A Scalable, Accurate and Extensible Network Emulation Platform using the IXP1200 Network Processor

Abhijeet Joglekar (abhijeet@cs.utah.edu)
University of Utah
IXA Student Design Competition Proposal

Abstract

Emulab, the Utah Network testbed, is a large-scale emulation environment available to researchers all over the world. Through automated experiment configuration, and features such as dynamically changing link, router, and traffic characteristics, Emulab brings much of a simulator's *ease of use* and *control* to emulation while retaining its *realism*. PCs serve many roles in the testbed: as end-hosts in an emulated distributed system, as traffic generation and consumption nodes, as routers in the emulated topology, and as link emulator nodes. As link emulators, their capacity is limited to 1 or 2 100 Mb ethernet lines per box, largely due to PCI bus arbitration overhead which negatively impacts scaling to large topologies.

In this proposal, we address the problem of scalability, and accuracy under high loads, of link and router emulation by using the IXP series of network processors as an emulation platform. The IXP's highly parallelized architecture along with special hardware units for common packet processing tasks enable packet forwarding at high rates with enough headroom for network emulation. We expect to be able to emulate a larger number of links and routers with a greater aggregate network bandwidth on a software emulation platform using the IXP1200 as compared to a modern PC class workstation. We will validate this claim by designing and implementing such an emulator on the IXP1200 network processor.

1 Introduction

1.1 Objective

To demonstrate that using IXP series of network processors instead of PC class workstations as an emulation platform can lead to more scalable, accurate and extensible network emulation.

1.2 Background

Traditionally, there exist three environments for testing and validating network and distributed systems research: network simulators [17], network emulators [2, 14] and live networks—each with obvious strengths and weaknesses. Single node WAN emulators like Dummynet [14], NistNet [8] and ONE [3] emulate WAN characteristics like network delay, bandwidth, packet loss, and packet reordering. Packets entering a Dummynet node are classified into flows using the IP firewall and then injected into Dummynet for emulation. Modelnet [18] is a scalable network emulator which supports hop-by-hop emulation of large network topologies on a gigabit PC cluster. End-nodes are configured to send all traffic through a core node cluster, which emulates the network by subjecting the packet to the loss/delay/bandwidth characteristics of every link in the path. Emulab [20], the Utah Network Testbed, provides an emulation environment available for remote use to networking researchers all over the world. Researchers specify a virtual topology graphically or via an ns script, along with traffic patterns, link and router characteristics and Emulab automatically maps these into real resources dedicated to the experiment. Through automated experiment configuration, the ability to dynamically modify network and traffic characteristics, and an interactive style of experimentation made possible due to creating and tearing experiments in minutes, Emulab brings much of a simulator's ease of use and control to emulation.

1.3 Motivation

Emulab's use of a PC class workstation for link emulation has two main drawbacks: *scalability and accuracy.* Since emulation runs in real-time, link and router emulation scaling depends on an emulator's packet forwarding capacity. A higher packet forwarding capacity with support for handling higher aggregate bandwidth translates into an ability to emulate more links and routers, which helps to scale to larger topology sizes. The accuracy of PC based emulators is dependent on the kernel's scheduling granularity. For gigabit forwarding of large packets, a low Hz rate causes burstiness in forwarding and loss of emulation accuracy. Increasing the Hz rate on the emulation kernel to 10 kHz or more reduces this burstiness but adds more overhead on the node due to a high rate of timer interrupts.

1.4 Research proposal

In this proposal, we address the problem of scalability, and accuracy under high loads, of link and router emulation by using the IXP series of network processors [9] as an emulation platform. Link emulation involves emulating link bandwidth, link delay, and packet loss along with other WAN characteristics like packet reordering and packet duplication. Router emulation involves emulating an abstraction of the forwarding path of a router which consists of the following key elements: a queuing discipline, a link

scheduling mechanism, and classifiers, meters and shapers for diffserv like applications. The first and second generation of IXP network processors support a multiprocessor architecture with 6 and 8 independent programmable microengines running at a frequency of 232 MHz and 600 MHz respectively. Each microengine supports hardware multithreading and low overhead context switches on memory and IO accesses for latency hiding. The processor also supports a distributed shared memory hierarchy [19] with low-latency access to QDR SRAM which can be used for small data structures such as link emulation descriptors and DDR SDRAM for storing packets for high delay-bandwidth product links. Special units like hash generators, pseudo random generators and timers support fast implementation of common packet processing tasks in hardware [10]. In addition to packet forwarding, these hardware units are also very useful for fast emulation. Hash generators can be used in classification engines, pseudo random generators in implementing RED [7] and for implementing link delay and link loss models while high resolution timers can support accurate link emulation.

The Network Systems group at Princeton has built a software based router on the IXP1200 network processor [16]. Their work shows that the IXP1200 can forward packets at an order of magnitude faster than PC based routers assuming infinite line rate, while at lower line rates, there is enough headroom for application specific processing. We have built a prototype emulator on this platform which does link delay and bandwidth shaping with promising results. We intend to use the ENP-2505 (bridalveil) IXP1200 board [5] with Intel's Active Computing Element (ACE) model [1] for our final implementation. To maintain emulation accuracy, we will take advantage of the multiprocessor architecture of the IXP1200. The emulation pipeline can be run on different microengines than the input and output pipelines. Under a high load, as the input and output pipelines are busy moving packets from/to the NIC to packet memory, the emulation pipeline can use microengine local memory for most of its state and in parallel execute the emulation modules. If IO pipelines and emulation pipeline have to access shared resources like SRAM, SRAM priority command queues can be used to give priority to the emulation module [11]. To address extensibility of the emulator, we will design our emulation platform within the ACE programming framework. This model gives us extensibility for free, newer queuing disciplines, or link scheduling policies or link loss models can be plugged in as components into the input, output or emulation pipelines.

2 Proposed Work

2.1 Design and Implementation of the emulator

To support our claim of scalable, accurate and extensible emulation using IXP1200 network processors, we will design and implement an emulator on the ENP-2505 (bridalveil) IXP1200 board [5] within the ACE programming framework. Using the ACE model will be beneficial in two respects: as described earlier, we get extensibility for free, and porting our emulation framework to future versions of the processor like the IXP2400 will be relatively easier. We have not yet designed the system completely, but we envisage the emulation module to run in its own pipeline, different from the input and output pipelines. Our code will act as a plug-in to the bridging infrastructure provided by the L2bridge ACE reference design, similar to how Dummynet plugs into the bridging code on a FreeBSD node.

Packets entering the emulator are classified into flows based on the 4 tuple of src IP, dst IP, src port and dst port. Since our target application is emulation, we do not need to implement a generic classifier like IP firewall on FreeBSD. A simple hash based classifier can use the hardware hash unit and packet header fields to quickly stash the packet into its flow. The emulation code is based on Dummynet, however it will be a complete rewrite in microcode within the ACE framework. Packet scheduling uses a heap of pipes with the deadline of packets in the pipe as the key. At every tick, the scheduler handles pipes whose deadline is reached. Packet descriptors are dequeued from the bandwidth queue at a programmed rate and inserted in a delay queue where the link delay is emulated. Once emulation is complete, packets are queued in output queues to be handled by the output pipeline.

2.2 Challenges

The IXP1200 is not as well understood as a PC platform and thus not an easy system to program. The biggest challenge is to design the application to make optimal use of the vast set of parallel resources while avoiding synchronization bottlenecks and ensuring consistency of shared data. The ACE model provides a framework to develop individual code components and then create pipelines of these components, on the microengines as well as on the StrongArm. However the application programmer has to decide the number and types of pipelines, whether a context or functional pipeline [10], and the state sharing mechanisms among pipelines. In our application, the challenge lies in splitting components among different pipelines, and instantiating multiple instances of the same pipeline on more than one microengine as the pipeline complexity increases. For instance, if we add a classification engine to the input and bridging pipeline, we might have to instantiate the ingress code on two microengines to handle the 4 100 Mb ports because of the extra overhead. Note that default ACE configuration uses only 1 microengine for handling receive on 4 slow ports. The other option is to run classification on a separate microengine which has the disadvantage that it involves state sharing and packet handoff through memory which can become expensive.

Another challenge lies in measuring the relative performance of alternative implementations of a component, and understanding how components interact in the system. For instance, we will run micro-level benchmarks [6] for measuring the overhead of the two queue Dummynet model and also the overhead of the heap data structure for scheduling pipes. Based on the results of these benchmarks, we will specifically look at two optimizations: use of a combined bandwidth and delay queue to avoid extra memory copies and the use of a calendar queue instead of heap for pipe scheduling to scale to large number of pipes [4].

2.3 Validation and Emulation Scalability measurements

We will run validation tests to confirm that the emulator does emulate links and routers to the desired accuracy for different loads up to its maximum capacity. These tests will involve validating all the features of the emulator like link delay, bandwidth, and packet loss rate. We will then measure the maximum packet forwarding capacity of the emulator for both large and small packets. We expect that the forwarding capacity will be limited by the wire (4 100 Mb interfaces) for large packets and by the CPU for small packets. We will also run scaling tests to see how our implementation scales on multiplexing multiple virtual links onto physical links.

3 Preliminary Work

As discussed earlier in section 1.3, PC emulators do not scale well for emulation of large network topologies because of their modest packet forwarding rates. Below we present results from some experiments that we conducted in Emulab to measure their packet forwarding capacity and scalability. We do not have access to commercial emulators [15, 13]. We used Dummynet in our measurements since it is representative of other single node PC emulators like ONE, NistNet for the capacity and scale metrics.

3.1 Dummynet forwarding capacity measurements

- The Dummynet hardware for the first test was a 850 MHz PC with a 32 bit, 33 Mhz PCI, 512 MB memory, four *fxp* 100 Mb ports running FreeBSD-4.7. We ran the kernel with polling enabled for all the four ports thus avoiding interrupt overheads and receive livelocks. All Dummynet pipes were configured with 10ms delay, infinite bandwidth and 0 packet loss. For 64 byte packets, the Dummynet node was able to forward 105 Kpps for a two port configuration. For maximum sized packets of 1514 bytes, the node was able to forward at line rates for 1, 2, and 3 ports while it was able to forward about 320 Mbps for a 4 port test. Since packets on the emulator traverse the PCI bus twice, once on the way in, once on the way out, a 400 Mbps link traffic translates rougly into a 800 Mbps PCI traffic which seems to become the bottleneck in the large packet test.

- To confirm the PCI bottleneck, we ran the same experiment on a different hardware. This test was run on a 2 Ghz machine with a 32 bit, 33 MHz PCI, 512 MB memory and two *nge* gigabit interfaces. The node ran FreeBSD-4.7 which had polling drivers for the *nge* gigabit interfaces. While the packet rate for small packets increased to about 170 Kpps, for large packets it was approximately 26 Kpps or 314 Mbps which was rougly same as the 850 MHz PC test. Thus it seems that the PCI is indeed a bottleneck on our hardware, especially for large packets.

3.2 Dummynet scaling with increasing number of pipes

Most realistic wide-area experimental scenarios have link bandwidths which are much lower than the 100 Mb physical link bandwidth. Thus many virtual links of lower bandwidth can be multiplexed onto physical links. For instance, if Dummynet can support an aggregate of 400Mb for large packets, this translates into 4 100Mb physical links, or 40 10Mb virtual links or 400 1Mb virtual links. The support for multiple virtual links involves two main components: a classification engine for flow classification and a scheduling mechanism to schedule the multiple pipes. While Dummynet uses a heap for pipe scheduling which scales reasonably well with increasing number of pipes, the bottleneck seems to be in the classification stage. It uses the FreeBSD IP firewall which scales poorly as O(n) where n is the number of classification rules. We however could not measure the exact scaling numbers, since as discussed above, PCI becomes the bottlneck on our hardware configuration.

4 Project Budget

We request funds to participate in the Network Processors Conference (NPC 2003 East) [12]. This would include travel fare, registration fees and lodging expenses for the duration of the conference. We will provide the exact expenses after the conference programme and registration fees are announced.

5 Conclusion

We have described some preliminary work which measured Dummynet based PC emulator's forwarding capacity. We also implemented a prototype emulator on the IXP1200 evaluation board. Based on this experience, we propose that using IXP series of network processors instead of PC class workstations as an emulation platform can lead to more scalable, accurate and extensible network emulation.

We will design and implement such an emulator and integrate it with Emulab. The Proposed work section describes our initial design and some of the challenges involved in programming the IXP1200.

References

[1] Intel Corporation: Intel IXA SDK ACE Programming Framework Developer's Guide, June 2001.

[2] J. S. Ahn, P. B. Danzig, Z. Liu, and L. Yan. Evaluation of TCP Vegas: Emulation and Experiment. In *Proc. of SIGCOMM '95*, pages 185–195, Cambridge, MA, Aug. 1995.

[3] M. Allman, A. Caldwell, and S. Ostermann. ONE: The Ohio Network Emulator. Technical Report TR-19972, School of Electrical Engineering and Computer Science, Ohio University, Aug. 1997. http://ctd.grc.nasa.gov/5610/network-emulation.html.

[4] R. Brown. Calendar Queues: A Fast O(1) Priority Queue Implementation of the Simulation Event Set Problem. *Communications of the ACM*, 31(10):1220–1227, Oct. 1988.

[5] Radisys Corporation: ENP-2505 Hardware Reference Manual, March 2002.

[6] P. Chandra, F. Hady, R. Yavatkar, T. Bock, M. Cabot, and P. Mathew. Benchmarking Network Processors. In *Workshop on network processors held in conjunction with HPCA*, February 2002.

[7] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, Aug. 1993.

[8] N. N. A. A. I. (ITG). NISTNet, 1997. Available at http://www.antd.nist.gov/itg/nistnet/.

[9] Intel Network Processors and Resources. http://developer.intel.com/design/network/products/npfamily.

[10] Intel Corporation - Intel IXP1200 Network Processor - 2nd Generation Intel NPU. http://ixaedu.com/resources/NPBook_IXP2400.pdf.

[11] E. Johnson and A. Kunze. *IXP1200 Programming*. Intel Press.

[12] Network Processors Conference, NPC2003 East. http://www.networkprocessors.com/English/East/index.html.

[13] PacketStorm network emulator. http://www.packetstorm.com.

[14] L. Rizzo. Dummynet and Forward Error Correction. In *Proc. of the FREENIX Track: USENIX Annual Technical Conference*, June 1998.

[15] ShunraStorm network emulator. http://www.shunra.com/PDFs/StormSpec.pdf.

[16] T. Spalink, S. Karlin, L. Peterson, and Y. Gottlieb. Building a Robust Software-Based Router Using Network Processors. In *Proc. of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, pages 216–229, Chateau Lake Louise, Banff, Alberta, Canada, October 2001.

[17] The VINT Project. *The* ns *Manual*, Apr. 2002. http://www.isi.edu/nsnam/ns/ns-documentation.html.

[18] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker. Scalability and Accuracy in a Large-Scale Network Emulator. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 271–284, Boston, MA, Dec. 2002.

[19] M. Venkatchalam, P. Chandra, and R. Yavatkar. A highly flexible, distributed multiprocessor architecture for network processing. *Special Issue of Computer Networks*, 2003. To appear.

[20] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, Dec. 2002.