Software Routers: NetSlice

Hakim Weatherspoon
Assistant Professor, Dept of Computer Science
CS 5413: High Performance Systems and Networking
October 15, 2014

Goals for Today

- NetSlices: Scalable Multi-Core Packet Processing in User-Space
Packet Processors

• Essential for evolving networks
  – Sophisticated functionality
  – Complex performance enhancement protocols
Packet Processors

• Essential for evolving networks
  – Sophisticated functionality
  – Complex performance enhancement protocols

• Challenges: *High-performance* and *flexibility*
  – 10GE and beyond
  – Tradeoffs
Software Packet Processors

• Low-level (kernel) vs. High-level (userspace)

• *Parallelism* in userspace: Four major difficulties
  – Overheads & Contention
  – Kernel network stack
  – Lack of control over hardware resources
  – Portability
Overheads & Contention

- Cache coherence
- Memory Wall
- Slow cores vs. Fast NICs
• 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
Portability

• Hardware dependencies
• Kernel and device driver modifications
  – Zero-copy
  – Kernel bypass
Outline

• Difficulties in building packet processors
• NetSlice
• Evaluation
• Discussions
• Conclusion
NetSlice

• Give power to the application
  – Overheads & Contention
  – Lack of control over hardware resources
    • Spatial partitioning exploiting NUMA architecture
– Kernel network stack
  • Streamlined path for packets
– Portability
  • No zero-copy, kernel & device driver modifications
NetSlice Spatial Partitioning

• Independent (parallel) execution contexts
  – Split each Network Interface Controller (NIC)
    • One NIC queue per NIC per context
  – Group and split the CPU cores
  – Implicit resources (bus and memory bandwidth)
NetSlice Spatial Partitioning Example

- **2x quad core Intel Xeon X5570 (Nehalem)**
  - Two simultaneous hyperthreads – OS sees 16 CPUs
  - Non Uniform Memory Access (NUMA)
- QuickPath point-to-point interconnect
  - Shared L3 cache
Streamlined Path for Packets

- Inefficient conventional network stack
  - One network stack “to rule them all”
  - Performs too many memory accesses
  - Pollutes cache, context switches, synchronization, system calls, blocking API
Portability

- No zero-copy
  - Tradeoffs between portability and performance
  - NetSlices achieves both
- No hardware dependency
- A run-time loadable kernel module
NetSlice API

- Expresses fine-grained hardware control
- Flexible: based on ioctl
- Easy: open, read, write, close

```c
#include "netslice.h"

struct netslice_rwlock_multi {
    int flags;
} rw_multi;

struct netslice_cpu_mask {
    cpu_set_t k_peer, u_peer;
} mask;

fd = open("/dev/netslice-1", O_RDWR);

rw_multi.flags = MULTI_READ | MULTI_WRITE;
ioctl(fd, NETSLICE_RW_MULTI_SET, &rw_multi);
ioctl(fd, NETSLICE_CPUMASK_GET, &mask);
sched_setaffinity(getpid(), sizeof(cpu_set_t), &mask.u_peer);

for (;;) {
    ssize_t cnt, wcnt = 0;
    if ((cnt = read(fd, iov, IOVS)) < 0)
        EXIT_FAIL_MSG("read");
    for (i = 0; i<cnt; i++)
        /* iov_rlen marks bytes read */
        scan_pkg(iov[i].iov_base, iov[i].iov_rlen);
    do {
        size_t wr_iovs;
        /* write iov_rlen bytes */
        wr_iovs = write(fd, &iov[wcnt], cnt-wcnt);
        if (wr_iovs< 0)
            EXIT_FAIL_MSG("write");
        wcnt += wr_iovs;
    } while (wcnt<cnt);
} 
```
NetSlice Evaluation

• Compare against state-of-the-art
  – RouteBricks in-kernel, Click & pcap-mmap user-space

• Additional baseline scenario
  – All traffic through single NIC queue (receive-livelock)

• What is the basic forwarding performance?
  – How efficient is the streamlining of one NetSlice?

• How is NetSlice scaling with the number of cores?
Experimental Setup

- **R710 packet processors**
  - dual socket quad core 2.93GHz Xeon X5570 (Nehalem)
  - 8MB of shared L3 cache and 12GB of RAM
    - 6GB connected to each of the two CPU sockets
    - Two Myri-10G NICs

- **R900 client end-hosts**
  - four socket 2.40GHz Xeon E7330 (Penryn)
  - 6MB of L2 cache and 32GB of RAM
Simple Packet Routing

- End-to-end throughput, MTU (1500 byte) packets
Linear Scaling with CPUs

- IPsec with 128 bit key—typically used by VPN
  - AES encryption in Cipher-block Chaining mode
Outline

• Difficulties in building packet processors
• Netslice
• Evaluation
• Discussions
• Conclusion
Software Packet Processors

• Can support 10GE and more at line-speed
  – Batching
    • Hardware, device driver, cross-domain batching
  – Hardware support
    • Multi-queue, multi-core, NUMA, GPU
  – Removing IRQ overhead
  – Removing memory overhead
    • Including zero-copy
  – Bypassing kernel network stack
## Software Packet Processors

<table>
<thead>
<tr>
<th></th>
<th>Batching</th>
<th>Parallelism</th>
<th>Zero-Copy</th>
<th>Portability</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw socket</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
<td>✔️</td>
<td>User</td>
</tr>
<tr>
<td>RouteBricks</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
<td>Kernel</td>
</tr>
<tr>
<td>PacketShader</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
<td>User</td>
</tr>
<tr>
<td>PF_RING</td>
<td>✗</td>
<td>✔️</td>
<td>✗</td>
<td>✔️</td>
<td>User</td>
</tr>
<tr>
<td>netmap</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>User</td>
</tr>
<tr>
<td>Kernel-bypass</td>
<td>✗</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>User</td>
</tr>
<tr>
<td>NetSlice</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>✔️</td>
<td>User</td>
</tr>
</tbody>
</table>
# Software Packet Processors

<table>
<thead>
<tr>
<th></th>
<th>Batching</th>
<th>Multi-queue</th>
<th>Zero-Copy</th>
<th>Portability</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw socket</td>
<td>✔️</td>
<td>⚠️</td>
<td>⚠️</td>
<td>✔️</td>
<td>User</td>
</tr>
<tr>
<td>RouteBricks</td>
<td>✔️</td>
<td>✔️</td>
<td>⚠️</td>
<td>⚠️</td>
<td>Kernel</td>
</tr>
<tr>
<td>PacketShader</td>
<td>✔️</td>
<td>✔️</td>
<td>⚠️</td>
<td>⚠️</td>
<td>User</td>
</tr>
<tr>
<td>PF_RING</td>
<td>⚠️</td>
<td>✔️</td>
<td>⚠️</td>
<td>✔️</td>
<td>User</td>
</tr>
<tr>
<td>netmap</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>⚠️</td>
<td>User</td>
</tr>
<tr>
<td>Kernel-bypass</td>
<td>⚠️</td>
<td>✔️</td>
<td>✔️</td>
<td>⚠️</td>
<td>User</td>
</tr>
<tr>
<td>NetSlice</td>
<td>✔️</td>
<td>✔️</td>
<td>⚠️</td>
<td>✔️</td>
<td>User</td>
</tr>
</tbody>
</table>
## Software Packet Processors

<table>
<thead>
<tr>
<th></th>
<th>Batching</th>
<th>Multi-queue</th>
<th>Zero-Copy</th>
<th>Portability</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw socket</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
<td>✔️</td>
<td>User</td>
</tr>
<tr>
<td>RouteBricks</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
<td>Kernel</td>
</tr>
<tr>
<td>PacketShader</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
<td>User</td>
</tr>
<tr>
<td>PF_RING</td>
<td>✗</td>
<td>✔️</td>
<td>✗</td>
<td>✔️</td>
<td>User</td>
</tr>
<tr>
<td>netmap</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>User</td>
</tr>
<tr>
<td>Kernel-bypass</td>
<td>✗</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>User</td>
</tr>
<tr>
<td>NetSlice</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>✔️</td>
<td>User</td>
</tr>
</tbody>
</table>
### Software Packet Processors

<table>
<thead>
<tr>
<th></th>
<th>Batching</th>
<th>Multi-queue</th>
<th>Zero-Copy</th>
<th>Portability</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw socket</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
<td>✔️</td>
<td>User</td>
</tr>
<tr>
<td>RouteBricks</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
<td>Kernel</td>
</tr>
<tr>
<td>PacketShader</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
<td>User</td>
</tr>
<tr>
<td>PF_RING</td>
<td>✗</td>
<td>✔️</td>
<td>✗</td>
<td>✔️</td>
<td>User</td>
</tr>
<tr>
<td>netmap</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>User</td>
</tr>
<tr>
<td>Kernel-bypass</td>
<td>✗</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>User</td>
</tr>
<tr>
<td>NetSlice</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>✔️</td>
<td>User</td>
</tr>
<tr>
<td>Domain</td>
<td>Batching</td>
<td>Parallelism</td>
<td>Zero-Copy</td>
<td>Portability</td>
<td>Domain</td>
</tr>
<tr>
<td>----------------</td>
<td>----------</td>
<td>-------------</td>
<td>-----------</td>
<td>-------------</td>
<td>--------</td>
</tr>
<tr>
<td>User</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>User</td>
</tr>
<tr>
<td>Kernel</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>Kernel</td>
</tr>
<tr>
<td>User</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>User</td>
</tr>
<tr>
<td>User</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>User</td>
</tr>
<tr>
<td>User</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>User</td>
</tr>
<tr>
<td>User</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>User</td>
</tr>
<tr>
<td>User</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>User</td>
</tr>
<tr>
<td>User</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>User</td>
</tr>
</tbody>
</table>
## Software Packet Processors

<table>
<thead>
<tr>
<th></th>
<th>Batching</th>
<th>Parallelism</th>
<th>Zero-Copy</th>
<th>Portability</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw socket</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>User</td>
</tr>
<tr>
<td>RouteBricks</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>Kernel</td>
</tr>
<tr>
<td>PacketShader</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>User</td>
</tr>
<tr>
<td><strong>PF_RING</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>netmap</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>User</td>
</tr>
<tr>
<td>Kernel-bypass</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>User</td>
</tr>
<tr>
<td>NetSlice</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>User</td>
</tr>
</tbody>
</table>

- **Optimized for RX path only**
## Software Packet Processors

<table>
<thead>
<tr>
<th>Software</th>
<th>Batching</th>
<th>Parallelism</th>
<th>Zero-Copy</th>
<th>Portability</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw socket</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>User</td>
</tr>
<tr>
<td>RouteBricks</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>Kernel</td>
</tr>
<tr>
<td>PacketShader</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>User</td>
</tr>
<tr>
<td>PF_RING</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>User</td>
</tr>
<tr>
<td>netmap</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>User</td>
</tr>
<tr>
<td>Kernel-bypass</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>User</td>
</tr>
<tr>
<td>NetSlice</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>User</td>
</tr>
</tbody>
</table>
Discussions

• 40G and beyond
  – DPI, FEC, DEDUP, ...

• Deterministic RSS

• Small packets
Conclusion

• NetSlices: A new abstraction
  – OS support to build packet processing applications
  – Harness implicit parallelism of modern hardware to scale
  – Highly portable

• Webpage: http://netslice.cs.cornell.edu
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