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);
}

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

2.8 Gbps
400 us
23 us
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

```c
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

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

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

![Diagram of netmap architecture]

- User
- Kernel
- Netmap buffers
- Netmap ports
- Backends
  - NIC port
  - Vale port
  - Pipe port
  - Stack port
  - NIC ring
  - Switch
  - Endpoint
  - TCP/IP
  - NIC port
- NIC’s memory model as abstraction
  - Efficient raw packet I/O

```c
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);
  }
```
netmap (4) w/ PASTE

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

<table>
<thead>
<tr>
<th>User</th>
<th>Netmap buffers</th>
<th>Netmap ports</th>
<th>Stack port</th>
<th>NIC port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel</td>
<td>netmap API (ring, slot, descriptor structures, poll() etc.)</td>
<td>NIC ring</td>
<td>Stack port</td>
<td>NIC port</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Vale port</td>
<td>Pipe port</td>
<td>TCP/IP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Switch</td>
<td>Endpoint</td>
<td></td>
</tr>
</tbody>
</table>

```c
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);
    } else 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

```c
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
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**

time

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

![Graph showing performance comparison between FreeBSD and PASTE with different connection counts and core configurations. The y-axis represents throughput in thousand transactions per second (1K trans/s), and the x-axis represents different connection counts and core configurations. The number on each bar indicates the mean latency (round trip) in microseconds (us).](image-url)
Netmap to the stack

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

```c
1. poll(app_ring)

3. mysoupcall (so) {
    mark_readable(so->so_rcv);
}

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);
}

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

2. for (bufi in nic_rxring) {
    nmb = NMB(bufi);
    m = m_gethdr();
    m->m_ext.ext_buf = nmb;
    ifp->if_input(m);
}

3. mysoupcall (so) {
    mark_readable(so->so_rcv);
}

4. for (bufi in readable) {
    set(bufi, fd(so), app_ring);
}
```

TCP/UDP/SCTP/IP impl.
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()

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

```c
3. mysoupcall (so) {
   mark_readable(so->so_rcv);
}
```

- TCP/UDP/SCTP/IP impl.

```c
2. for (bufi in nic_rxring) {
   nmb = NMB(bufi);
   m = m_gethdr();
   m->m_ext.ext_buf = nmb;
   ifp->if_input(m);
}
```

```c
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 happened to mbuf?

1. poll(app_ring)

3. mysoupcall (so) {
   mark_readable(so->so_rcv);
}

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

<table>
<thead>
<tr>
<th>mbuf dtor</th>
<th>soupcall</th>
<th>Status</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>Y</td>
<td>App readable</td>
<td>In-order TCP segments</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>Consumed</td>
<td>Pure acks</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>Held by the stack</td>
<td>Out-of-order TCP segments</td>
</tr>
</tbody>
</table>
Netmap to the stack

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

<table>
<thead>
<tr>
<th>mbuf dtor</th>
<th>soupcall</th>
<th>Status</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>Y</td>
<td>App readable</td>
<td>In-order TCP segments</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>Consumed</td>
<td>Pure acks</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>Held by the stack</td>
<td>Out-of-order TCP segments</td>
</tr>
</tbody>
</table>

- 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

<table>
<thead>
<tr>
<th>bufi</th>
<th>off</th>
<th>len</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>96</td>
<td>120</td>
</tr>
<tr>
<td>2</td>
<td>96</td>
<td>987</td>
</tr>
<tr>
<td>6</td>
<td>96</td>
<td>512</td>
</tr>
</tbody>
</table>

### B+tree

```
  3 5
  0 5 7
(1, 96, 120)
(2, 96, 987)
(6, 96, 512)
```
Persistent memory abstraction

- netmap is a good abstraction for storage stack

Write-Ahead Log

<table>
<thead>
<tr>
<th>bufi</th>
<th>off</th>
<th>len</th>
<th>csum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>96</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>96</td>
<td>987</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>96</td>
<td>512</td>
<td></td>
</tr>
</tbody>
</table>

From TCP header!

B+tree

- (1, 96, 120)
- (2, 96, 987)
- (6, 96, 512)
Persistent memory abstraction

- netmap is a good abstraction for storage stack

Write-Ahead Log

<table>
<thead>
<tr>
<th>bufi</th>
<th>off</th>
<th>len</th>
<th>csum</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>96</td>
<td>120</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>96</td>
<td>987</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>96</td>
<td>512</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

From TCP header!
From packet metadata provided by NIC!

B+tree

(1, 96, 120)
(2, 96, 987)
(6, 96, 512)

From packet metadata provided by NIC!
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](https://micchie.net/paste)
  - Working with netmap team to merge
  - Awaiting for FreeBSD supports for persistent memory