Userspace networking: beyond the kernel bypass with RDMA!

Using the RDMA infrastructure for performance while retaining kernel integration

Benoît Ganne, bganne@cisco.com
Why a native network driver?

- Why userspace networking?
  - **Performance** (avoid kernel overhead)
  - Update network functions seamlessly (no reboot required, containerization)

- Why your own network driver?
  - **Performance** (metadata translation tax, feature tax)
  - Ease-of-use (no reliance on hugepages, etc.)

- Why you should think twice?
  - No integration with kernel (interface fully owned by userspace)
  - You care about rx/tx packets but device initialization & setup is 95% of the work
  - *Hardware is hard* (more on that later)

Source: [https://docs.fd.io/csit/master/report/vpp_performance_tests/packet_throughput_graphs/ip4-2n-slxv710.html](https://docs.fd.io/csit/master/report/vpp_performance_tests/packet_throughput_graphs/ip4-2n-slxv710.html)
RDMA

• « Remote Direct Memory Access »
  • Designed for message passing and data transfer
  • Has evolved to use Ethernet transport (iWARP, RoCE)

• Key properties
  • Hardware offload
  • Kernel bypass
  • Zero Copy data transfer
  • High network bandwidth

➤ Great for kernel networking!
Extending RDMA for Ethernet

• Not designed for efficient Ethernet communication – but!
  • Ethernet-capable HW (initially for transport)
  • High performance (200Gbps today)
  • Kernel bypass with well established API and native Linux kernel support

• Why not extend it to support userspace networking?
  • Introduce new IBV_QPT_RAW_PACKET queue pair type
  • Support for bifurcation with flow steering
    • Keep your Linux netdev
    • Support MACVLAN, IPVLAN model…

Incoming packets are steered to Linux netdev or userspace application based on flows
Using RDMA for Ethernet

How to send 20 Mpps with 1 CPU

1. Get a handle to the device you want to use
   
2. Initialize queues
   - Queue Pair (QP) = Submission Queue (SQ) + Completion Queue (CQ)
   - Protection Domain (PD) = where the NIC is allowed to read/write data (packets)

3. Send packets
   - Put Work Queue Elements (WQE – kind of IOV) in SQ
   - Notify new packets to send
   - Poll CQ for completion

Full example at https://github.com/bganche/rdma-pktgen

```c
/* step 1: get device handle */
dev = ibv_get_device_list(&dev_nb);
ctx = ibv_open_device(dev[i]);

/* step 2: initialize queues for RX/TX */
cq = ibv_create_cq(ctx, TXNB, 0, 0, 0);
pd = ibv_alloc_pd(ctx);
qp = ibv_create_qp(pd, type=IBV_OPT_RAW_PACKET);
ibv_modify_qp(qp, state=IBV_QPS_INIT, IBV_QP_STATE | IBV_QP_PORT);
ibv_modify_qp(qp, state=IBV_QPS_RTR, IBV_QP_STATE);
ibv_modify_qp(qp, state=IBV_QPS_RTS, IBV_QP_STATE);

mr = ibv_reg_mr(pd, addr, len, 0);

/* step 3: RX/TX in a busy loop */
while (1) {
    ibv_poll_cq(cq, TXNB, wc);
    ibv_post_send(qp, wr[wc[i].wr_id], 0);
}
```
Going deeper with Direct Verbs

- RDMA user API is ibverb
  - Simple enough, mostly standard, open-source
  - Not full performance (metadata translation tax, feature tax)
- Direct Verbs
  - ibverb extension to access DMA ring-buffers directly
  - Hardware-dependent!
  - Setup done through ibverb, then get DMA rings addresses

```c
/* convert ibverb SQ/CQ to DV SQ/CQ */
struct mlx5dv_obj obj = {
    .qp = { .In = ibv_qp, .out = dv_qp },
    .cq = { .In = ibv_cq, .out = dv_cq },
};
mlx5dv_init_obj (&obj, MLX5DV_OBJ_Q | MLX5DV_OBJ_QP);

/* get SQ and doorbell addresses */
sq_base = dv_qp->sq.buf;
sq_dbrec = dv_qp->dbrec;
sq_db = dv_qp->bf.reg;
sq_sz = dv_qp->sq.wqe_cnt;

/* get CQ and doorbell addresses */
cq_cqes = dv_cq->buf;
cq_dbrec = dv_cq->dbrec;
cq_sz = dv_cq->cqe_cnt;
```
VPP native RDMA driver

• ibverb version
  • Available since 19.04
  • ~ 20 Mpps L2-xconnect per core

• Direct Verb
  • Development underway
  • *Hardware is hard*: while trying to debug my driver I almost bricked my NIC

• Next
  • Add support for hardware offloads (checksum offload, TSO)
A call to action

• We love this model
  • No need to write code boilerplate to initialize the NIC: we can focus on what matters (rx/tx packets)
  • Seamless integration with Linux kernel
  • Great performance

• But is has limitations
  • Need RDMA-capable NIC: must support Hardware security model, etc.
  • Only supported on Mellanox for now

• Could other technologies enable this approach?
  • Disclaimer: a bit outside of my domain knowledge here…
  • vfio-mdev?
  • AF_XDP?