PASTE: Fast End System Networking with netmap

Michio Honda,
Giuseppe Lettieri, Lars Eggert and Douglas Santry
BSDCan 2018

Contact: @michioh, micchie@sfc.wide.ad.jp
Code: https://github.com/micchie/netmap/tree/stack
This talk is about:
What are problems with current network stack?
How do we solve it?
This talk is **NOT** about:

*User-space network stack is awesome*
Problem 1: Current socket API is slow

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
Problem 2: Current stack cannot utilize Non-Volatile Main Memory efficiently

- Review: NVMMs offer fast, byte-addressable persistence
Problem 2: Current stack cannot utilize Non-Volatile Main Memory efficiently

- Review: NVMMs offer fast, byte-addressable persistence
Problem 2: Current stack cannot utilize Non-Volatile Main Memory efficiently

Durable-write request (1400B) and response (64B) over HTTP and TCP

\[ n = \text{kevent}(\text{fds}) \]
\[ \text{for } (i=0; i<n; i++) \{ \]
\[ \quad \text{read}(\text{fds}[i], \text{buf}); \]
\[ \quad \ldots \]
\[ \quad \text{memcpy}(\text{nvmm}, \text{buf}); \]
\[ \quad \text{clflush}(\text{nvmm}); \]
\[ \quad \ldots \]
\[ \quad \text{write}(\text{fds}[i], \text{res}); \]
\[ \} \]

Server has Xeon 2640v4 2.4 Ghz
Client has Xeon 2690v4 2.6 Ghz and runs \textit{wrk} HTTP benchmark tool
Summary

- Per-socket system call and I/O req. must be avoided
- Data copy (even to NVMM) must be avoided
Getting architecture right

How do we address these problems while preserving benefits offered by the current stack and socket API today?
Scalable, flexible end system networking architecture

- True zero copy (even to NVMM)
- System call and I/O batching across multiple sockets
- Support for kernel TCP/IP
- Protocol independence
- Blocking and busy polling
- Protection

What we benefit from socket API today
PASTE building blocks

- Two netmap extensions:
  - **stack** port
    - integrates the kernel TCP/IP implementation
    - same level of abstraction with pipe and vale ports
  - **extmem** subsystem
    - supports arbitrary (user virtual address) memory region for netmap objects
    - `mmap()`-ed file in NVMM can be used
PASTE in Action

App thread

Shared memory region

slot cur [0] [7]
20 21 22 23 24 25 26 27

bufs

[0] [4] [8]

TCP/IP

NIC
PASTE in Action

1. Run NIC I/O and TCP/IP

- `poll()` triggers NIC I/O and TCP/IP processing
PASTE in Action

1. Run NIC I/O and TCP/IP

- Imagine 7 packets received
### PASTE in Action

1. Run NIC I/O and TCP/IP

- They are in-order TCP segments, so the kernel sets them to app ring slots
- Zero copy
  - swap with buffers in the current app ring
- Advance tail pointer to indicate new app data
PASTE in Action

1. Run NIC I/O and TCP/IP

- `poll()` returns
PASTE in Action

1. Run NIC I/O and TCP/IP
2. Read data on ring

App reads buffers in ring slots from `cur` to `tail`
PASTE in Action

1. Run NIC I/O and TCP/IP
2. Read data on ring
3. Update ring pointer

- App advances cur
  - Return buffers in slot 0-6 to the kernel at next poll()
    - Buffer indices are also 0-6 in this case
Zero copy write

- App does not have to return buffers to the kernel
  - e.g., useful for KVSes

App thread

1. Run NIC I/O and TCP/IP
2. Read data on ring
3. Record data
4. Swap out buf(s)
5. Update ring pointer

Shared memory region

- Slot [0]
- Slot [7]
- Slot [8]

B+tree

- [3, 5]
- [0, 5, 7]
- (1, 96, 120)
- (2, 96, 987)
- (6, 96, 512)

netmap API

slot [0]

[0]
[4]
[8]

bufs

TCP/IP

NIC
Durable zero copy write

1. Run NIC I/O and TCP/IP
2. Read data on ring
3. Flush buf(s)
4. Flush metadata
5. Swap out buf(s)
6. Update ring pointer

- `mmap()` a file on NVMM
  - `/mnt/pm/pp`
- Create netmap objects in it with extmem
- Create B+tree also in NVMM
How app code look like

```c
nmd = nm_open("stack:0");
ioctl(nmd->fd, "stack:em0");
s = socket();bind(s);listen(s);
int fds[2] = {nmd, s};
for (;;) {
poll(fds, 2,);
if (fds[1] & POLLIN)
ioctl(nmd,, accept(fds[1]));
if (fds[0] & POLLIN) {
    for (slot in nmd->rxring) {
        int fd = slot->fd;
        char *p = NETMAP_BUF(slot) + slot->offset;
    }
}
}
```

*use of extmem can be specified at nm_open()

What’s going on in poll()

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

netmap
PASTE performance

Single CPU core

Don’t store data

Store data in NVMM

Server has Xeon 2640v4 2.4 Ghz, Intel X540 10 GbE NIC and HPE NVDIMM
Client has Xeon 2690v4 2.6 Ghz and the same NIC, and runs wrk HTTP benchmark tool
PASTE performance

Multiple CPU cores

Server has Xeon 2640v4 2.4 Ghz, Intel X540 10 GbE NIC and HPE NVDIMM
Client has Xeon 2690v4 2.6 Ghz and the same NIC, and runs wrk HTTP benchmark tool
Redis

PASTE performance

Server has Xeon 2640v4 2.4 Ghz, Intel X540 10 GbE NIC and HPE NVDIMM
Client has Xeon 2690v4 2.6 Ghz and the same NIC
Changes needed in FreeBSD core

@@ -1101,6 +1101,8 @@
drop:
    if (so->so_proto->pr_usrreqs->pru_close != NULL)
        (*so->so_proto->pr_usrreqs->pru_close)(so);
+       if (so->so_dtor != NULL)
+               so->so_dtor(so);

SOCK_LOCK(so);
@@ -111,6 +111,7 @@
struct socket {

    int so_ts_clock;        /* type of the clock used for timestamps */
    uint32_t so_max_pacing_rate;    /* (f) TX rate limit in bytes/s */
+    void (*so_dtor)(struct socket *so); /* (a) optional destructor */
union {
    /* Regular (data flow) socket. */
Summary

- PASTE integrates the kernel TCP/IP implementations and emerging NVMM with netmap API
- Status
  - In the process of upstreaming to netmap (w/ Giuseppe Lettieri)

Academic paper:
Michio Honda, Giuseppe Lettieri, Lars Eggert and Douglas Santry, “PASTE: A Network Programming Interface for Non-Volatile Main Memory”, USENIX NSDI 2018

Code:
https://github.com/micchie/netmap/tree/stack

Contact:
@michioh, micchie@sfc.wide.ad.jp