B
7: percpu_array flags 0x0
   key 4B value 4B max_entries 65536 memlock 4722688B
8: percpu_array flags 0x0
   key 4B value 8B max_entries 65536 memlock 4722688B
9: percpu_array flags 0x0
   key 4B value 8B max_entries 65536 memlock 4722688B
```

bpftool inspecting eBPF maps

```
# bpftool map dump id 5
Key:
c6 12 32 03
value (CPU 00): 00 00 00 00 00 00 00 00
value (CPU 01): 00 00 00 00 00 00 00 00
value (CPU 02): 00 00 00 00 00 00 00 00
value (CPU 03): 00 00 00 00 00 00 00 00
value (CPU 04): 00 00 00 00 00 00 00 00
value (CPU 05): 00 00 00 00 00 00 00 00
value (CPU 06): 00 00 00 00 00 00 00 00
value (CPU 07): 00 00 00 00 00 00 00 00
Found 1 element
# printf "%d.%d.%d.%d\n" 0xc6 0x12 0x32 0x03
198.18.50.3
```

bpftool now with JSON output

```
# bpftool map --json dump id 5
[{"key":["0xc6","0x12","0x32","0x03"],"values":[{"cpu":0,"value":["0x00","0x00","0x00","0x00","0x00","0x00","0x00","0x00","0x00","0x00"]}, {"cpu":1,"value":["0x00","0x00","0x00","0x00","0x00","0x00","0x00","0x00","0x00"]}, {"cpu":2,"value":["0x00","0x00","0x00","0x00","0x00","0x00","0x00","0x00","0x00"]}, {"cpu":3,"value":["0x00","0x00","0x00","0x00","0x00","0x00","0x00","0x00"]}, {"cpu":4,"value":["0x00","0x00","0x00","0x00","0x00","0x00","0x00","0x00"]}, {"cpu":5,"value":["0x00","0x00","0x00","0x00","0x00","0x00","0x00","0x00"]}, {"cpu":6,"value":["0x00","0x00","0x00","0x00","0x00","0x00","0x00"]}, {"cpu":7,"value":["0x00","0x00","0x00","0x00","0x00","0x00","0x00","0x00"]}]}
```

THANKS TO:
• QUENTIN MONNET (NETRONOME)

bpftool now with JSON output (cont)

```
# bpftool map --json --pretty dump id 5
[{"key": ["0xc6", "0x12", "0x32", "0x03"],
  "values": [{"cpu": 0,
    "value": ["0x00", "0x00", "0x00", "0x00", "0x00", "0x00", "0x00", "0x00"]
  }, {"cpu": 1,
    "value": ["0x00", "0x00", "0x00", "0x00", "0x00", "0x00", "0x00", "0x00"]
  }, {"cpu": 2,
    "value": ["0x00", "0x00", "0x00", "0x00", "0x00", "0x00", "0x00", "0x00"]
  }
  ]
  [...]
}
```


Load another XDP program on another interface

```
# bpftool prog show
4: xdp tag 575d0fd6aa6dde66
   loaded_at Oct 25/15:04 uid 0
   xlated 864B jited 566B memlock 4096B map_ids 5,6,7,8,9
8: xdp tag 0381911915bc8d7f
   loaded_at Oct 25/23:07 uid 0
   xlated 496B jited 339B memlock 4096B map_ids 14
# bpftool map show id 14
14: percpu_array name rxcnt flags 0x0
   key 4B value 8B max_entries 256 memlock 20480B
```

xdp_monitor as a debugging tool

```
# ./xdp_monitor --stats
ACTION          result          pps             pps-human-readable  measure-period
XDP_REDIRECT    Success         31533           31,533              2.000119
XDP_REDIRECT    Error           0               0                   2.000121
XDP_ABORTED     Exception       13274271        13,274,271          2.000121
```

Above results from XDP_REDIRECT+ cpumap:

- Misconfig resulted in all UDP (pktgen) traffic drop via XDP_ABORTED

Cmd: # xdp_redirect_cpu --dev ixgbe1 --prog 3 --cpu 2

- TCP request-response traffic flowing to another CPU (31Kpps)

Cmd: # netperf -H 172.16.0.2 -t TCP_RR

xdp_redirect_cpu + cpumap output

Program running while xdp_monitor was inspecting system

```
# ./xdp_redirect_cpu --dev ixgbe1 --prog 3 --cpu 2

Running XDP/eBPF prog_num:3
XDP-cpumap      CPU:to  pps          drop-pps      extra-info
XDP-RX          0       13,273,868    0             13,273,868   cpu-dest/err
XDP-RX          4       31,530        0             0            cpu-dest/err
XDP-RX          total   13,305,399    0
cpumap-enqueue  4:2    31,530        0             1.00         bulk-average
cpumap-enqueue  sum:2   31,530        0             1.00         bulk-average
cpumap_kthread  2       31,530        0             31,530       sched
cpumap_kthread  total   31,530        0             31,530       sched-sum
redirect_err    total   0             0
xdp_exception   total   0             13,273,869
```

Great tools, but “patches accepted”

- `bpftool`
 - Decode/pretty-print more values stored in maps
 - Inspect BPF progs before loaded (compare `tag` to running programs)
 - Accumulate results in percpu maps (examples use them as counters)
- `xdp_monitor`
 - Use as a framework/example for more application development
 - JSON output
 - `--oneshot` support to gather current stats rather than running interactively

Recent changes: XDP metadata for BPF

- XDP metadata: generic and flexible
 - Communication channel between XDP-hook and TC-hooks
 - XDP dynamic reserve **part of packet headroom**
 - Max 32-Bytes avail, BPF prog choose meaning
 - Later BPF hooks (e.g. TC) load prog that knows meaning
 - Can access, extract and populate SKB members,
 - e.g. `skb->mark`
- Provide way for XDP to cooperate with network stack
 - By saving info in `xdp_buff->data_meta` area

Recent changes: **XDP_REDIRECT**

- New XDP return code `XDP_REDIRECT`
 - Innovative part: Redirect using maps (use `bpf_redirect_map()`)
- Redirect via maps:
 - Introduces RX bulking, via flush operation after `napi_poll`
 - Dynamic adaptive bulking
 - Method of adding bulking without introducing additional latency
 - Bulk only frames available in driver NAPI poll loop
- New map types for redirect
 - `devmap` - `BPF_MAP_TYPE_DEVMAP`
 - Bulk effect via delaying HW tail/doorbell (like `xmit_more`)
 - `cpumap` - `BPF_MAP_TYPE_CPUMAP`
 - Bulk 8 frame to remote CPU, amortize cross CPU cost
 - Provide CPU separation at XDP “layer”

Use-cases

Even new use-cases you did not realize were possible...

Well known use-cases

- DDoS protection
- Load-balancing router (Facebook use-case)
- Forwarding between containers (Cilium use-case)
- Rapid prototyping of protocol extensions

Fix NIC and existing kernel limitations

- Handling protocols currently unknown to kernel
 - Kernel upgrade not always easy or possible
 - As described earlier XDP+BPF can help
 - BUT even harder to upgrade hardware NIC
 - NIC hardware cannot parse protocol
 - Only safe option for hardware is delivery to single RX-queue
 - Single core cannot scale to handle all traffic
- XDP_REDIRECT via cpumap helps
 - Allow redistributing load on CPUs
 - Benchmarks (ixgbe) shows it scales to 11 Mpps per RX CPU

Enable XDP offload of routing stack

- Functions like IPv4 forward could be handled by XDP
 - See proposal for XDP sample ([xdp_router_ipv4](#)) implementing IPv4 forward
- Use normal Linux tools to change Routing and Neighbor tables
 - Maintain BPF shadow maps of routing and ARP table
 - Subscribe to changes via rtnetlink updates
- Use XDP_REDIRECT to rewrite packets and forward between known destinations

The End

Are we out of time yet?

XDP Summary

- In-kernel fast-path solution
- Programmable networking inside the network stack!
- Lower maintenance and deployment cost as it is part of the Linux Kernel
- Does not take over NIC hardware and isolate it from the network stack

Thanks to

- XDP + BPF combined effort of many people
 - Alexei Starovoitov
 - Daniel Borkmann
 - Brenden Blanco
 - Tom Herbert
 - John Fastabend
 - Martin KaFai Lau
 - Jakub Kicinski
 - Michael S. Tsirkin
 - Jason Wang
 - Saeed Mahameed
 - Tariq Toukan
 - Edward Cree