Software Routers: NetMap

Hakim Weatherspoon
Assistant Professor, Dept of Computer Science
CS 5413: High Performance Systems and Networking
October 8, 2014

Goals for Today

• Gates Data Center and Fractus tour

• NetMap: A Novel Framework for Fast Packet I/O
Production-side and Experimental side

- 1 rack of 19 machines each
- Network
  - 1x HP 10/100 Mb management switch
  - 1x Top-of-Rack switch: Dell Force10 10 Gb Ethernet data switch
    - Contains a bonded pair of 40 Gb links to the research rack (80 Gb total)
- Servers
  - 2x 8-core Intel Xeon E5-2690 2.9 GHz CPUs (16 cores/32 threads total)
  - 96 GB RAM
  - A bonded pair of 10 Gb Ethernet links (20 Gb total)
  - 4x 900 GB 10k RPM SAS drives in RAID 0
- Multiple virtual machines on one physical machine
- Applications run unmodified as on real machine
- VM can migrate from one computer to another
Goals for Today

- Gates Data Center and Fractus tour

- NetMap: A Novel Framework for Fast Packet I/O
Direct Packet I/O Options

- Good old sockets (BPF, raw sockets)
  - Flexible, portable, but slow
Direct Packet I/O options

- Good old sockets (BPF, raw sockets)
  - Flexible, portable, but slow
  - Raw socket: *all* traffic from *all* NICs to user-space
  - Too general, hence complex network stack
  - Hardware and software are loosely coupled
  - Applications have no control over resources
Direct Packet I/O Options

• Good old sockets (BPF, raw sockets)
  – Flexible, portable, but slow

• Memory mapped buffers (PF_PACKET, PF_RING)
  – Efficient, if mbufs/skbuks do not get in the way
Direct Packet I/O Options

- Good old sockets (BPF, raw sockets)
  - Flexible, portable, but slow
- Memory mapped buffers (PF_PACKET, PF_RING)
  - Efficient, if mbufs/skbufs do not get in the way
Direct Packet I/O Options

- Good old sockets (BPF, raw sockets)
  - Flexible, portable, but slow
- Memory mapped buffers (PF_PACKET, PF_RING)
  - Efficient, if mbufs/skbufs do not get in the way
- Run in the kernel (NETFILTER, PFIL, Netgraph, NDIS, Click)
  - Can be fast, especially if bypassing mbufs
- Custom Libraries (OpenOnLoad, Intel DPDK)
  - Vendor specific: Normally tied to vendor hardware
- Can we find a better (fast, safe, HW-independent) solution?
### Traditional Network Stack

- How slow is the traditional raw socket and host network stack?

<table>
<thead>
<tr>
<th>File</th>
<th>Function/description</th>
<th>time (ns)</th>
<th>delta (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>user program</td>
<td>sendto, system call</td>
<td>8</td>
<td>96</td>
</tr>
<tr>
<td>uipec_syscalls.c</td>
<td>sys_sendto</td>
<td>104</td>
<td></td>
</tr>
<tr>
<td>uipec_syscalls.c</td>
<td>sendit</td>
<td>111</td>
<td></td>
</tr>
<tr>
<td>uipec_syscalls.c</td>
<td>kern_sendit</td>
<td>118</td>
<td></td>
</tr>
<tr>
<td>uipec_socket.c</td>
<td>sosend</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>uipec_socket.c</td>
<td>sosend_dgram, sockbuf locking, mbuf allocation, copyin</td>
<td>146</td>
<td>137</td>
</tr>
<tr>
<td>udp_usrreq.c</td>
<td>udp_send</td>
<td>273</td>
<td>57</td>
</tr>
<tr>
<td>udp_usrreq.c</td>
<td>udp_output</td>
<td>273</td>
<td></td>
</tr>
<tr>
<td>ip_output.c</td>
<td>ip_output, route lookup, ip header setup</td>
<td>330</td>
<td>198</td>
</tr>
<tr>
<td>if_ethersubr.c</td>
<td>ether_output, MAC header lookup and copy, loopback</td>
<td>528</td>
<td>162</td>
</tr>
<tr>
<td>if_ethersubr.c</td>
<td>ether_output_frame</td>
<td>690</td>
<td></td>
</tr>
<tr>
<td>ixgbe.c</td>
<td>ixgbe_mq_start</td>
<td>698</td>
<td>720</td>
</tr>
<tr>
<td>ixgbe.c</td>
<td>ixgbe_mq_start_locked</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ixgbe.c</td>
<td>ixgbe_xmit</td>
<td>730</td>
<td>220</td>
</tr>
<tr>
<td>–</td>
<td>on wire</td>
<td>950</td>
<td></td>
</tr>
</tbody>
</table>
Traditional Network Stack

Significant amount of time spent at all levels of the stack

• The system call cannot be avoided (or can it?)
• Device programming is extremely expensive
• Complex mbufs are a bad idea
• Data copies and mbufs can be saved in some cases
• Headers should be cached and reused if possible
Motivation for a new design

Design Guidelines

• No requirement/reliance on special hardware features
• Amortize costs over large batches (syscalls)
• Remove unnecessary work (copies, mbufs, alloc/free)
• Reduce runtime decisions (a single frame format)
• Modifying device drivers is permitted, as long as the code can be maintained
NetMap summary

Framework for raw packet I/O from userspace

• 65 cycles/packet between the wire in userspace
  – 14Mpps on one 900 MHz core
• Device and OS independent
• Good scalability with number of CPU frequency and number of cores
• libpcap emulation library for easy porting of applications
packet buffers
  - Numbered and fixed size

netmap rings
  - Device independent copy of NIC ring
cur: current tx/rx position
avail: available slots
  Ptrs stored as offsets or indexes (so not dependent of virtual mem)

netmap_if
  - Contains references to all rings attached to an interface
Protection

Rely on standard OS mechanism

• The NIC is not exposed to the userspace
• Kernel validates the netmap ring before using its contents
• Cannot crash the kernel or trash other processes memory
Data Ownership

• No races between kernel and user
• Global fields, and \([ \text{cur}\ldots\text{cur+avail}-1 ]\):
  – Owned by the application, updated during system calls
• other slots/buffers
  – Owned by the kernel
• Interrupt handler never touches shared memory regions
NetMap API

• Access
  – open: returns a file descriptor (fd)
  – ioctl: puts an interface into netmap mode
  – mmap: maps buffers and rings into user address space

• Transmit (TX)
  – Fill up to \texttt{avail} bufs, starting at slot \texttt{cur}
  – \texttt{ioctl(fd, NIOCTXSYNC)} queues the packets

• Receive (RX)
  – \texttt{ioctl(fd, NIOCTXSYNC)} reports newly received packets
  – Process up to \texttt{avail} bufs, starting at \texttt{cur}

• poll()/select()
NetMap API

• Access
  – open: returns a file descriptor (fd)
  – ioctl: puts an interface into netmap mode
  – mmap: maps buffers and rings

• Transmit (TX)
  – Fill up to avail bufs, starting at cur
  – ioctl(fd, NIOCTXSYNC) queues

• Receive (RX)
  – ioctl(fd, NIOCTXSYNC) reports
  – Process up to avail bufs, starting cur

• poll()/select()
Multiqueue/multicore support

• One netmap ring per physical NIC ring
• By default, the **fd** is bound to all NIC rings, but
  – ioctl can restrict the binding to a single NIC ring pair
  – multiple **fd**’s can be bound to different rings on the same card
  – The **fd**’s can be managed by different threads
  – Threads mapped to cores with **pthread_setaffinity()**
Netmap Mode

• Data path
  – Normal path between NIC and host stack is removed

• Control path
  – OS believes NIC is still there
  – Ifconfig, ioctl, etc still work
NetMap and Host Stack

Netmap Mode

• Data path
  – Packets from NIC end up in netmap ring
  – Packets from TX netmap ring are sent to the NIC

• Control path
  – OS believes NIC is still there
  – Ifconfig, ioctl, etc still work
Zero copy

- Zero-copy forwarding by swapping buffer indexes

```c
...  
src = &src_nifp->slot[i]; /* locate src and dst slots */
dst = &dst_nifp->slot[j];
/* swap the buffers */
tmp = dst->buf_index;
dst->buf_index = src->buf_index;
src->buf_index = tmp;
/* update length and flags */
dst->len = src->len;
/* tell kernel to update addresses in the NIC rings */
dst->flags = src->flags = BUF_CHANGED;
...
```

- Zero-copy also works with rings from/to host stack
  - Firewalls, NAT boxes, IDS mechanisms
Performance Results

- TX tput vs clock speed and number of cores
Performance Results

- TX tput vs burst size
Performance Results

• RX tput vs packet size
• Forwarding performance

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Mpps</th>
</tr>
</thead>
<tbody>
<tr>
<td>netmap-fwd (1.733 GHz)</td>
<td>14.88</td>
</tr>
<tr>
<td>netmap-fwd + pcap</td>
<td>7.50</td>
</tr>
<tr>
<td>click-fwd + netmap</td>
<td>3.95</td>
</tr>
<tr>
<td>click-etherswitch + netmap</td>
<td>3.10</td>
</tr>
<tr>
<td>click-fwd + native pcap</td>
<td>0.49</td>
</tr>
<tr>
<td>openvswitch + netmap</td>
<td>3.00</td>
</tr>
<tr>
<td>openvswitch + native pcap</td>
<td>0.78</td>
</tr>
<tr>
<td>bsd-bridge</td>
<td>0.75</td>
</tr>
</tbody>
</table>
Framework for raw packet I/O from userspace

- 65 cycles/packet between the wire in userspace
  - 14Mpps on one 900 MHz core
- Device and OS independent
- Good scalability with number of CPU frequency and number of cores
- `libpcap` emulation library for easy porting of applications
Before Next time

• Project Progress
  – Need to setup environment as soon as possible
  – And meet with groups, TA, and professor
• Lab3 – Packet filter/sniffer
  – Due Thursday, October 16
  – Use Fractus instead of Red Cloud

• **Required review and reading for Friday, October 15**
  – http://dl.acm.org/citation.cfm?id=2396563

• Check piazza: http://piazza.com/cornell/fall2014/cs5413
• Check website for updated schedule