io_uring: path to zerocopy

Kernel Recipes 2022

Pavel Begunkov

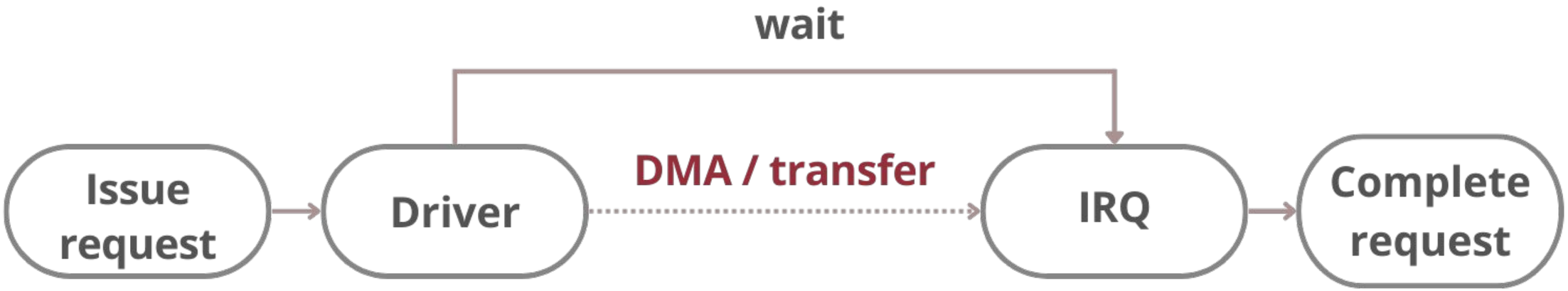
FACEBOOK

Zerocopy

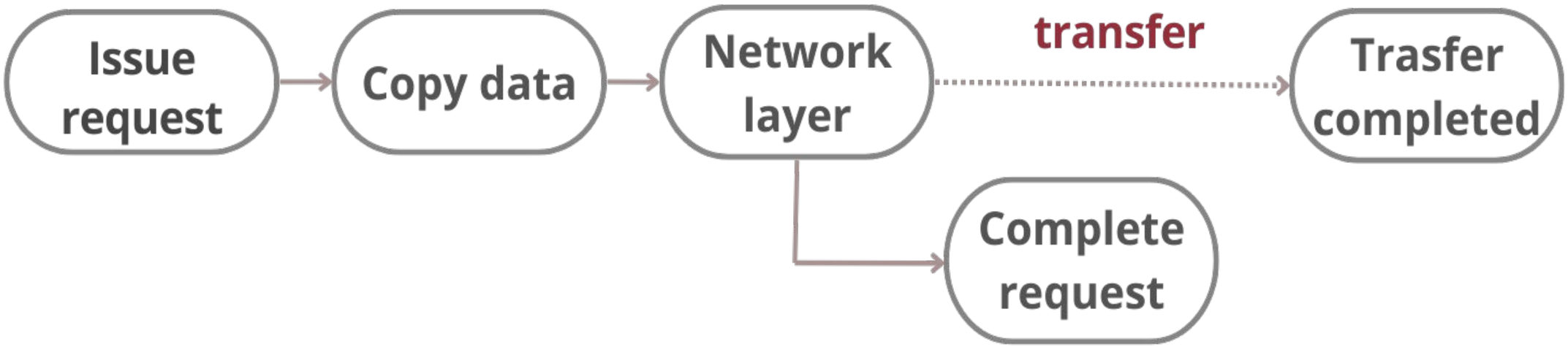
What it's the goal?

- a good way to zerocopy with io_uring
- and have a more consistent API when possible
- improving zerocopy performance
- peer-to-peer DMA

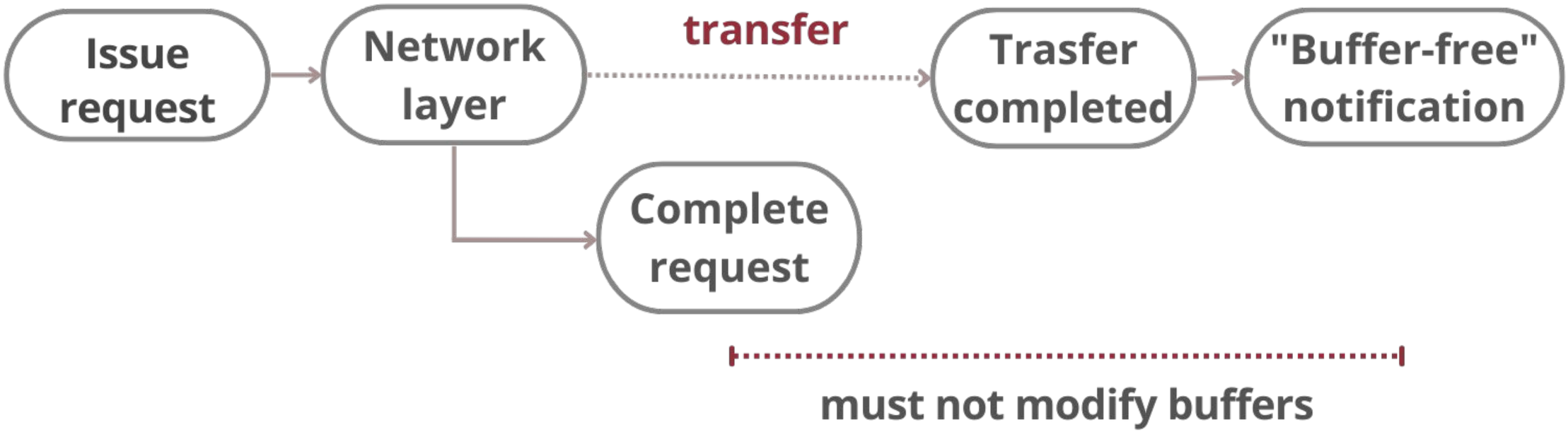
Storage I/O


wait
Issue request
Driver
DMA / transfer
IRQ
Complete request

Network send



Network send: MSG_ZEROCOPY



Storage-like

Pros:

- simple and easy to use
- 1 CQE per request, more efficient (?)

Cons:

- can’t append w/o waiting for an ACK
- forces TCP to alloc a new skbuff for each request

Two-step

Cons:

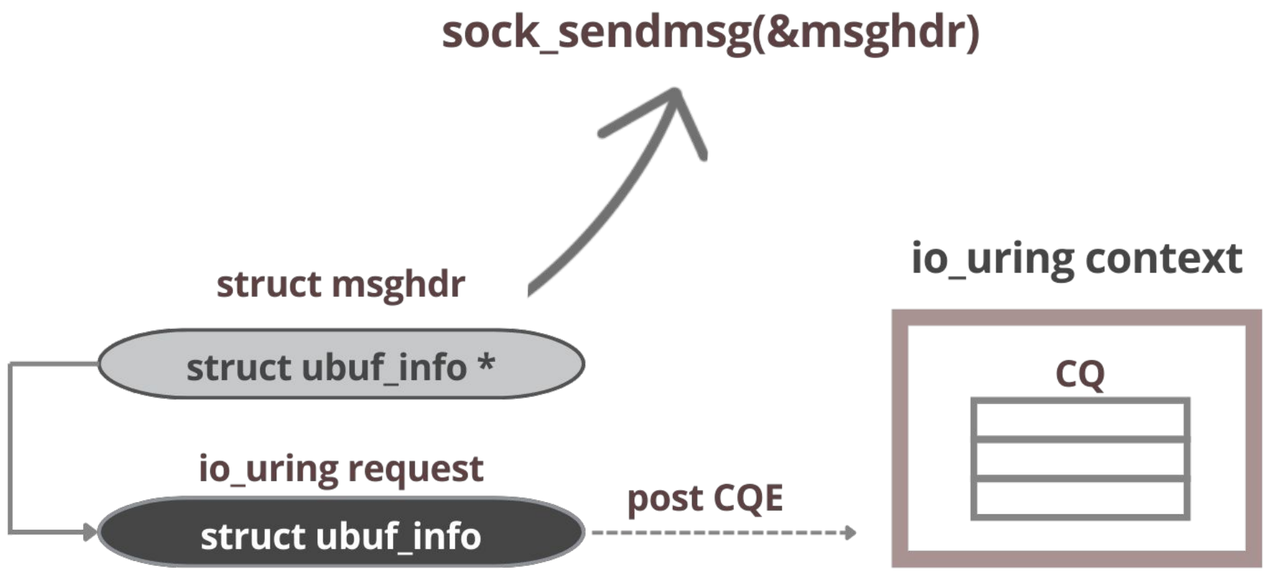
- >1 CQEs
- More cumbersome

Pros:

- but works with TCP
- more flexible
- can simulate the storage style w/ flags

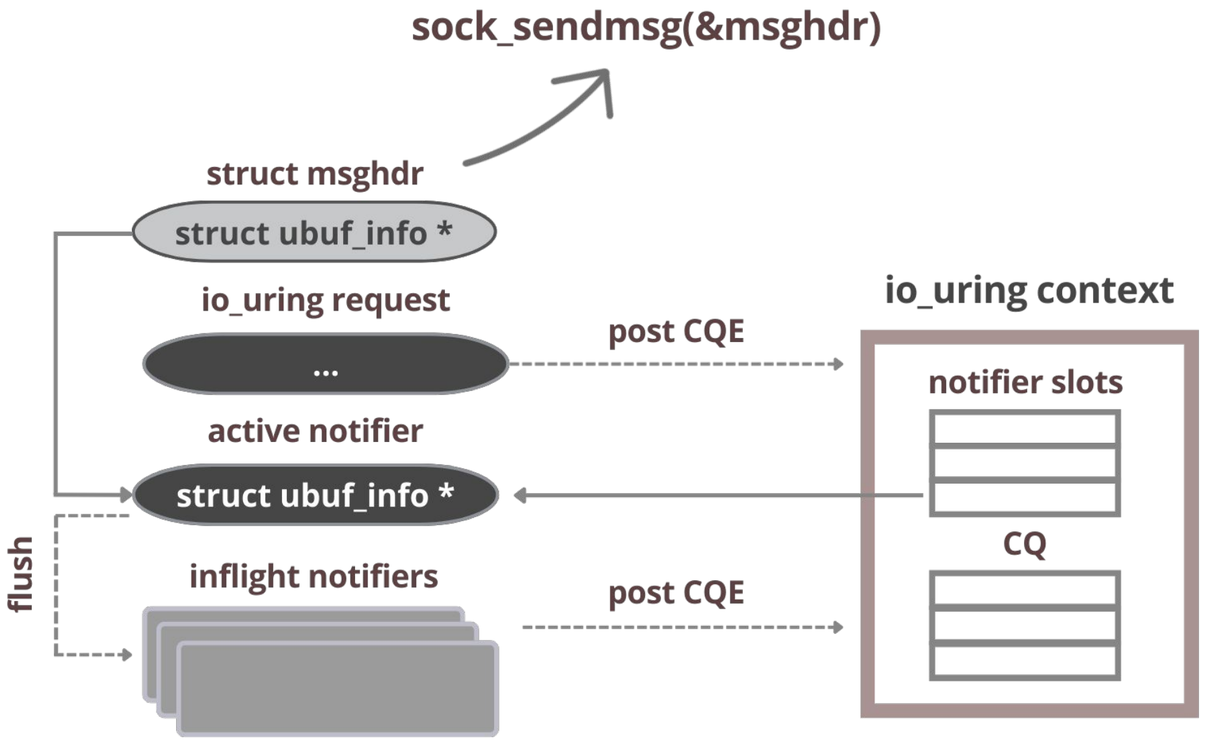
v1: “Storage” style send

```
struct msghdr {
  struct ubuf_info *ubuf;
};
struct io_kiocb { // io_uring request
  struct ubuf_info ubuf;
  struct io_ring_ctx *ctx;
};

void io_sendzc(req) {
   struct msghdr = { .ubuf = &req->ubuf; };
   req->ubuf.callback = io_zc_callback;
   sendmsg(req->file, &msghdr);
   ubuf_put(&req->ubuf); // might trigger ubuf->callback()
}
// ubuf.refs dropped to zero
void io_zc_callback(struct ubuf_info *ubuf) {
  struct io_kiocb *req = container_of(ubuf);
  io_uring_post_completion(req);
};
```



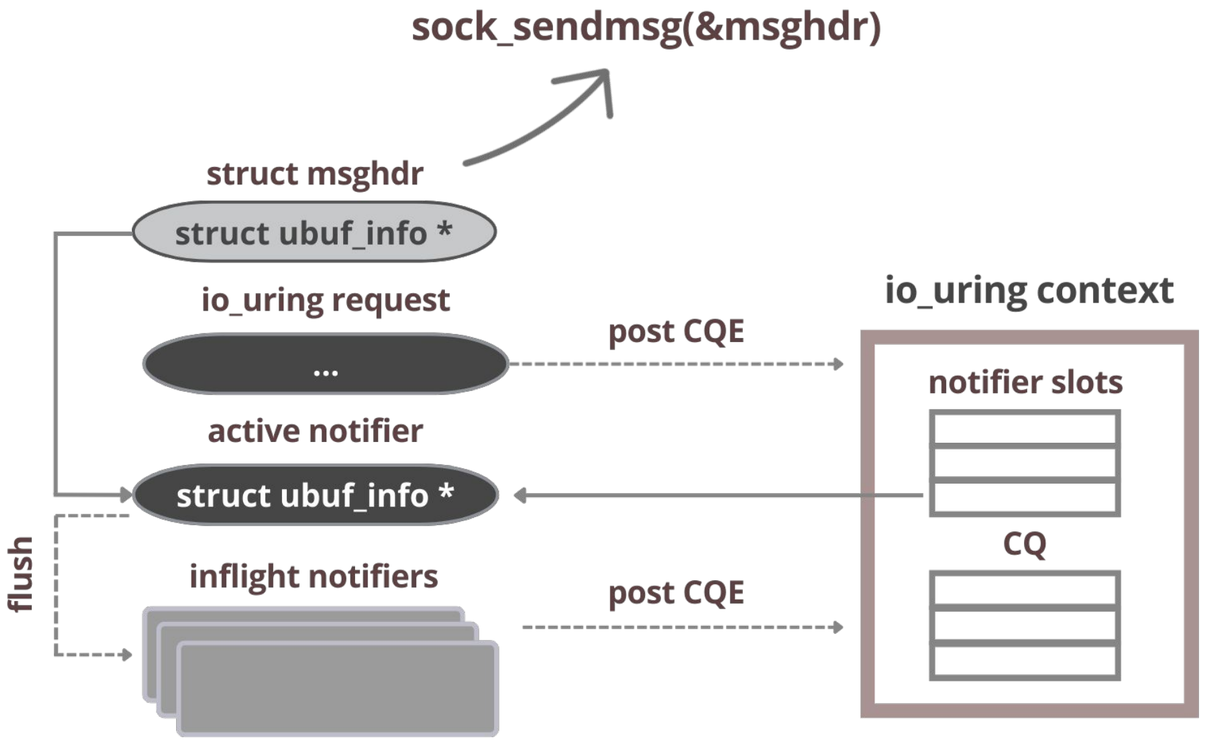
v2: notification registration

```
struct notification_slot_desc { // uapi
    __u64 user_data;
};
struct notification_slot {
    struct ubuf_info *cur_ubuf;
    u64 user_data; // passed back in cqe::user_data
};
struct io_ring_ctx {
    struct notification_slot slots[];
};
void io_uring_register_notifications(
            __user u64 *user_tags, int nr) {
    ctx->slots = alloc();
    for_each_slot(i, slot) {
       slot->tag = user_tags[i];
       slot->cur_ubuf = alloc();
       // put on flush
       slot->cur_ubuf.refs = 1;
       slot->cur_ubuf.callback = io_zc_callback;
    }
}
```



v2: notification binding

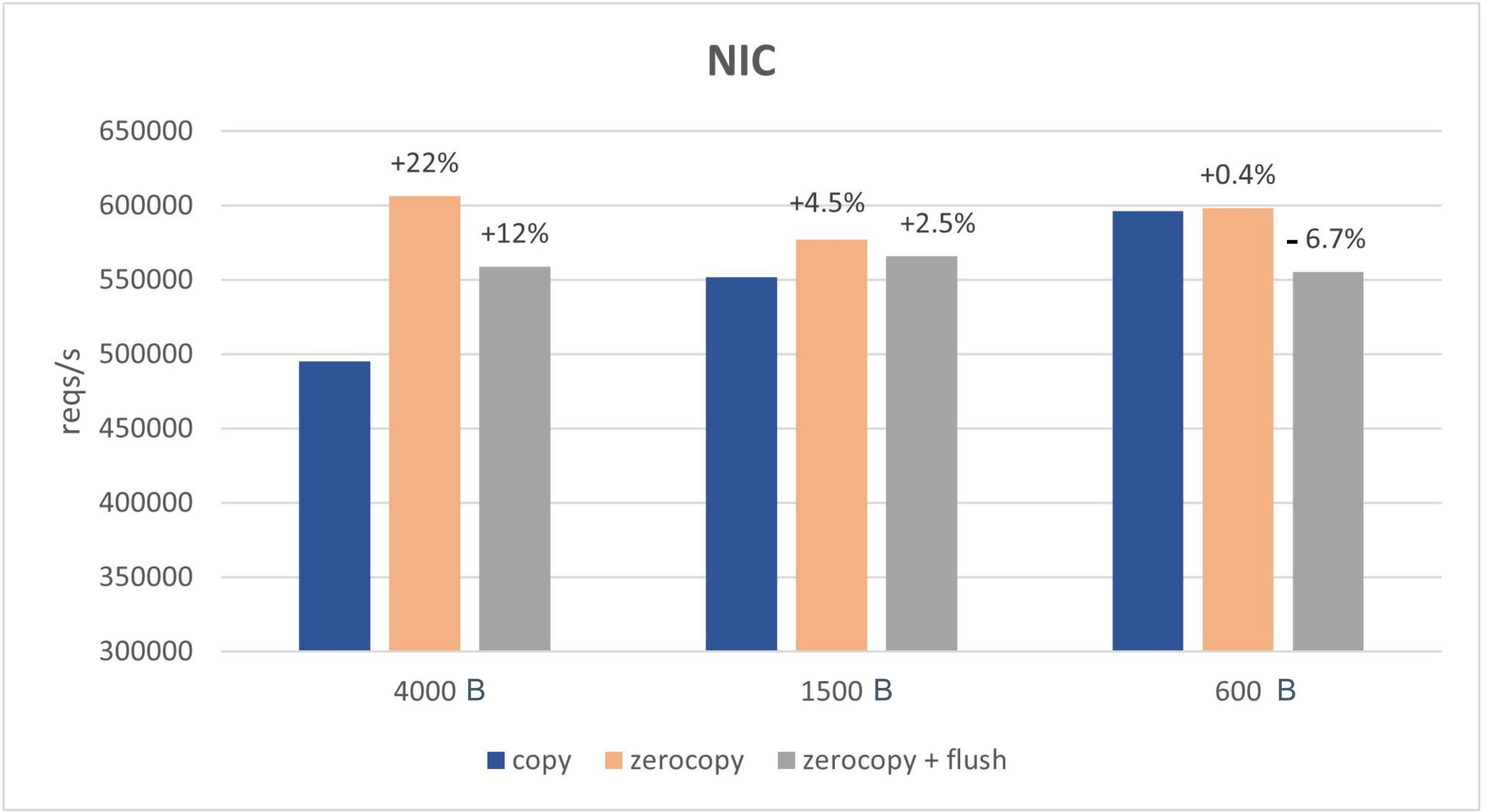
```
struct io_kiocb { // io_uring request
   struct io_ring_ctx *ctx;
   int slot_idx; // ctx->slots
};
void io_uring_request_sendzc(req) {
   msghdr.ubuf = ctx->slots[req->slot_idx].cur_ubuf;
   zcopy_get(msghdr.ubuf);
   sendmsg(&msghdr);
   complete_req(req);
}

void io_uring_request_flush_notification(req) {
   slot = &ctx->slots[req->flush_idx];
   zcopy_put(slot->cur_ubuf);
   slot->cur_ubuf = alloc_new_ubuf();
}
// called when ubuf.refs are dropped to zero
void io_zc_callback(struct ubuf_info *ubuf) {
   io_uring_post_cqe(ubuf);
};
```

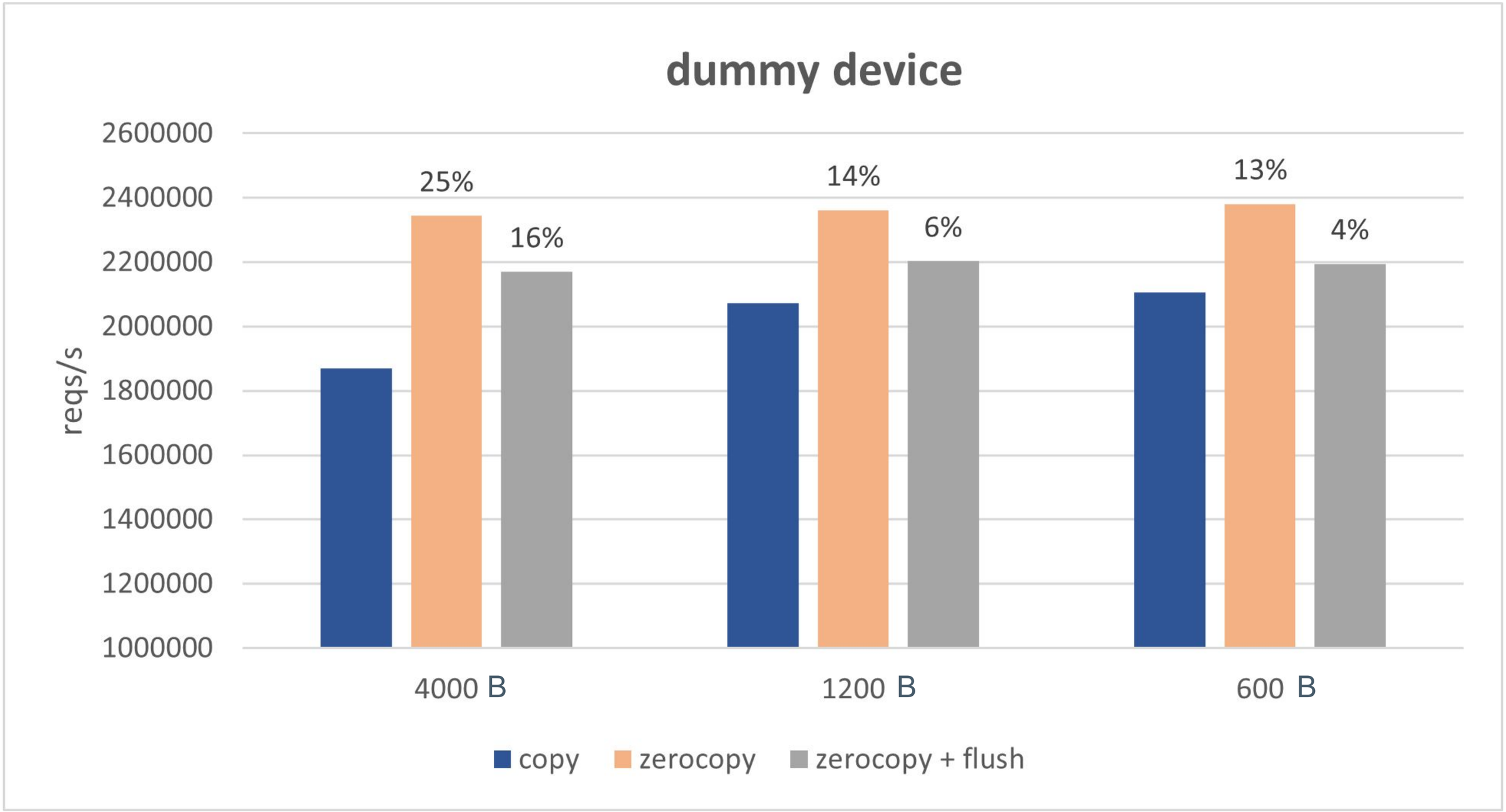


Performance

- Notifications are in the CQ, no extra syscall
- Optionally, can use registered buffers
  - No page table traversals
  - No hot path mm accounting
  - No page refcounting

    io_uring binds lifetime of the pages to ubuf_info
- Cached ubuf_info allocation
- Amortised ubuf_info refcounting


NIC
650000
600000
550000
500000
450000
400000
350000
300000
reqs/s
+22%
+12%
+4.5%
+2.5%
+0.4%
- 6.7%
4000 B
1500 B
600 B
copy
zerocopy
zerocopy + flush


dummy device
reqs/s
2600000
2400000
2200000
2000000
1800000
1600000
1400000
1200000
1000000
25%
16%
14%
6%
13%
4%
4000 B
1200 B
600 B
copy
zerocopy
zerocopy + flush

WIP: DMA, peer-to-peer and dmabuf

```
// normal buffer registration
struct iovec vecs[] = {...};
struct io_uring_rsrc_update2 upd = {
   .data = iovecs,
};
io_update_buffers(&upd);

// userspace dma registration
struct {
   int dma_buf_fd;
   struct iovec vec;
   int target_fd;    // e.g. -1, socket or bdev
   int flags;
} bufs[] = {...};

struct io_uring_rsrc_update2 upd = {
   .data = bufs;
};
io_update_buffers(&upd);
```

Work in progress, points for discussion

- uniform API for block, network, etc.
- **p2pdma** as a backend (need net support)
- **dmabuf** frontend
- ->target_fd is used to resolve "`struct device`"
- might need a notion of device groups
- optional caching of DMA mappings

Common pain:

p2pdma need to be backed by struct pages

Future plans: zerocopy receive

No new bright ideas

- mmap: TCP_ZEROCOPY_RECEIVE
- providing buffers: **zctap / AF_XDP**

The current sentiment is to take the **zctap** / **AF_XDP** approach

- Hardware limitations
- Userspace provides a pool of buffers