Efficient and Flexible Virtual Machine Networking through eBPF

Jason Wang
Principal Software Engineer

June 26th 2019
Outline

● Review
● Problems
● Introduction to eBPF/XDP
● Simple usage of XDP
● Advanced Features: advantages and limitations
● Status
● Q&A
Overview of virtual machine networking

- Virtio: IPC between host and guest
- Vhost: virtio dataplane in kernel
- TAP: network driver for userspace, works with OVS/bridge for complex cases
- Macvtap: stacked device on top of lower NIC, for simple use cases
Issues

● Efficiency:
  - Slower than userspace datapath or VF
  - Only get 10% of userspace if measured by PPS
  - No fundamental barrier but why?

● Flexibility:
  - New features were added slowly
    • Developing kernel module is hard
    • Need new kernel/qemu/host to get new feature
  - Mew features just new new firmware (virtual)
eBPF introduction

- **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

```
0: (79) r1 = *(u64 *)(r1 +8)
1: (7b) *(u64 *)(r10 -8) = r1
2: (b7) r1 = 1
3: (7b) *(u64 *)(r10 -16) = r1
4: (18) r1 = 0xffff8801a6098a00
6: (bf) r2 = r10
7: (07) r2 += -8
8: (85) call bpf_map_lookup_elem#1
9: (15) if r0 == 0x0 goto pc+3
```
eBPF introduction (cont)
XDP – eXpressed DataPath

Efficient and Flexible Virtual Machine Networking through eBPF
XDP – Why it was efficient and flexible?

- **Efficiency:**
  - Earliest point: before networking stack
  - Lightweight metadata
  - Driver specific optimization
    - Simple assumption: e.g. page per frame
    - Page recycling: either vendor specific or through page pool
    - Batching: devmap
  - Offloading

- **Flexible:**
  - Co-operate with exist kernel networking stack
    - Management, configuration, debugging, visibility, mature protocol stack
  - Separation policy (either in userspace or well defined exist in kernel) from mechanism (datapath)
  - Generic mode fallback
How does XDP can help

- Virtio-net XDP support accelerates guest datapath
- TAP XDP support processes packets early
- Redirect XDP frames between TAP and another XDP capable NIC to accelerate host datapath
Performance

Mpps for simple packet processing with i40e

- Guest TC + Host skb
- Guest XDP + Host skb
- Guest XDP + XDP_REDIRECT
- Guest XDP + XDP_REDIRECT (linux-next + nosmap)
- Host XDP

64B UDP single flow
Advanced Features

- bpfilter
- XDP offload
- XDP for stacked device
- OVS XDP datapath
- AF_XDP for VM
- eBPF and vhost
bpfilter

- eBPF based backend for iptables
- translate rules of iptables to eBPF and attach to XDP (native, generic or offload)
- bpfilter.ko (to reduce the attack surface)
  - ELF file running in userspace
  - Based on user mode helpers (UMH)
  - Shipped and built from kernel tree, work with modprobe, modinfo
  - Special thread
- Only skeleton merged, main logic is RFC
bpfilter internals

JIT on major archs, offload, verifier, transparent to admin, Write rules in C, ...

userspace kernel

iptables

bpftiler.ko

Rule translation

Netfilter

bpf umh launcher

bpf() syscall

verifier

JIT

BPF program attached and run

TC

Generic XDP

native XDP

HW (offload)
Offload XDP to host?

- No virtualization overhead
- No virtio overhead
- Packet does not need to enter guest if it could be handled by eBPF as fast path! No datacopy in this path.
- XDP_PASS as a fallback to slowpath for the packets can not be dealt with eBPF/XDP
- Further offload to hardware (macvtap)
eBPF transport through virtio

- Userspace program
- bpf()
- virtio
- Qemu
- eBPF proxy
- Guest FD
- Host FD
- bpf() syscall
- intercept bpf() through offload ops
- bypass guest JIT/verifier
- send raw bytecode to Qemu
- qemu loads it on host as a proxy
- only packet manipulation helpers
Service chaining

VM0

VM1

VM2

VM3

XDP prog

eth0

vhost0/TAP0

offloaded XDP prog

native XDP prog

vhost1/TAP1

vhost2/TAP2

vhost3/TAP3

XDP prog

offloaded XDP prog

native XDP prog

offloaded XDP prog

native XDP prog

offloaded XDP prog

native XDP prog

offloaded XDP prog

native XDP prog

eth1

XDP prog

slow path

fast path(zc)
POC performance

Mpps for simple packet processing with i40e

- Guest TC + Host skb
- Guest XDP + Host skb
- Guest XDP + XDP_REDIRECT
- Offloaded XDP for TAP(vhost)
- Offloaded XDP for macvtap(NIC)
- Host XDP
XDP for stacked device

- **Stacked device**
  - The virtual device that is based on the function of lower device: bond, team, macvlan, bridge, OVS, failover, etc
  - Implemented through skb based rx handler

- **Problem:**
  - native XDP can not run on such device (but XDP generic)
  - But production environment use them heavily
  - Userspace topology logic?
XDP for stacked device (example)
XDP rx handler

- tap0
- eth0
- eth1
- bond0

- macvlan0
- macvtap0

- container
- VM
- virtio
- vhost

- XDP prog
- XDP_REDIRECT
- XDP_RX_HANDLER
- packets

Efficient and Flexible Virtual Machine Networking through eBPF
POC Performance

Mpps for simple packet processing with mlx4

perf loss since macvlan_hash_lookup(), offload to hardware?

- macvlan0(XDP generic)
- macvlan0(XDP native)
- mlx4(XDP native)

64B UDP single flow
OVS XDP datapath

- Inspired by OVS TC flower datapath:
- Implement TC flower logic through XDP: tc-xdp?
- OVS control load appropriate XDP program to the interface, or update the action through maps
- Native XDP for acceleration
- XDP generic for fallback
- Can do things that is not easy for hardware offload: e.g contrack
- Limitation: match/action chaining
OVS XDP datapath

```
# xdp-flower add dev eth0 protocol ip \
  action mirred egress redirect dev tap0
```
**AF_XDP (XSK)**

- Evolved from AF_PACKET but based on XDP, up to (20x ?) compares to AF_PACKET
- Optimized ring layout
  - ideas come from virtio 1.1
  - unify?
- Redirect XDP frames to socket directly
- Socket were bound to specific queue
- Two modes:
  - Zerocopy (driver/vendor support)
  - Generic
- Limitation: umem, zc (PIN), packet size limitation, non zerocopy perf is very poor, metadata is too simple
OVS AF_XDP datapath

For better performance, Need Zerocopy support in TAP
Still need to go through vhost_net
AF_XDP passthrough

- A new kind of network device in guest - XSK PT(passthrough) device
  - When bind to XSK, backend can setup AF_XDP socket on host
  - Guest can drive AF_XDP ring on host
  - Inspired by netmap passthrough
- Guest APP still uses AF_XDP ring layout and API
- Host AF_XDP speed were preserved
- XSK PT device were only used for:
  - Configuration
  - Control: start/stop
  - Synchronization: kick/interrupt
eBPF based vhost datapath

- Problem to solve:
  - deal with different ring layout is painful
  - bug fixes need restart datapath
  - POC for new ring layout
- How about decouple the ring layout specific code out of kernel through eBPF
  - Descriptor translation and manipulation being done through eBPF program
  - New ring layout was simply implemented by attaching eBPF program, no new code in kernel
- Challenges:
  - eBPF performance
  - Batching
eBPF based vhost datapath

Guest

Virtio-net drv

virtio

Virtio-net Device model

bytecode for split

bytecode for packed

Qemu

vhost_ioctl()

map

Vhost

XDP prog

XDP frames
Status

- `bpfilter`: only skeleton
- `Virtio-net XDP offload`: POC
- `XDP for stacked device`: generic path, native path RFC
- `OVS XDP datapath`: WIP
- `OVS AF_XDP datapath`: RFC
- `AF_XDP (zerocopy) for TAP`: RFC
- `AF_XDP passthrough`: planning
- `eBPF base vhost datapath`: planning
- `Libvirt support`: planning
Reference

- bpfilter: https://lwn.net/Articles/747504/
- XDP for stacked device: https://lwn.net/Articles/762464/
- AF_XDP: Documentation/networking/af_xdp.rst
- OVS AF_XDP: https://mail.openvswitch.org/pipermail/ovs-dev/2019-April/358373.html
THANK YOU