Doubling FreeBSD request-response throughputs over TCP with PASTE

Michio Honda, Giuseppe Lettieri

AsiaBSDCon 2019

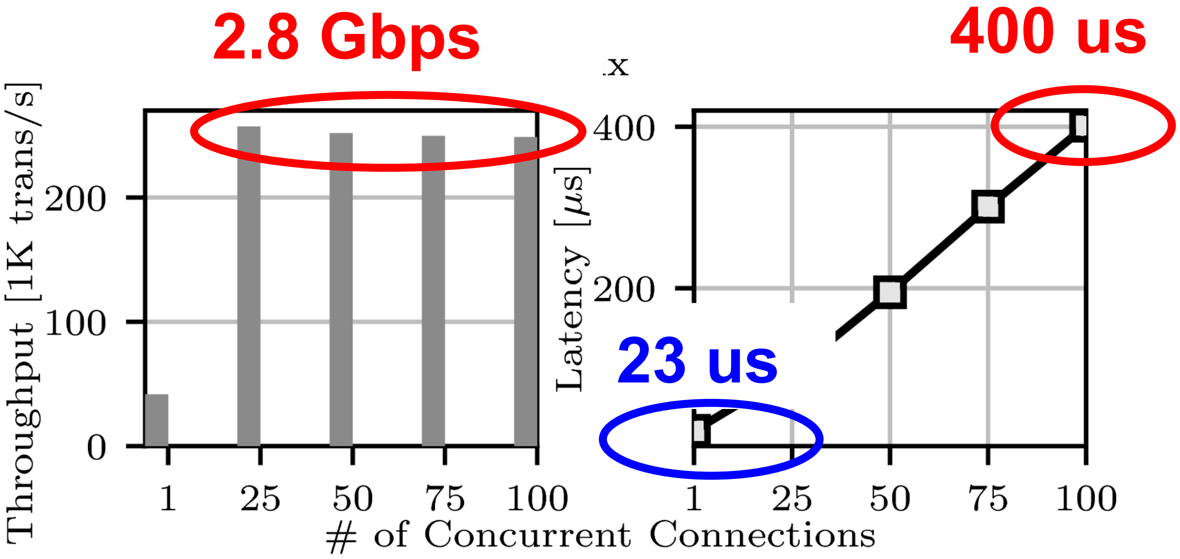
Contact: @michioh, micchie@sfc.wide.ad.jp
Code: https://micchie.net/paste/
Paper: https://www.usenix.org/conference/nsdi18/presentation/honda

Disk to Memory

- Networks are faster, small messages are common
  - System call and I/O overheads are dominant
- Persistent memory is emerging
  - Orders of magnitude faster than disks, and byte addressable
- read(2)/write(2)/sendfile(s) resemble networks to disks
- **We need APIs for in-memory (persistent) data**

Case Study: Request (1400B) and response (64B) over HTTP and TCP

```
n = kevent(fds)
for (i=0; i<n; i++) {
    read(fds[i], buf);
    ...
    write(fds[i], res);
}
```



2.8 Gbps

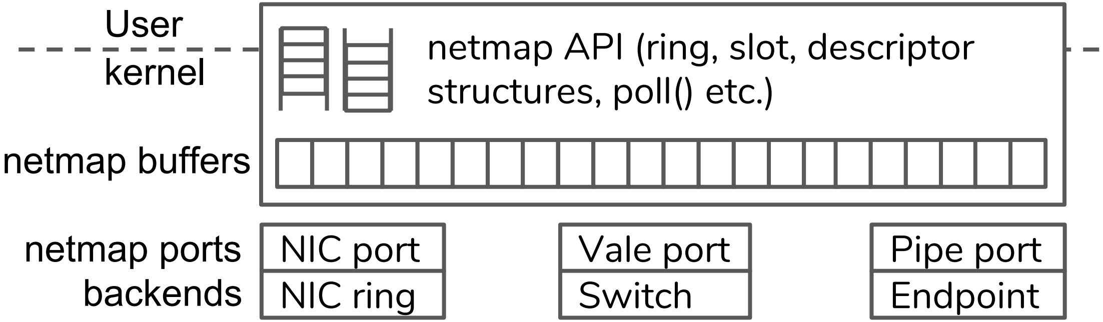
400 us

23 us

Server has Xeon 2640v4 2.4 Ghz (uses only 1 core) and Intel X540 10 GbE NIC
Client has Xeon 2690v4 2.6 Ghz and runs `wrk` HTTP benchmark tool

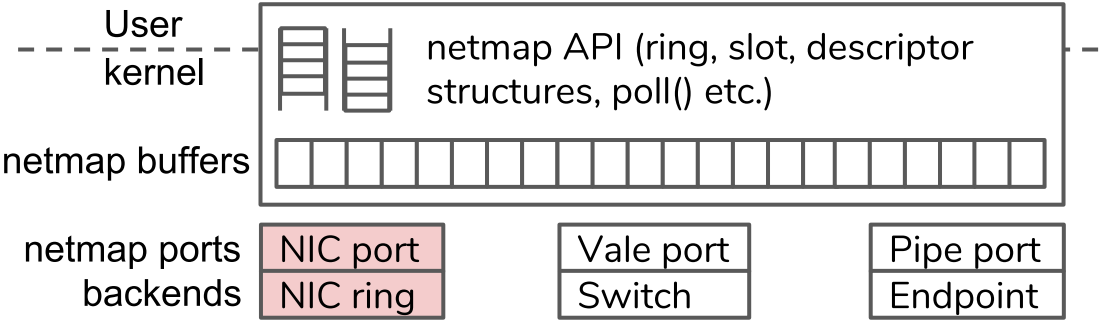
Starting point: netmap (4)

- NIC’s memory model as abstraction
  - Efficient raw packet I/O



Starting point: netmap (4)

- NIC’s memory model as abstraction
  - Efficient raw packet I/O



User

kernel

netmap API (ring, slot, descriptor structures, poll() etc.)

netmap buffers

netmap ports

backends

NIC port

NIC ring

Vale port

Switch

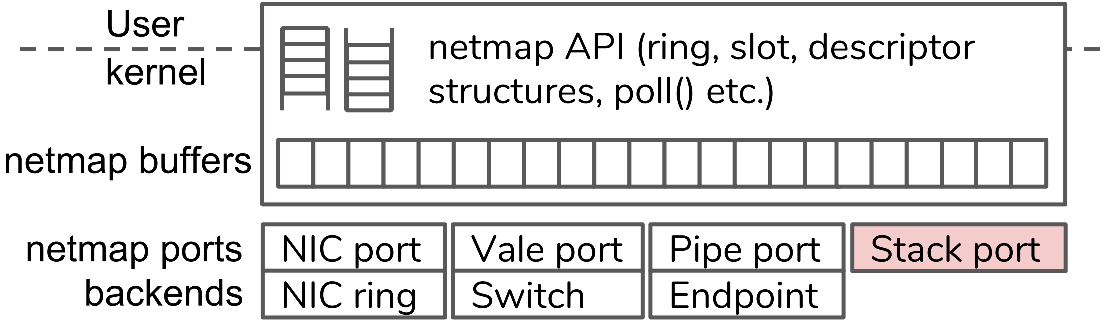
Pipe port

Endpoint

```
nmd = nm_open(“netmap:ix0”);
struct netmap_ring *ring =
  nmd->rx_rings[0];
while () {
  struct pollfd pfd[1] = {nmd};
  poll(pfd, 1);
  if (!(pfd[0]->revent & POLLIN))
    continue;
  int cur = ring->cur;
  for (; cur != ring->tail;) {
    struct netmap_slot *slot;
    int l;
    slot = ring->slot[cur];
    char *p = NETMAP_BUF(ring, cur);
    l = slot->len;
    /* process packet at p */
    cur = nm_next(ring, cur);
  }
}
```

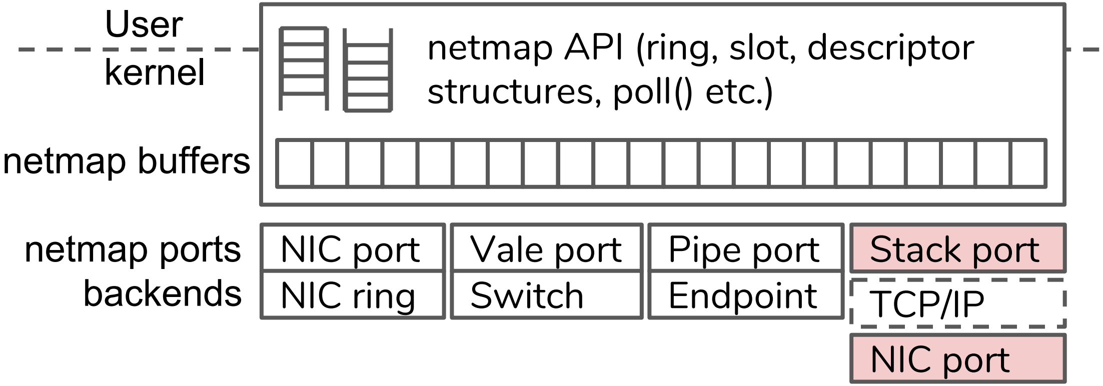
netmap (4) w/ PASTE

- NIC's memory model as abstraction
  - Efficient raw packet I/O



netmap (4) w/ PASTE

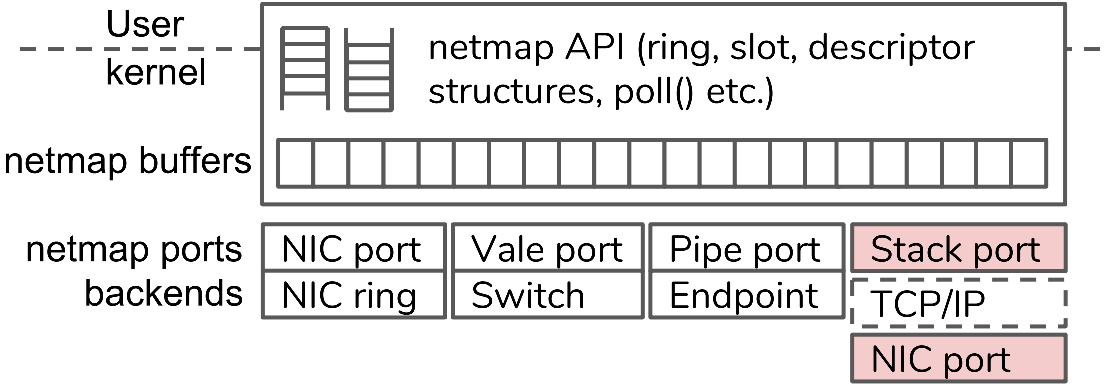
- NIC’s memory model as abstraction
  - Efficient raw packet I/O



netmap (4) w/ PASTE

```
nmd = nm_open("stack:0");
ioctl(nmd, NIOCCONFIG, "stack:ix0");
struct netmap_ring *ring =
  nmd->rx_ring[0];
s = socket(); bind(s); listen(s);
```

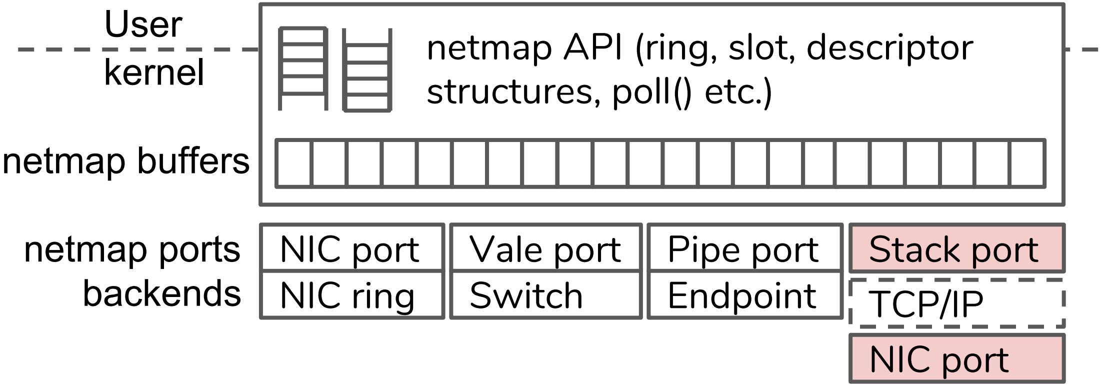
- NIC's memory model as abstraction
    - Efficient raw packet I/O



netmap (4) w/ PASTE

- NIC’s memory model as abstraction
  - Efficient raw packet I/O

```
nmd = nm_open(“stack:0”);
ioctl(nmd, NIOCCONFIG, “stack:ix0”);
struct netmap_ring *ring =
  nmd->rx_ring[0];
s = socket(); bind(s); listen(s);
while () {
  struct pollfd pfd[2] = {nmd, s};
  poll(pfd, 2);
  if (pfd[1]->revent & POLLIN) {
    new = accept(s);
    ioctl(nmd, NIOCCONFIG, &new);}
```



User

kernel

netmap API (ring, slot, descriptor structures, poll() etc.)

netmap buffers

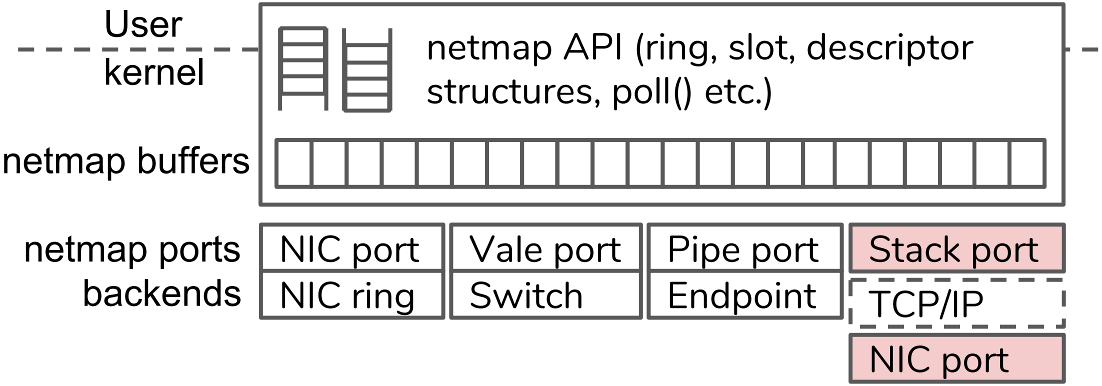
netmap ports

backends

NIC port	Vale port	Pipe port	Stack port
NIC ring	Switch	Endpoint	TCP/IP
			NIC port

netmap (4) w/ PASTE

- NIC’s memory model as abstraction
  - Efficient raw packet I/O



User

kernel

netmap API (ring, slot, descriptor structures, poll() etc.)

netmap buffers

netmap ports

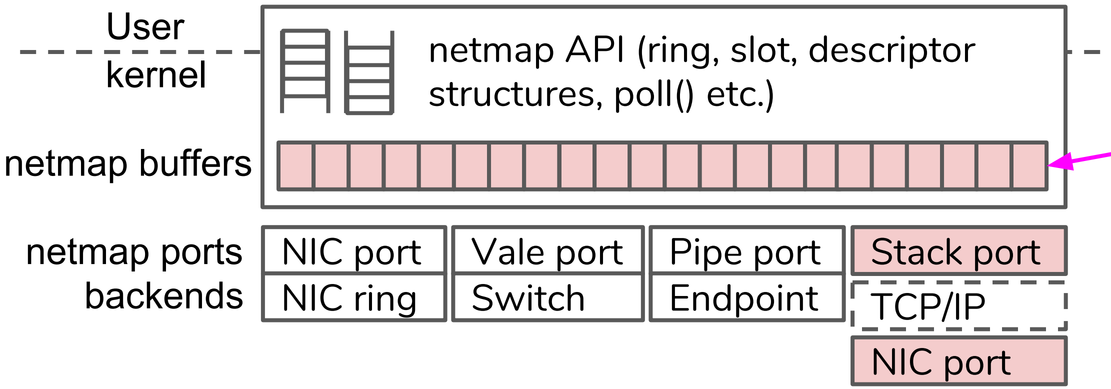
backends

NIC port	Vale port	Pipe port	Stack port
NIC ring	Switch	Endpoint	TCP/IP
			NIC port

```
nmd = nm_open(“stack:0”);
ioctl(nmd, NIOCCONFIG, “stack:ix0”);
struct netmap_ring *ring =
  nmd->rx_ring[0];
s = socket(); bind(s); listen(s);
while () {
  struct pollfd pfd[2] = {nmd, s};
  poll(pfd, 2);
  if (pfd[1]->revent & POLLIN) {
    new = accept(s);
    ioctl(nmd, NIOCCONFIG, &new);}
  if (!(pfd[0]->revent & POLLIN))
    continue;
  int cur = ring->cur;
  for (; cur != ring->tail;) {
    struct netmap_slot *slot;
    int l, fd, off;
    slot = ring->slot[cur];
    char *p = NETMAP_BUF(ring,cur);
    l = slot->len;
    fd = slot->fd;
    off = slot->offset;
    /* process data at p + off */
    cur = nm_next(ring, cur);
  }
}
```

netmap (4) w/ PASTE

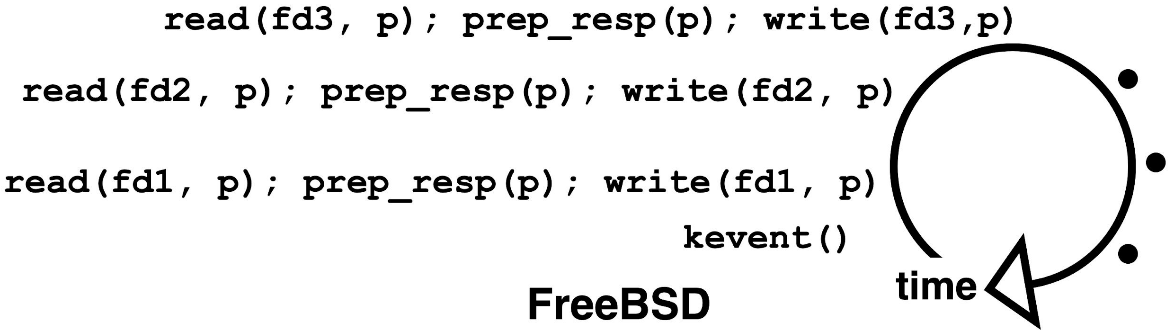
- NIC's memory model as abstraction
    - Efficient raw packet I/O



```
m = mmap("/mnt/pmemfs/pmemfile")
nmd = nm_open("stack:0", m);
```

System Call and I/O Batching, and Zero Copy

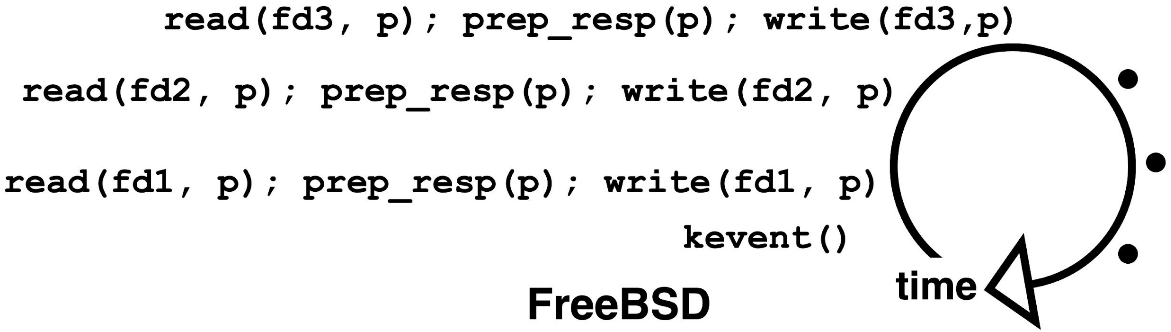
- FreeBSD suffers from per-request read/write syscalls



```
read(fd3, p); prep_resp(p); write(fd3,p)
read(fd2, p); prep_resp(p); write(fd2, p)
read(fd1, p); prep_resp(p); write(fd1, p)
kevent()
```

System Call and I/O Batching, and Zero Copy

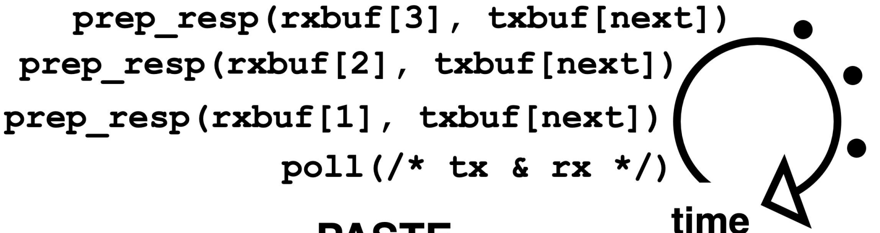
- FreeBSD suffers from per-request read/write syscalls



```
read(fd3, p); prep_resp(p); write(fd3,p)
read(fd2, p); prep_resp(p); write(fd2, p)
read(fd1, p); prep_resp(p); write(fd1, p)
kevent()
```

FreeBSD

- PASTE does not need that
- I/O is also batched under poll()

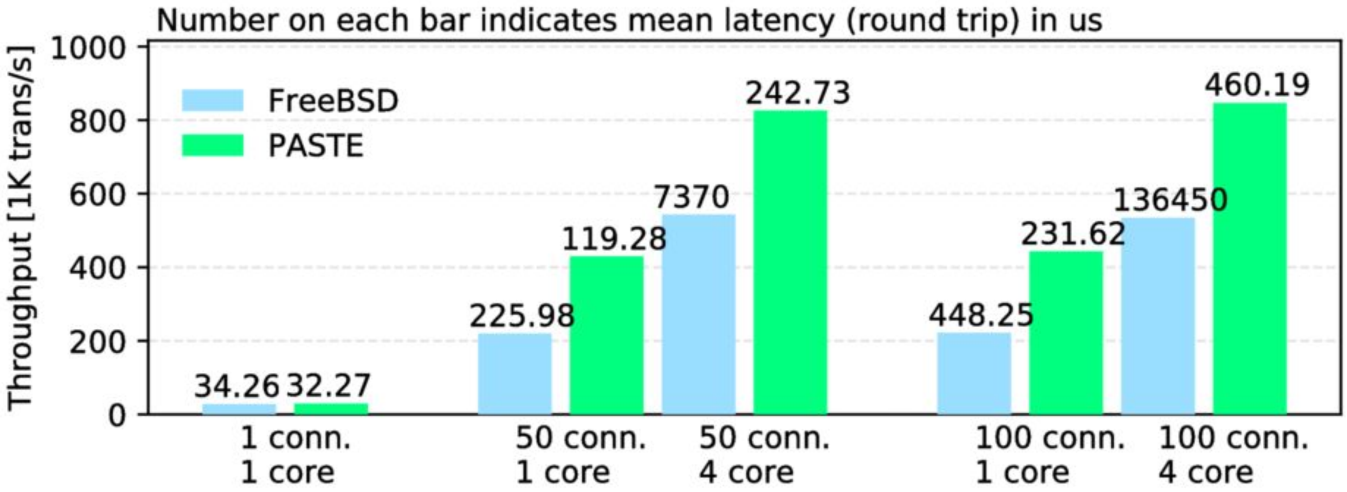


```
prep_resp(rxbuf[3], txbuf[next])
prep_resp(rxbuf[2], txbuf[next])
prep_resp(rxbuf[1], txbuf[next])
poll(/* tx & rx */)
```

PASTE

Performance

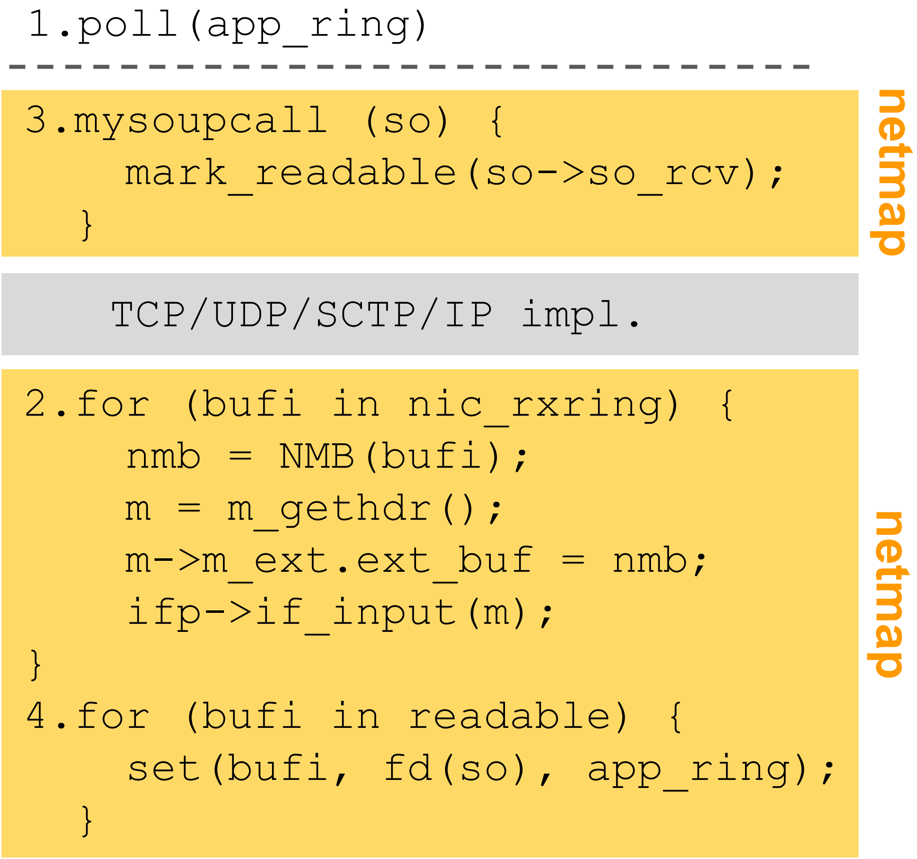
-



64B RPC-like workload

Netmap to the stack

- What's going on in poll()
  - I/O at the underlying NIC



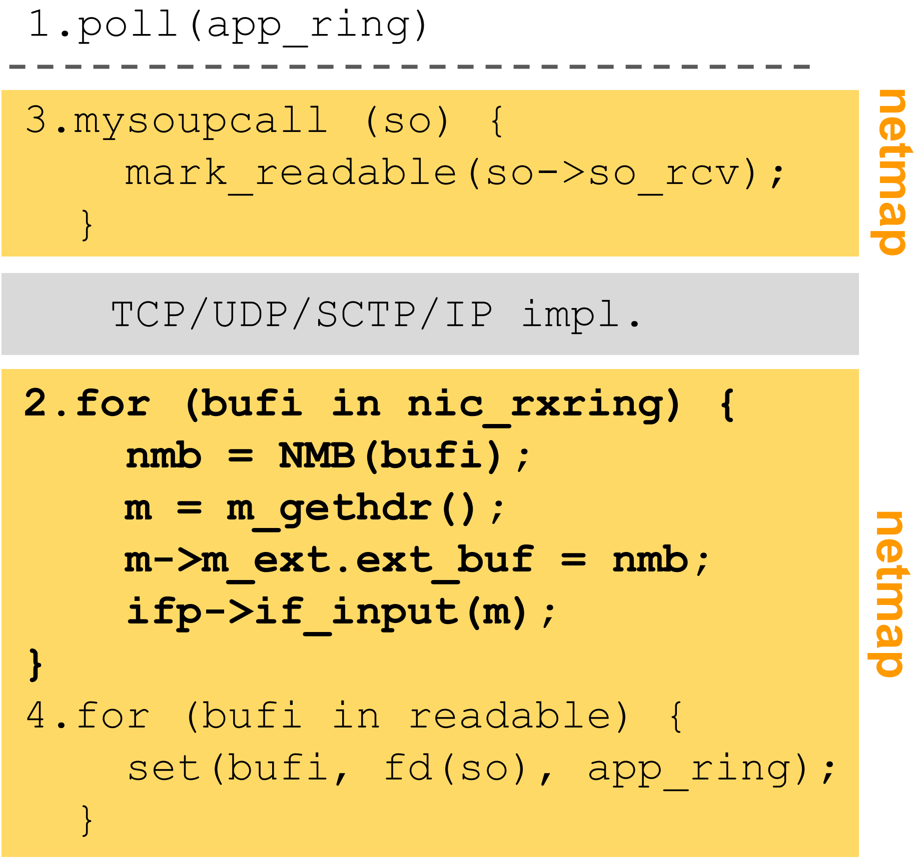
```
1.poll(app_ring)
------------------------------------
3.mysoupcall (so) {
      mark_readable(so->so_rcv);
   }

      TCP/UDP/SCTP/IP impl.

2.for (bufi in nic_rxring) {
      nmb = NMB(bufi);
      m = m_gethdr();
      m->m_ext.ext_buf = nmb;
      ifp->if_input(m);
}
4.for (bufi in readable) {
      set(bufi, fd(so), app_ring);
   }
```

Netmap to the stack

- What’s going on in poll()
    - I/O at the underlying NIC
    - **Push netmap packet buffers into the stack**



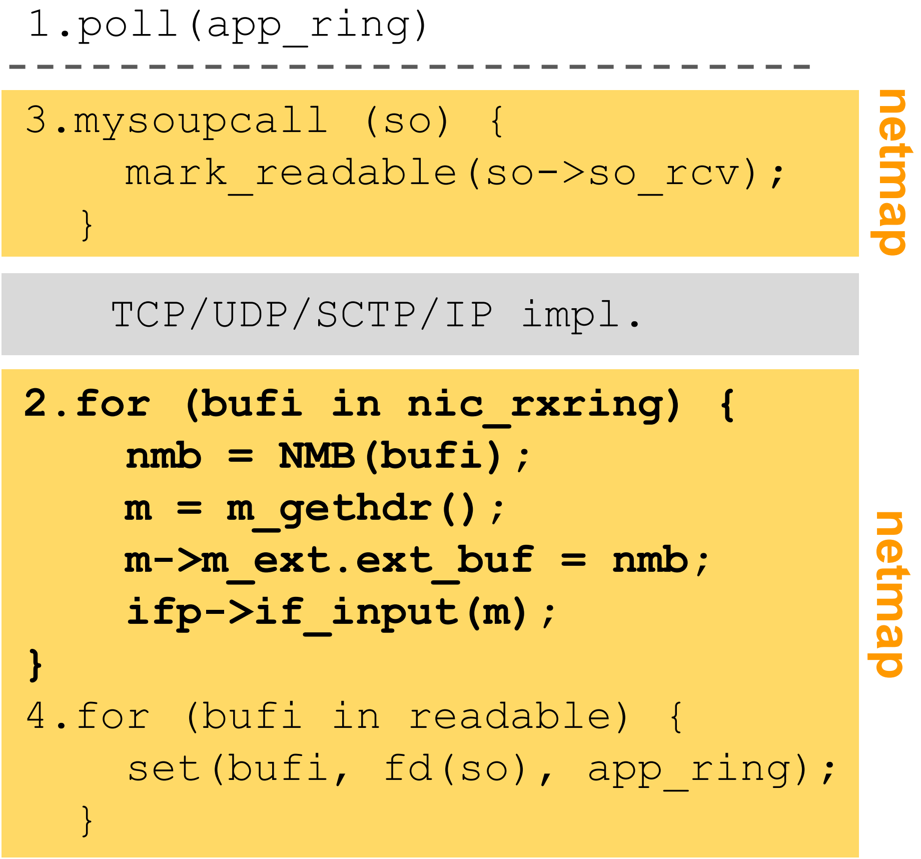
```
1.poll(app_ring)
------------------------------------
3.mysoupcall (so) {
     mark_readable(so->so_rcv);
  }

     TCP/UDP/SCTP/IP impl.

2.for (bufi in nic_rxring) {
     nmb = NMB(bufi);
     m = m_gethdr();
     m->m_ext.ext_buf = nmb;
     ifp->if_input(m);
}
4.for (bufi in readable) {
     set(bufi, fd(so), app_ring);
  }
```

Netmap to the stack

- What’s going on in poll()
  - I/O at the underlying NIC
  - **Push netmap packet buffers into the stack**
    - Have an mbuf *point* a netmap buffer
    - Then if_input()



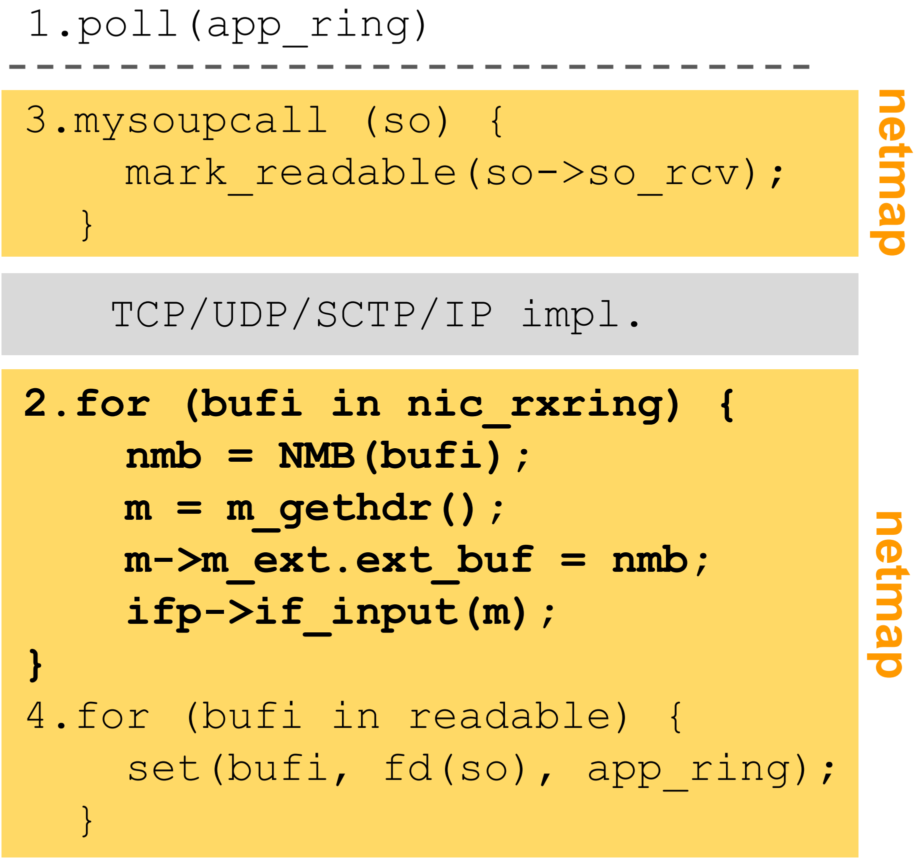
```
1.poll(app_ring)
---------------------------------
3.mysoupcall (so) {
     mark_readable(so->so_rcv);
  }

    TCP/UDP/SCTP/IP impl.

2.for (bufi in nic_rxring) {
     nmb = NMB(bufi);
     m = m_gethdr();
     m->m_ext.ext_buf = nmb;
     ifp->if_input(m);
}
4.for (bufi in readable) {
     set(bufi, fd(so), app_ring);
  }
```

Netmap to the stack

- What’s going on in poll()
  - I/O at the underlying NIC
  - **Push netmap packet buffers into the stack**
    - **Have an mbuf *point* a netmap buffer**
    - **Then if_input()**
    - **How to know what has happend to mbuf?**



```
 1.poll(app_ring)
--------------------------------
 3.mysoupcall (so) {
       mark_readable(so->so_rcv);
   }

     TCP/UDP/SCTP/IP impl.

 2.for (bufi in nic_rxring) {
       nmb = NMB(bufi);
       m = m_gethdr();
       m->m_ext.ext_buf = nmb;
       ifp->if_input(m);
 }
 4.for (bufi in readable) {
       set(bufi, fd(so), app_ring);
   }
```

Netmap to the stack

- **After if_input(), check the mbuf status**

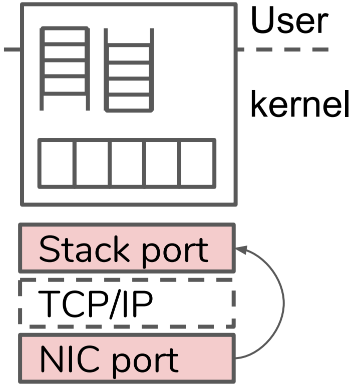
mbuf dtor	soupcall	Status	Example
Y	Y	App readable	In-order TCP segments
Y	N	Consumed	Pure acks
N	N	Held by the stack	Out-of-order TCP segments

Netmap to the stack

- **After if_input(), check the mbuf status**

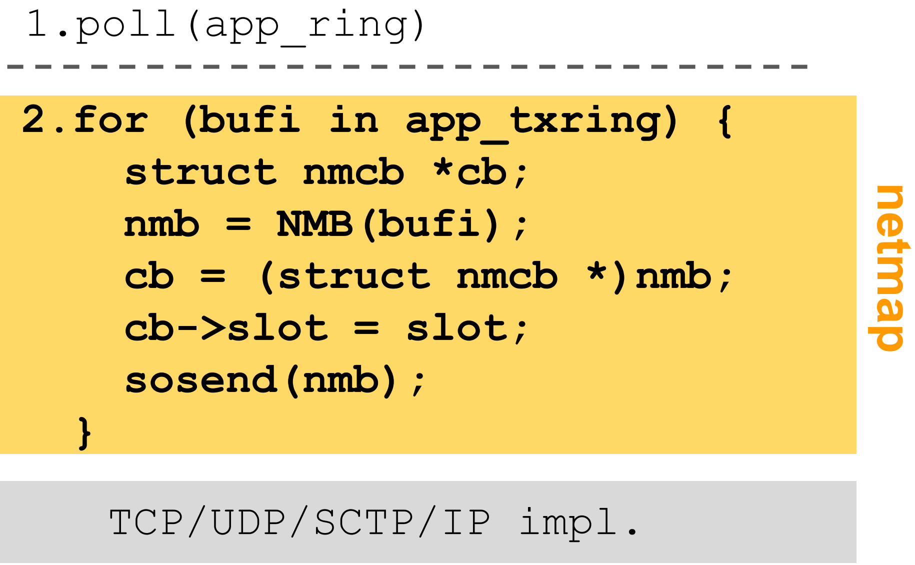
mbuf dtor	soupcall	Status	Example
Y	Y	App readable	In-order TCP segments
Y	N	Consumed	Pure acks
N	N	Held by the stack	Out-of-order TCP segments

- **Move App-readable packet to stack port (buffer index only, zero copy)**



Netmap to the stack (TX)

- What's going on in poll()
    - **Push netmap packet buffers into the stack**
        - **Embed netmap metadata to the buffer headroom**
        - **Then sosend()**



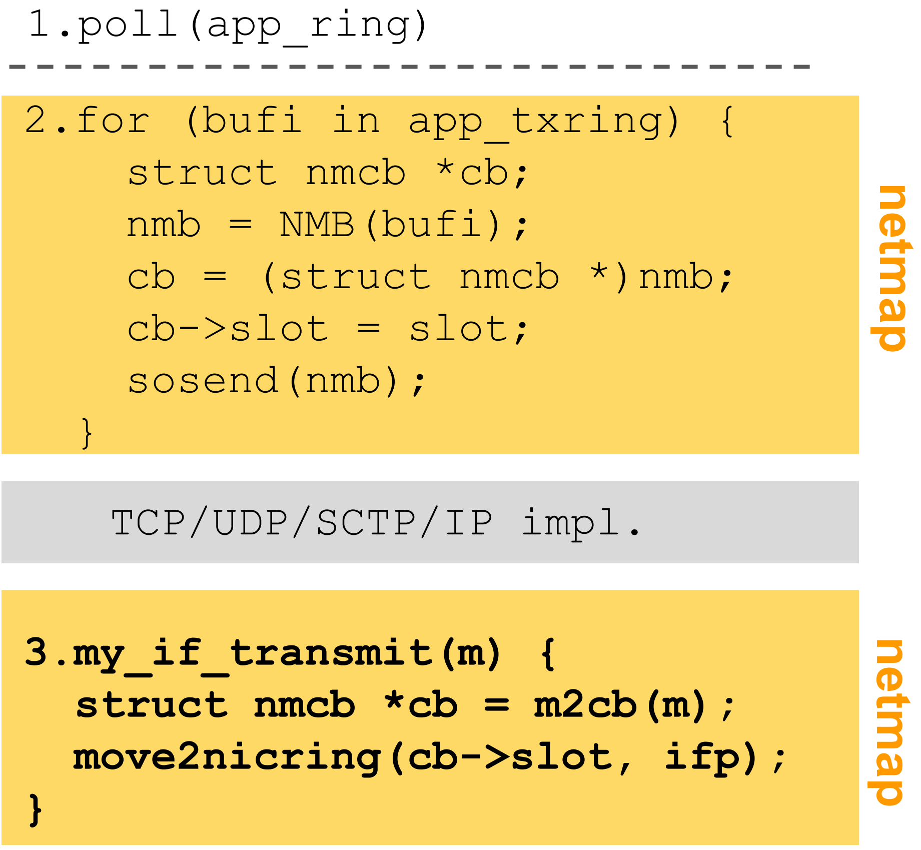
```
1.poll(app_ring)
```

```
2.for (bufi in app_txring) {
      struct nmcb *cb;
      nmb = NMB(bufi);
      cb = (struct nmcb *)nmb;
      cb->slot = slot;
      sosend(nmb);
  }
```

```
TCP/UDP/SCTP/IP impl.
```

Netmap to the stack (TX)

- What’s going on in poll()
  - **Push netmap packet buffers into the stack**
    - Embed netmap metadata to the buffer headroom
    - Then sosend()
    - **Catch mbuf at if_transmit()**
    - **NIC I/O happens after all the app rings have been processed (batched)**



```
1.poll(app_ring)
--------------------------------------
```

```
2.for (bufi in app_txring) {
      struct nmcb *cb;
      nmb = NMB(bufi);
      cb = (struct nmcb *)nmb;
      cb->slot = slot;
      sosend(nmb);
  }
```

```
    TCP/UDP/SCTP/IP impl.
```

```
3.my_if_transmit(m) {
   struct nmcb *cb = m2cb(m);
   move2nicring(cb->slot, ifp);
}
```

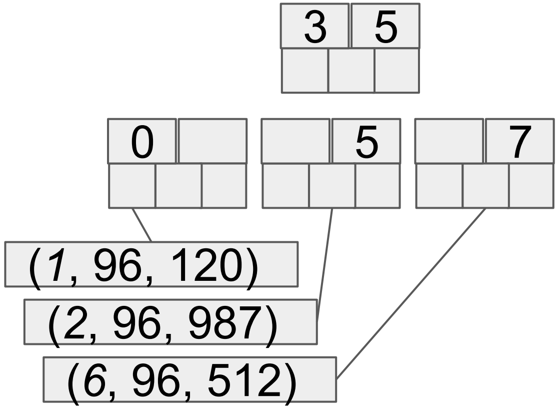
Persistent memory abstraction

- netmap is a good abstraction for storage stack

Write-Ahead Log

bufi	off	len
1	96	120
2	96	987
6	96	512

B+tree



Persistent memory abstraction

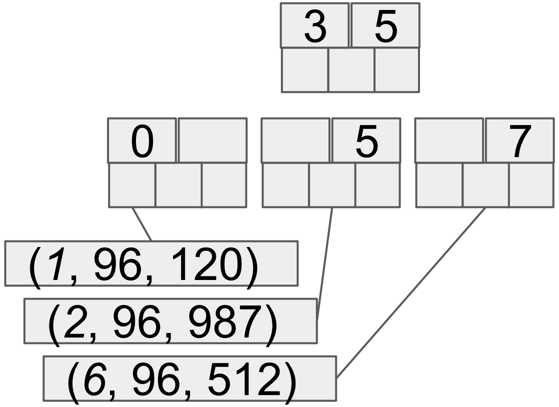
- netmap is a good abstraction for storage stack

Write-Ahead Log

bufi	off	len	csum
1	96	120	
2	96	987	
6	96	512	

From TCP header!

B+tree



Persistent memory abstraction

- netmap is a good abstraction for storage stack

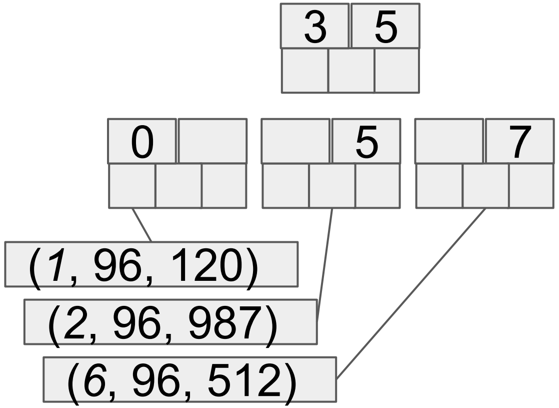
Write-Ahead Log

bufi	off	len	csum	time
1	96	120		
2	96	987		
6	96	512		

From TCP header!

From packet metadata
provided by NIC!

B+tree



Summary

- Convert end-host networking from disk to memory abstraction
- netmap can go beyond raw packet I/O
  - TCP/IP support
  - Persistent memory integration
- Status
  - https://micchie.net/paste
  - Working with netmap team to merge
  - Awaiting for FreeBSD supports for persistent memory