High Capacity Network Link Emulation Using Network Processors

Masters Thesis Proposal
Abhijeet Joglekar
Outline of the Talk

- Background
  - Using Network Processors for Network Link Emulation
  - Preliminary Work
  - Proposed Work and Evaluation
- Conclusion
- Schedule
Research Domain

- **Network Emulation**
  - Real applications and protocols are subjected to a synthetic network environment.
  - Involves emulation of routers, *links*, end-nodes, traffic sources/sinks.
  - Example Emulation Environments: Emulab, Modelnet.

- Focus in this proposal is on *network link emulation*.

- Challenges:
  - Emulation runs in real-time, so number of emulated links depends on the packet forwarding capacity. More link emulation features imply more processing overhead thus reducing number of emulated links. Not a problem in simulation, since no requirement of keeping up with real-time.
Link Emulation Aspects/Metrics

- **Forwarding capacity** - Is largely independent of the packet size.

- **Link emulation capacity** - Measures the number of physical links and number of virtual links that can be multiplexed onto physical links.

- **Accuracy** - Depends on timer granularity, emulation load.

- **Extensibility** - How easy it is to plug in new delay/loss models. A component based emulation environment might support extensibility.

- **Feature Set** - Delay/loss models, bandwidth, packet loss, queuing discipline.
Current Solutions

- PC based link emulators

  - **User-space:**
    - **NSE link emulation**
      Link emulation runs in a single user-level process, packets captured from the network, injected into the emulator, and then sent out using raw IP interface.

  - **In-kernel:**
    - **Dummynet, NistNet, ONE**
      Act as a plug-in into the routing/bridging code in the kernel, transparent to the applications.

**Merits**
Well understood programming environment, easy to set up and deploy in a testbed.

**Limitations**
Link emulation capacity limited by a PCs forwarding capacity.
Outline of the Talk

- Background
  - Using Network Processors for Network Link Emulation
- Preliminary Work
- Proposed Work and Evaluation
- Conclusion
- Schedule
A network processor provides a higher capacity software network link emulation platform than a PC.
Network Processors

- Emerging field of programmable processors optimized for data plane packet processing functions.

- Occupy a middle ground between PCs and ASICs in both programmability and performance. Value is support for adding new network services in software while maintaining high performance.

- Typically use parallelism, special hardware units, and instruction sets tuned to networking applications.

- High packet forwarding capacity with headroom for application specific processing.
Example Network Processor: IXP1200

Fig: IXP1200 Architecture
IXP1200 – Features ....

- Hardware multi-threaded microengines, separate program counters and registers for each thread enable low-overhead context switching.

- Threads can hide memory/IO latencies by swapping to let other threads run or starting off asynchronous memory/IO operations and then doing other useful work.

- Non-preemptive context scheduler on microengines and global registers aid low-overhead intra-microengine synchronization.

- On-chip Scratchpad memory supports atomic bit-test-set/bit-test-clear operations for inter-microengine synchronization.
IXP1200 – Features

- SRAM memory locks help in intra-microengine synchronization, SRAM LIFO buffer queues provide low synchronization overhead O(1) buffer management scheme.

- Hardware hash unit computes 48 bit and 64 bit hashes, up to 3 hashes per instruction.

To summarize:

A highly parallel and optimized architecture which enables high packet forwarding rate.
Validation of Thesis

- Designing and implementing a network link emulator on the IXP1200 network processor.
- Validating its correctness.
- Measuring packet forwarding rate for both small and large packets and capacity for virtual link multiplexing.
- Comparing with Dummynet implementation on a PC.
Outline of the Talk

- Background
- Using Network Processors for Network Link Emulation
  - Preliminary Work
- Proposed Work and Evaluation
- Conclusion
- Schedule
Preliminary Work

Consisted of two parts:

- PC dummynet’s capacity measurements to identify the bottlenecks in a PC implementation.

- Prototype implementation on the IXP1200 evaluation board with Princeton VERA firmware to support the performance/higher capacity hypothesis.
Dummynet

- Implemented in FreeBSD.

- `ipfw` is the front-end program used to add rules to the IP firewall to pass packets to Dummynet.

- Link is abstracted by a pipe which has a bandwidth, delay, loss rate, queue size.

- A pipe is implemented using two queues: packets are inserted first into a bandwidth queue, removed at a fixed rate into a delay queue where propagation delay is emulated.

- Heap is used for pipe scheduling, key is the departure time of the first packet in the pipe.
Dummynet: Two Queue Model

Diagram showing the relationship between application, protocol stack, network, and queues (R-queue and P-queue).
Experiment 1

- 850 MHz PC, 32/33 PCI, four Intel EtherExpress 100 Mb interfaces.
- Polling enabled for all cards.
- Each of the four interfaces set up on a different LAN, with traffic sources and sinks.
- Dummynet pipes configured to 10ms delay, infinite bandwidth, no packet loss.

<table>
<thead>
<tr>
<th>Pkt Size (bytes)</th>
<th>No. of ports</th>
<th>Pkt Rate Per Port (Kpps)</th>
<th>Total Pkt. Rate (Kpps)</th>
<th>BW (Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>1</td>
<td>89</td>
<td>89</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>52.5</td>
<td>105</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>23.5</td>
<td>94</td>
<td>48</td>
</tr>
<tr>
<td>1518</td>
<td>1</td>
<td>8.1</td>
<td>8.1</td>
<td>98</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>8.1</td>
<td>16.2</td>
<td>196</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>8.1</td>
<td>24.3</td>
<td>295</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>6.9</td>
<td>26.6</td>
<td>323</td>
</tr>
</tbody>
</table>

32/33 PCI bus (80 MB- 100 MB data bandwidth) appears to be the bottleneck for large packets in the four port test.
Experiment 2

- A 2 GHz PC, 32/33 PCI, two National Semiconductor gigabit interfaces.
- Polling drivers for the interfaces.
- 180 Kpps (121 Mbps) for 64 byte packets.
- 26 Kpps (320 Mbps) for max. sized packets.

Even with better CPU, capacity for large packets is same as experiment 1, which suggests that 32/33 PCI is the bottleneck.

Experiment 3

Luigi (Dummynet’s author) reported
- 1 GHz machine, 64/66 PCI, two Intel Pro gigabit interfaces
- Polling enabled for interfaces
- 170 Kpps (114 Mbps) for 64 byte packets.
- Line rate or 81 Kpps (1 Gbps) for max size packets.

Since PCI is not the bottleneck, we get an idea of CPU bottleneck for Dummynet emulation in this experiment.
Prototype Link Emulator

- **Hardware platform**
  - IXP1200 evaluation board, running at 166 MHz, 32 MB SDRAM, 2 MB SRAM.

- **Software platform**
  - Princeton firmware and an early version of Princeton’s VERA (Virtual extensible router architecture) software model. (SOSP ’01)

- Added emulation code which supports link delay and bandwidth emulation for 4 ports, with one pipe per pair of interfaces (4 pipes)
Emulation & Forwarding Pipeline
Emulation Cost

- Without emulation, forwarding rate was 274.96 Kpps across 4 ports.
- With emulation stage added to the pipeline, rate was 274.76 Kpps, showing a negligible slowdown of 2.6 ns per packet. (4 pipes configured to 5 ms delay and infinite bandwidth).

Emulation Accuracy

<table>
<thead>
<tr>
<th>Link Delay (ms)</th>
<th>Observed RTT (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.181</td>
</tr>
<tr>
<td>1</td>
<td>2.24</td>
</tr>
<tr>
<td>5</td>
<td>10.50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Link Bandwidth (Mbps)</th>
<th>Observed Bandwidth (Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10.0673</td>
</tr>
<tr>
<td>15</td>
<td>15.1024</td>
</tr>
</tbody>
</table>
Limitations of the Prototype

- Does not work well for all packet sizes.

- For large packets, MAC reports overflows and corrupted packets.

- Communicated with Princeton team, they were facing same problem.

- Since we want to deploy this in Emulab, this problem has to be fixed. Decided to switch to Intel’s firmware and software model. Initial tests do not show this problem.
Outline of the Talk

- Background
- Using Network Processors for Network Link Emulation
- Preliminary Work
  ✓ Proposed Work and Evaluation
- Conclusion
- Schedule
Bridalveil IXP1200 Board

- Runs at 232 MHz, has 4 100 Mb ports, 256 MB SDRAM, 8 MB SRAM. About 192 MB of SDRAM is enough to delay 480Kpps for 200 ms.

- Set up as a PCI card in an Emulab PC.

- Linux and GNU tool chain on the StrongArm, familiar programming environment.

- ACE (Active Computing Element) software component model on StrongArm and microengines.

- Currently ACE supports use of only 128 MB SDRAM and 4 MB SRAM. Modified some kernel ACE drivers to support all memory on the board.
Design & Implementation of Link Emulator...

- A Dummynet-like implementation within the ACE framework on the Bridalveil board.

- The emulation code will be a plug-in into Ethernet bridging reference design provided by Intel.

- Bridging reference uses
  - 1 microengine (4 threads) for receive and bridging components.
  - 1 microengine (4 threads) for transmit.

- Thus, 4 microengines at our disposal.
Emulation code consists of:
- Classification step added to receive pipeline to classify packets into flows (different link characteristics).
- A separate emulation pipeline running on a different microengine.

Bridge reference code measurements:
- 464 Kpps (313 Mbps) across 4 ports for 64 byte packets.
- 32 Kpps (400 Mbps) across 4 ports for max. sized packets.

Extra computing power in idle microengines can be used for emulation, and thus we expect that packet rate will not reduce much with emulation (also suggested by prototype implementation).
Features

- Delay, bandwidth, packet loss, FIFO queuing.

- Virtual link multiplexing to support multiple flows/virtual nodes sending packets over the same physical link.

- Prototype supported delay and bandwidth emulation for fixed number of pipes.

- Additional emulation code in the final implementation:
  - Lookup cache for active flows, hash based classification on 4 tuples. Rare misses in the cache invoke Longest Prefix Match.
  - Pipe scheduling using heap.
  - Single physical delay + bandwidth queue to avoid copying packet descriptors between queues.
  - Packet loss rate emulation.
Validation and Capacity Measurements

- Validating the features of the link emulator: delay, bandwidth, packet loss rate, FIFO queuing.
- Measuring packet forwarding rate for a range of packet sizes.
- Measuring virtual link multiplexing capacity.
Conclusion

We show that network processors provide a higher capacity link emulation platform than a PC by:

- Implementing a Dummynet-like link emulator on the IXP1200 network processor.
- Measuring the capacity of the emulator and comparing with a Dummynet implementation on a PC.
<table>
<thead>
<tr>
<th>Task</th>
<th>Duration</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design, Implementation and basic testing</td>
<td>6 weeks</td>
<td>April 15</td>
</tr>
<tr>
<td>Validation, capacity measurements</td>
<td>4 weeks</td>
<td>May 15</td>
</tr>
<tr>
<td>Writing first thesis draft and submitting to Jay</td>
<td>2-3 weeks</td>
<td>June 5</td>
</tr>
<tr>
<td>2 rounds of feedback + changes (Jay)</td>
<td>5-6 weeks</td>
<td>July 15</td>
</tr>
<tr>
<td>Feedback from committee + changes</td>
<td>2-3 weeks</td>
<td>Aug 7</td>
</tr>
<tr>
<td>Thesis slides and defense</td>
<td>1 week</td>
<td>Aug 15</td>
</tr>
</tbody>
</table>