Accelerating VM networking through XDP

Jason Wang
Red Hat
Agenda

- Kernel VS userspace
- Introduction to XDP
- XDP for VM
- Use cases
- Benchmark and TODO
- Q&A
Kernel Networking datapath

- **TAP**
  - A driver to transmit to or receive from userspace
  - Backend for vhost_net
- **Vhost**
  - Virtio protocol to cooperate with guest driver
- **OVS**
  - Forwarding packets between interfaces
Userspace Networking datapath

- **vhost-user**
  - For implementing vhost dataplane in another userspace process

- **OVS-Dpdk**
  - Userspace OVS
  - Datapath were accelerated through dpdk
    - Polling mode driver
    - VFIO
    - Vhost-user slave

![Diagram of Userspace Networking datapath]

- **Guest**
  - virtio-net drv
  - TX
  - RX

- **OVS-dpdk**
  - vhost_pmd

- **OVS core**

- **NIC pmd**
Because we(kernel) are slow

Packet Per Second (Mpps)

<table>
<thead>
<tr>
<th>vhost(kernel)</th>
<th>RX</th>
<th>TX</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.78</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.83</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>vhost(dpdk)</th>
<th>RX</th>
<th>TX</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13.88</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Why? Difference in datapath

<table>
<thead>
<tr>
<th></th>
<th>dpdk</th>
<th>kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Meta-data</strong></td>
<td>rte_mbuf, head of packet</td>
<td>heavyweight, skb, extra allocation</td>
</tr>
<tr>
<td><strong>Memory model</strong></td>
<td>memory pool for both TX and RX</td>
<td>vendor specific (RX), protocol specific (TX)</td>
</tr>
<tr>
<td><strong>DMA</strong></td>
<td>statically mapped</td>
<td>dynamically mapped</td>
</tr>
<tr>
<td><strong>Batching (forwarding)</strong></td>
<td>aggressive</td>
<td>almost no</td>
</tr>
<tr>
<td><strong>Busy polling</strong></td>
<td>Polling Mode Driver</td>
<td>NAPI (polling + event)</td>
</tr>
<tr>
<td><strong>VM datapath in host</strong></td>
<td>vhost pmd</td>
<td>TAP + vhost_net</td>
</tr>
<tr>
<td><strong>vhost</strong></td>
<td>bulking, prefetching, busy polling</td>
<td>no bulking, no prefetching, event based (or limited busy polling)</td>
</tr>
</tbody>
</table>
Question:

A fast and lightweight data path for kernel?
XDP = eBPF + early packet processing
**eBPF**

- **Generic, efficient, secure in-kernel (Linux) virtual machine. Programs are injected and attached in the kernel, event-based.**

- **extend BPF**
  - Evolution from classical BPF, assembly-like, interpreter
  - Effective: more registers and instructions, larger stack
  - Read or write access to context (packets for net)
  - LLVM backend
  - safety: in kernel verifier
  - JIT (Just in time)
  - Bpf() syscall for managing program
eBPF features

- Maps: key-value entries (hash, array, ...) shared between eBPF program and with the user space
- Tail calls: “long jump” form one program into another, context is preserved
- Helpers: for each type of eBPF program, a set of white-list of kernel functions that could be called:
  - Get time
  - Debug information printing
  - Lookup or update maps
  - Shrink or grow packets
  - XDP is a type of eBPF program
A fast path in kernel stack - XDP

- Short for eXpressed Data Path (4.8+)
- Allow run eBPF program at lowest point in the software stack

Use cases:
- Early dropping - DDoS
- Load balancing
- Monitoring
- Fast forwarding (networking device, CPU, userspace - AF_XDP)

Designed for:
- High performance, fast path in the kernel stack
  - Multiqueue for lockless TX, Simple memory layout, NAPI loop, Batching
- Programmability, new function without modifications for kernel
- Not a kernel bypass, work with kernel networking stack
- No need for specialized stack
**XDP context**

- **Simple**
  - Driver need prepare the XDP context and call XDP program
  - Allocated on the stack usually, no pressure for the memory allocator
  - Stored at the head of a packet
  - Packet address, metadata, and receive queue information
  - Linear, no frag, frag_list,

```c
struct xdp_buff {
    void *data;
    void *data_end;
    void *data_meta;
    void *data_hard_start;
    struct xdp_rxq_info *rxq;
};
```
XDP Actions

- **Driver behave depends on XDP program return value**
  - XDP_DROP: drop packet immediately
  - XDP_TX: transmit the packet back to the interface
  - XDP_PASS: build skb and pass the packet to network stack
  - XDP_REDIRECT: redirect packets
    - Bulking in forwarding path
XDP = eBPF program run inside driver

1. foo_kern.c -> foo_kern.bpf
2. foo_kern.bpf -> bytecode
3. bytecode -> foo_kern.bpf
4. bpf() syscall
5. netlink Set XDP

- foo_user.c
- eBPF Verifier/JIT
- Networking Stack
- skb
- NIC Driver
- XDP program
- machine code
- Maps
- Read/Write Maps
- NIC Ring
XDP support

- **eBPF**
  - Interpreter, JIT, Verifier, helpers

- **Networking device driver support**
  - Native mode, driver can work on XDP buff
    - Native XDP support before building skb
    - Simple memory model (e.g. one page per packet)
    - ndos for redirection, can transmit XDP buffs, bulking
    - Best performance

- **Networking core support**
  - Attach XDP program to a network device
  - XDP generic
    - skb level
    - Prototype developing
    - Deal with e.g. SG or GSO packets
Accelerating method I: accelerate guest networking

- **XDP support in virtio-net (4.10)**
- **Works for**
  - Kernel datapath on host
  - Userspace datapath on host
- **Use case:**
  - Typical usage for XDP:
    - DDOS, LB, AF_XDP, ...
  - Developing XDP features
  - Accelerate nested VM (+vhost_net)
Accelerating method II: Accelerate host VM datapath

- **XDP support in TAP (4.14)**
- **A userspace prog**
  - setup DEVMAP, record the destination interface (batching through flush)
  - `bpf_redirect()` for destination interface
  - return `XDP_REDIRECT`
- **Usecases:**
  - Host networking stack bypassing
  - VM2VM communication
XDP accelerated Receive filter (WIP)

- **Qemu/Libvirt**
  - intercept receive filter request guest through control vq
  - Setup maps for vlan/mac table
  - Attach XDP program to physical interface

- **XDP program**
  - query vlan/mac table for each packet
  - drop packets early if not permitted
  - otherwise redirect them to tap0

```
  +-----------------+          +-----------------+
  | Guest           |         | qemu            |
  +-----------------+          +-----------------+
      |                |         |                 |
      |                |         |                 |
      |                |         |                 |
      |  vhost_net     |         |                 |
      +-----------------+         +-----------------+
                   |         |                 |
                   |         |                 |
                   | userspace|
                   | kernel  |
                   +---------+
                        |
                        |   tape  |
                        +---------+
                              |
                              | XDP     |
                              | Program |
                              +---------+
                                  |
                                  |         |
                                  | XDP     |
                                  | DROP    |
                                  +---------+
                                        |
                                        |
                                        |  NIC   |
                                        +--------+
```

```
  +-----------------+          +-----------------+
  | Guest           |         | qemu            |
  +-----------------+          +-----------------+
      |                |         |                 |
      |                |         |                 |
      |                |         |                 |
      |  vhost_net     |         |                 |
      +-----------------+         +-----------------+
                   |         |                 |
                   |         |                 |
                   |         |                 |
                   | userspace|
                   | kernel  |
                   +---------+
                        |
                        |   tape  |
                        +---------+
                              |
                              | XDP     |
                              | REDIRECT|
                              +---------+
```
XDP accelerated switch (WIP)

- **switchd**
  - build forwarding table through eBPF maps
  - attach XDP to each interfaces

- **XDP program**
  - Mac table via map
  - for packets with destination in the table, redirect it directly through XDP (fast path)
  - for packets with unknown destination, return XDP_PASS and pass it back to normal kernel path for switchd to process (slow path)
**Acceleration Method III: XDP offload to host (WIP)**

- **Offload XDP to host**
  - Almost “self-contained”
  - Lower and faster
    - Datapath run inside host
    - No virt overhead
  - With XDP offload framework
  - Dedicated verifier
    - Classification:
      - Good: Filtering, dropping, XDP_TX
      - BAD: XDP_REDIRECT, debug print
      - Hard to support all from startup
    - Run inside guest if it can’t be offloaded
  - Verified again in host, JIT in host
  - MAPs in host, way to access it from guest userspace
# How XDP help for VM datapath

<table>
<thead>
<tr>
<th></th>
<th>dpdk</th>
<th>Kernel + XDP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Meta-data</strong></td>
<td>rte_mbuf, head of packet</td>
<td>lightweight, XDP_buff, head of packet</td>
</tr>
<tr>
<td><strong>Memory model</strong></td>
<td>memory pool for both TX and RX</td>
<td>vendor specific or generic XDP page pool, frame return API</td>
</tr>
<tr>
<td><strong>DMA</strong></td>
<td>statically mapped</td>
<td>dynamically mapped</td>
</tr>
<tr>
<td><strong>Batching (forwarding)</strong></td>
<td>aggressive</td>
<td>batching via devmap (XDP_REDIRECT)</td>
</tr>
<tr>
<td><strong>Busy polling</strong></td>
<td>Polling Mode Driver</td>
<td>NAPI (polling + event)</td>
</tr>
<tr>
<td><strong>VM datapath in host</strong></td>
<td>vhost pmd</td>
<td>TAP + vhost_net, vhost_net can see XDP buff directly (RX)</td>
</tr>
<tr>
<td><strong>vhost</strong></td>
<td>bulking, prefetching, busy polling</td>
<td>Batch dequeuing XDP buffs from TAP, no prefetching, event based (or limited busy polling)</td>
</tr>
</tbody>
</table>
Packet Per Second (Mpps)

<table>
<thead>
<tr>
<th>Scenario</th>
<th>RX</th>
<th>TX</th>
</tr>
</thead>
<tbody>
<tr>
<td>macvtap</td>
<td>0.88</td>
<td>0.93</td>
</tr>
<tr>
<td>XDP_REDIRECT</td>
<td>4.03</td>
<td>4.14</td>
</tr>
<tr>
<td>dpdk testpmd</td>
<td>14.33</td>
<td>13.94</td>
</tr>
</tbody>
</table>
Future work

- **WIP**
  - XDP accelerated vswitch
  - Virtio-net XDP offload

- **Performance optimization**
  - Send and receive XDP_buff to vhost_net directly
  - Better bulking in TAP/vhost_net
    - (XDP_REDIRECT can give +10Mpps on host between physical ports)
  - Better inter driver co-operation (inter driver page recycling)
  - AF_XDP instead of TAP?

- **More driver support for XDP/XDP_REDIRECT**
  - supported by i40e, ixgbe, tuntap, virtio-net, mlx5(partial)
  - please add XDP_REDIRECT support for your driver! (VF as well)
Summary

• Performance of kernel datapath for VM is improve(ing)!
  – XDP
    • Fast kernel datapath
    • Userspace defined policy
  – More coming soon

• Patches are welcomed!
Thanks!
<table>
<thead>
<tr>
<th>xdp1</th>
<th>Early packet dropping (XDP_DROP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>xdp2</td>
<td>Rewrite (macswap) + forwarding (XDP_TX)</td>
</tr>
<tr>
<td>xdp_tx_iptunnel</td>
<td>Tunneling (header adjustment) + forwarding (XDP_TX)</td>
</tr>
<tr>
<td>xdp_redirect</td>
<td>Simple redirection (XDP_REDIRECT)</td>
</tr>
<tr>
<td>xdp_redirect_map</td>
<td>Batching (DEVMAP) + redirection (XDP_REDIRECT)</td>
</tr>
<tr>
<td>xdp_fwd_kern</td>
<td>ipv4+ipv6 forwarding (XDP_REDIRECT)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>