Why AF_XDP?

A fast and flexible channel between userspace and kernel
• Another way to get better performance besides DPDK
• A more friendly way to do kernel-bypassing
  – Dynamically steering packets using XDP program
• Userspace datapath is easier to maintain than a kernel module
• Share the same datapath with OVS-DPDK

AF_XDP (Userspace) Caveat

- Device directly DMA buffer into userspace
  - OVS runs datapath in userspace (dpif-netdev)

- Difficulties when integrating features inside linux kernel
  - TCP/IP stack
  - Connection tracking using netfilter
  - TC rate limiting
We used the ovs_perf suite for testing
10G ethernet, wirespeed test
Topology: PVP and P tests [single physical port]
OpenFlow rules, NORMAL rule (l2 forwarding)
Packet sizes: 64, 256, 512, 1514
Flows: 1, 100, 1000
No latency tests :

ovs_perf can be found here: https://github.com/chaudron/ovs_perf
Last years presentation: https://ovsfall2018.sched.com/event/IO9n/ovs-and-pvp-testing
• What will we compare?
  – AF_XDP TAP vs Kernel
  – AF_XDP TAP vs AF_XDP VHOST
  – AF_XDP VHOST vs DPDK
  – Native AF_XDP vs AF_XDP DPDK PMD
Kernel datapath results

Physical Port Loopback

Physical to Virtual, L3

Open vSwitch CPU utilization

Total System CPU utilization
PVP test, using single port
NOTE: All native AF_XDP tests were run with use-need-wakeup = true
PVP: kernel tap, vhost_net
AF_XDP userspace datapath vs Kernel datapath

• So for the comparison we pick one test
  – Use the PVP tests, as it represents a real life scenario
  – Use 64 byte packets as this does not fill the pipe
  – Use 100 streams
AF_XDP userspace datapath vs Kernel datapath

AF_XDP vs Kernel graphs show performance metrics such as throughput and error rates. The graphs indicate that AF_XDP has a lower error rate compared to the Kernel, with a higher throughput in certain scenarios.

The diagrams also show a comparison of CPU utilization between AF_XDP and Kernel, with AF_XDP generally having lower CPU usage across different load conditions.

Overall, the data suggests that AF_XDP can offer better performance in terms of both throughput and error rates, while also being more efficient in terms of CPU usage.
AF_XDP userspace datapath vs Kernel, conclusion

• Pros
  – Use less CPU power
  – More throughput
  – No kernel module dependencies

• Cons
  – Missing kernel datapath features, see datapath feature table: https://docs.openvswitch.org/en/latest/faq/releases/
  – It also has no “QoS - Policing support”
  – Traffic from a “kernel” interface uses slow path (same as DPDK)
Physical Port Loopback

Physical to Virtual, L3

Open vSwitch CPU utilization

Total System CPU utilization
PVP: dpdk vhostuser
PVP: dpdk vhostuser

AF_XDP userspace datapath results + DPDK vhost
AF_XDP TAP vs AF_XDP VHOST

AF_XDP

AF_XDP VHOST

AF_XDP

AF_XDP VHOST
AF_XDP TAP vs AF_XDP VHOST, conclusion

**Pros**
- VHOST Use less CPU power (Qemu & TAP)
- Throughput roughly doubles
- Constant CPU usage (even if you add more interfaces)

**Cons**
- Need to setup DPDK also
- Separate memory pool for DPDK (huge pages)
AF_XDP vs DPDK userspace datapath
**Pros**
- Less CPU power needed (can use irq pinning / multiqueue)
- Throughput increase of roughly 1.6x

**Cons**
- Need to setup DPDK
- PMD network driver problems
- Can’t use XDP program steering
OVS with AF_XDP DPDK PMD

- DPDK has a native AF_XDP PMD
- Allow you to use existing DPDK environment
- If enhanced it could allow for packet steering
Physical Port Loopback
PVP: dpdk vhostuser

Physical to Virtual to Physical, L3

Open vSwitch CPU utilization

Total System CPU utilization
Native AF_XDP vs AF_XDP PMD datapath, cont.

• **Pros**
  – Throughput increase
    (due to mbuf reuse vs copy in native AF_XDP)
  – QoS - Policing support

• **Cons**
  – Need to setup DPDK
  – No XDP packet steering (yet)
Future Items

- Shared umem between ports to avoid memcpy [OVS]
  - This is why the AF_XDP PMD performs better

- Native zero copy support for veth/tap interfaces [Kernel]

- VHOST library to avoid including/using DPDK [OVS]

- Egress QoS support for AF_XDP interfaces [OVS]
Future Items, cont.

- CI testing of AF_XDP [OVS]
- Load custom XDP programs [OVS]
  - Patch is currently on the mailinglist:
    netdev-afxdp: Enable loading XDP program
- Allow more finegrane driver loading [OVS]
  - skb mode, or driver mode with or without zero-copy
  - Patch is currently on the mailinglist:
    netdev-afxdp: Best-effort configuration of XDP mode
Conclusion

• Stuff we did not do
  – Compare latency
  – Compare multiqueue support

• AF_XDP sits between kernel and DPDK
  – From throughput and CPU usage perspective
  – Missing some kernel feature (and DPDK QoS - Policing support)

• AF_XDP requires kernel support
  – But if the kernel support AF_XDP there is no kernel module dependency