Zero Copy Rx with io_uring

Pavel Begunkov and David Wei
Agenda

01 Problem statement
02 io_uring primer
03 Design
04 Preliminary results
05 Status + future work
06 Questions + discussions
01 Problem statement
Linux networking Rx overheads

- Memory and PCIe bandwidth bottlenecks
- Memcpy CPU overheads
Kernel bypass

- High throughput! Low latency!
- But libraries and applications expect kernel networking stack
- Re-architecting an entire system around kernel bypass is expensive
Proposal

- Hybrid solution
  - Standard control plane using kernel networking stack
  - Fast ZC Rx data plane using io_uring
- Two parts:
  - sk_buffs with page frags pointing to userspace pages end up in sockets
  - Read from socket using io_uring
02  io_uring primer
io_uring

- Rings shared between kernel and userspace
- Userspace submit requests into Submission Queue (SQ)
- Kernel posts completions into Completion Queue (CQ)
- Kick off work by entering kernel

Image credit:
https://medium.com/nttlabs/rust-async-with-io-uring-db3fa2642dd4
prepare request

```
struct io_uring_sqe *sqe;

sqe = io_uring_get_sqe(ring);
io_uring_prep_recv(sqe, sockfd, buf, len, flags);

Note this already moves the SQ tail
```
Submit

```rust
io_uring_submit_and_wait(ring, nr_completions);
```

Image credit:
https://medium.com/nttlabs/rust-async-with-io-uring-db3fa2642dd4
unsigned head;
int count = 0;
io_uring_for_each_cqe(ring, head, cqe) {
    // do stuff
    count++;
}
io_uring_cq_advance(ring, count);

Process completions

Image credit: https://medium.com/nttlabs/rust-async-with-io-uring-db3fa2642dd4
03  Design
Buffer management

- sk_buffs with page frags pointing to userspace pages end up in sockets
- To do this:
  - Fill hardware Rx queue filled with *userspace* pages
Buffer management: registration

- Register userspace memory with io_uring
- Pin pages
- `struct bio_vec` bvec[]
Buffer management:
fill HW Rx queue

- Page pool evolving to become generic allocator for NICs
- Add ZC page pool “inspired” by page pool
- Thin shim layer + driver changes
Header splitting + flow steering

- Only want payload
  - Header splitting
- Only want our specific application flows to hit our ZC hardware Rx queues
  - Flow steering
  - RSS
Kernel network stack

- Hardware side fully set up
- Hard IRQs
- Softirq - NAPI poll
- Construct sk_buffs
  - Marked as ZC Rx
  - Page frags → userspace pages
- Goes through networking stack
More rings

- Add two new shared ringbufs to io_uring:
  - Rx queue
  - Refill queue
- One pair per hardware Rx queue
Userspace: submit request

- Submit ZC receive request to io_uring
- Get SQE, prep, and submit
io_uring:
read socket

- Handle ZC receive request
- Read sk_buffs from socket
- No copy - payload already in userspace
- Post one ZC Rx queue entry per skb page frag

```
struct io_uring_rbuf_cqe {
    u32 off;
    u32 len;
    u16 region;
    u8 sock;
    u8 flags;
}
```
io_uring: notify userspace

- Post completion event into CQ
- Tells userspace to go look at a ZC Rx queue
Userspace: read data

- Look at a ZC Rx queue
- Each entry tells user where the payload is relative to the registered memory region

```c
struct io_uring_rbuf_cqe {
    u32 off;
    u32 len;
    u16 region;
    u8 sock;
    u8 flags;
};
```
Userspace: return buffers

- Return buffers to ZC page pool via refill queue
- Eventually used by NIC driver to refill hardware Rx queue

```c
struct io_uring_rbuf_rq {  
  u32 off;  
  u32 len;  
  u16 region;  
}
```
04 Preliminary Results
MemBW

Broadcom BCM57504 NIC @ 25 Gbps link
62 GB DRAM
iperf3 + io_uring + ZC Rx
AMD EPYC 7D13
iperf3
uProf
**MemBW**

Broadcom BCM57504 NIC @ 25 Gbps link
62 GB DRAM
iperf3 + io_uring + ZC Rx
Intel Xeon Platinum 8321HC
iperf3
pcm-memory

DDIO is off
05 Status + future work
Status

- V2 RFC is on the mailing list (netdev + io-uring)
- Hacky veth support if you want to play with the API
- Broadcom bnxt hardware support
- Multi-socket
- Copy fallback
Future work

- Jakub Kicinski’s memory provider API
- Proper test device
  - netdevsim?
- Tying flow steering rules with socket
- Dynamic Rx queue reconfiguration
- Support GPU device memory
  - Using Google’s TCP devmem proposal
06 Questions + discussions
07 Appendix
Open questions

- Containers + VMs support?
- TLS + kTLS?
- HugePages?
Copy fallback

- What if we run out of userspace memory allocated for ZC Rx?
- Fill HW Rx queue with kernel pages - as before
- When io_uring ZC receive finds sk_buffs with page frags that are not ZC pages, copy into a page from refill queue
Handling errors

• How much to allocate ahead of time?
• What if it runs out?
• What if header splitting fails?
  – Split too little - header malformed
  – Split too much - payload included
• What if flow steering fails?
  – ZC Rx packet ends up in non-ZC Rx queue
  – Non-ZC Rx packet ends up in ZC Rx queue
Integrating ZC Rx well

- NIC → userspace memory is only one hop in a long end to end pipeline
- What if data needs to be modified after ZC Rx? Another copy...
- API need to expose fine control over the placement of data to satisfy constraints e.g. alignment
  - Hardware also needs to support this too