The Open Network Laboratory*

John DeHart, Fred Kuhns, Jyoti Parwatikar,
Jonathan Turner, Charlie Wiseman and Ken Wong
The Applied Research Laboratory
Department of Computer Science and Engineering
Washington University in St. Louis
St. Louis, MO 63130
{jdd,fredk,jp,jst,cgw1,kenw}@arl.wustl.edu

ABSTRACT

The *Open Network Laboratory* (ONL) is a remotely accessible network testbed of high performance routers which has been designed with an eye towards ease of use for users from the naïve to the expert. The system is built around a set of high-performance routers that are extendible and easily configurable through the *Remote Laboratory Interface* (RLI), an intuitive graphical interface. The RLI also makes it easy to configure packet filters in the routers, assign flows or flow aggregates to separate queues with configurable QoS and attach hardware monitoring points to real-time charts. The RLI's real-time charts and user data facility makes it easy to directly view the effects of traffic as it moves through a router, allowing the user to gain better insight into system behavior and create compelling demonstrations. Each port of the router is equipped with an embedded processor that supports software *plugins* which allow users to extend the system's functionality. This paper describes the ONL and how it can be used in networking education. You should checkout our web site *onl.arl.wustl.edu* which includes a short video.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education, C.2 [**Internetworking**]: Routers

General Terms

Experimentation, Measurement

Keywords

Experimental Computer Science, Education, Real-Time Displays

1. INTRODUCTION

A traditional basic networking course covers the fundamental concepts in networking and is sometimes accompanied by a few laboratory exercises involving either some socket programming and/or simulation experiments. Rarely, can a student experiment with real, high performance routers and closely observe the effect of configuration changes at a variety of network points in real-time. The closed architectures of commercial routers makes them largely inaccessible for this type of activity and the time and effort required to make experimental modifications to these systems makes this approach prohibitively difficult. The *Open Network Laboratory* (ONL) dramatically reduces the "barrier-to-entry" for this kind of activity by providing access to a remote testbed of open, high performance routers and hosts that can be controlled through an intuitive *Remote Laboratory Interface* (RLI).

While ONL has been developed primarily as a tool for networking research, our early experience shows that it can also be a compelling educational tool. The interface provided by the RLI is direct and intuitive, allowing relatively naïve users to start using ONL productively with a minimum of explicit instruction. Getting a simple experiment up and running takes very little time, and the measurement and real-time display tools make it possible for students to see what is happening "under the overs". By observing how configuration changes affect traffic behavior, queue levels and packet drops, students make a much more direct connection between high level classroom concepts and their practical implications, dramatically accelerating the speed at which they gain understanding and insight into the subject matter. We have begun using the ONL in several of our classes and have found it to be extremely useful and popular with students.

Section 2 of the paper describes the architecture of ONL showing the technical components of the testbed. Section 3 describes the basic features of the Remote Laboratory Interface showing how an experiment can be remotely configured and monitored. Section 4 discusses more advanced features such as packet filters and queue management. Section 5 describes router plugins that are software modules that can be inserted along a router's data path to provide custom processing. Section 6 describes a project that demonstrates many of ONL's capabilities. Then, Section 7 concludes with a discussion of other experimental resources and future work.

2. ONL ARCHITECTURE

The Open Network Laboratory consists of four experimental routers called Network Service Platforms (NSPs) plus 40 rack-mounted PCs that serve as end systems and control processors (Figure 1). The hardware components are grouped into four *clusters* with each cluster consisting of a single NSP, a control processor (CP) that manages an NSP, a gigabit Ethernet subnet

[*] This work was supported by NSF (ANI-023826).

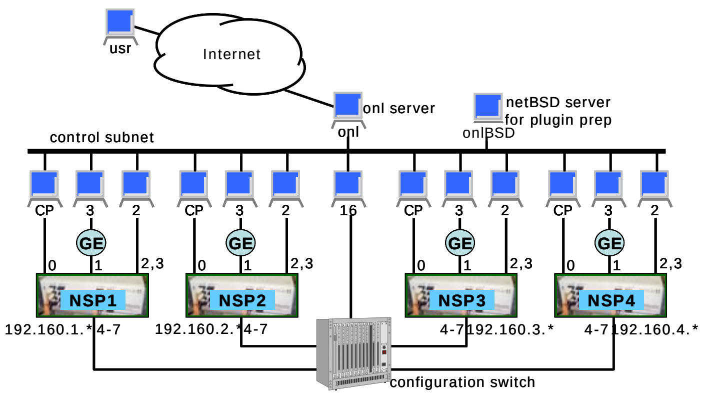


Figure 1. Open Network Laboratory Configuration.

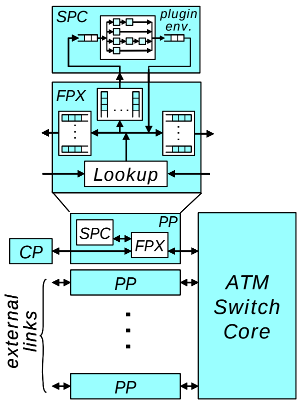


Figure 2. NSP Hardware.

with three connected hosts, and two directly connected hosts. This leaves four of each NSP's ports uncommitted. These four ports are connected to a *Configuration Switch* that serves as an "electronic patch panel" to connect NSPs to each other or to additional hosts. Users interact with the testbed using the RLI, which is a standalone Java application. The RLI communicates with the testbed through the main ONL server which relays messages to the various testbed components. A second server (onlBSD) host is provided to facilitate preparation of software plugins for the NSPs' embedded processors.

The core component is a modular, gigabit router (Figure 2). The system uses a cell-switched core and each port includes an embedded processor subsystem called the *Smart Port Card* (SPC) [1] and a programmable logic board called the *Field Programmable Port Extender* (FPX) [2]. An FPX includes a large field programmable gate array with four high speed memory interfaces providing access to 2 MB of SRAM and 128 MB of DRAM. The system supports several different types of line cards, including one for gigabit Ethernet (GigE). The core cell switch supports 1024 virtual circuits per port, virtual circuit traffic monitoring, multicast and two hardware priority levels. One port of the system is used by an external control processor for system management through in-band control cells.

Packets entering the system first pass to the FPX which is configured to do IP routing, flow classification and packet scheduling. Packets that require software processing are diverted to the SPC on either the input or output side of an NSP. An NSP uses a modular design that allows easy insertion of add-on cards like the FPX and SPC. Such cards are equipped with connectors at either end and are stacked on top of one another. This makes it easy to upgrade individual pieces and to configure systems with a variety of characteristics.

The SPC includes a dual port network interface chip (the *ATM Port Interconnect Controller* or APIC), which allows any portion of the traffic entering or leaving the system to be diverted to the Pentium processor module on the card. The APIC transfers IP packets directly to and from processor memory over a 32 bit PCI bus. In situations where 10% of the link traffic requires software processing, the SPC allows the execution of close to 50 instructions per byte, which is sufficient to implement moderately complex applications that examine and modify packet data.

The FPX contains two field programmable gate arrays. The *Network Interface Device* (NID) can be used to redirect any portion of the arriving traffic to the *Reprogrammable Application Device* (RAD), which is a Xilinx XCV2000E, with 80 KB of on-chip SRAM and 38,400 basic logic blocks, each containing one flip flop, a configurable four variable logic function generator and miscellaneous support circuits. The RAD is equipped with 2 SRAMs and 2 SDRAMs, which can operate at up to 100 MHz, giving it a raw memory bandwidth of up to 2.5 GBytes per second. The available resources allow it to support all the core packet processing functions required of an advanced router supporting gigabit link speeds. The FPX supports dynamic reconfiguration of the RAD. A complete new RAD configuration can be downloaded in just a few seconds.

3. THE REMOTE LAB INTERFACE

The RLI is a standalone Java application that allows a remote user to interactively configure an experiment and monitor a variety of measurement points within the testbed infrastructure. This section describes the basic features of the RLI including resource acquisition, routing table configuration and traffic monitoring. Later sections describe more advanced features such as bandwidth allocation and router plugins. Example exercises are described at the end of each section.

A network can be easily configured through the RLI's main menus and/or per port menus. Figure 3 shows the main RLI panel with its main drop-down menus at the top. The user has added components using the **Topology** menu. The links are shown as dashed lines, and the hosts and NSPs are shown in light shade indicating that actual testbed resources have not been allocated. A cluster consists of an NSP, with its Control Processor (CP), two directly connected hosts and a gigabit Ethernet subnet. Additional hosts were added and linked to each NSP by selecting the **Add Host** and **Add Link** menu items. The **Generate Default Routes** selection generates routing tables at each port that will forward

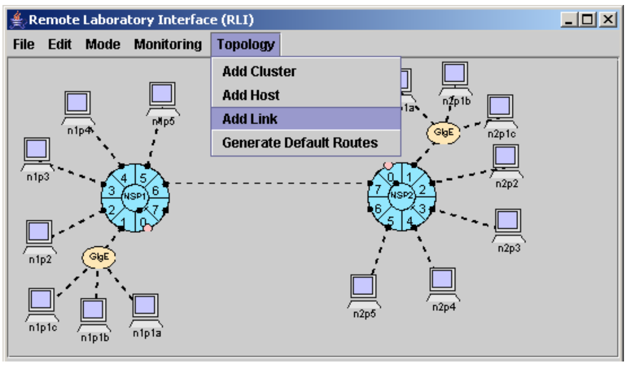


Figure 3. Topology Construction.

packets along minimum hop paths. Although we accepted default values for parameters such as link rates and queue sizes and accepted default routing in this example, the user can modify these settings as well as give special treatment to flows, and install plugins for special packet processing. A configuration can be saved to a file by selecting **File ⇒ Save As**, making it very easy to return to an experiment later.

This topology configuration phase is defined with logical resources and can be done without connecting to the ONL testbed. But in order to run an experiment, the logical configuration must be bound to actual resources. The user connects to the ONL server with an ssh tunnel and selects **File ⇒ Commit** to allocate, bind and initialize physical resources. A user can reserve resources in advance through a web-based reservation system or reserve the resources during the resource binding process. The RLI initiates the setup and as resources are allocated and initialized, dashed links become solid lines and components are displayed in a darker color (see Figure 4). Users are free to modify the configuration at any time through the RLI and commit those changes.

The RLI also provides access to a wide variety of measurement mechanisms that are built into the hardware. Most of these take the form of counters which track either the number of packets observed at a particular place or the amount of data observed. All can be connected to real-time displays, that can be customized in a variety of ways to best suit the user's needs. For example, users can monitor the total traffic sent or received on a link, the amount sent between input and output ports, the number of packets discarded due to various errors, the number of packets matching any given route or packet filter and per flow queue lengths. Figure 4 shows a situation where the user is monitoring the traffic generated by ping traffic from host n1p2 to host n2p3 as it leaves port 6 of NSP 1 and is about to add another plot showing the returning traffic coming into port 6 of NSP 1. Simple exercises can be constructed to familiarize students with the concept of packet forwarding using only the few features we have just described.

Packet Forwarding Exercise: Students construct a network with two or three routers with default routing at all ports. They configure a real-time chart showing the traffic flows on all inter-router links and use the *ping* utility to generate traffic between pairs of hosts. Then, they add route table entries with more specific address prefixes that route packets along more indirect paths and observe the effects on the ping packets. This exercise can be enhanced through the use of packet filters and more sophisticated traffic generators (both described in the next section).

4. FILTERS, QUEUES AND BANDWIDTH

The RLI also allows the user to easily access more advanced features of the hardware such as packet classification, queueing, special routing, and bandwidth sharing. This section describes an experiment that illustrates the effect of bandwidth allocation at a bottleneck link. It uses the two-NSP topology described in the previous section (Figure 4) as a starting point but adds filters that redirect flows to separate reserved queues. It also uses the *iperf* utility [3] to send UDP traffic from the three hosts n1p2, n1p3 and n1p4 to hosts n2p2, n2p3 and n2p4 through the bottleneck link joining port 6 of NSP 1 to port 7 of NSP 2. The RLI displays verify that the flows are receiving their prescribed bandwidth allocations.

These three flows are redirected to separate reserved queues by adding General Match filters in the FPX at the egress side of port 6 of NSP 1. The FPX has three parallel lookup tables at each port: 1) a *Route Table* that uses longest prefix matching, 2) a *Flow Table* that uses Exact Match (EM) filters, and 3) a *Filter Table* that uses General Match (GM) filters. Both EM and GM filters match on a packet's IP address fields, transport layer port fields and protocol field. But GM filters allow wild-carding on fields, and the highest priority entry is chosen when a packet matches multiple GM filters,.

We have set the configuration parameters for the queues at port 6 of NSP 1 so that the egress link capacity is 300 Mbps, and the internal switch capacity has been set to 600 Mbps giving a 2:1 switch speed advantage. The link bandwidth can be set to any rate up to 1 Gb/s. In this example, the desired bandwidth ratios of queues 300-302 were set to 4:2:1 by modifying entries in the Egress Queue Table which control the bandwidth shares of a Weighted Deficit Round Robin (WDRR) packet scheduler. The egress queue sizes for each of these flows were also set in the Egress Queue Table.

Figure 5 shows two RLI charts. The top chart shows the bandwidths in incremental (or stacked) form. The bottom solid curve (1.2 to 1.6) shows the bandwidth entering the bottleneck

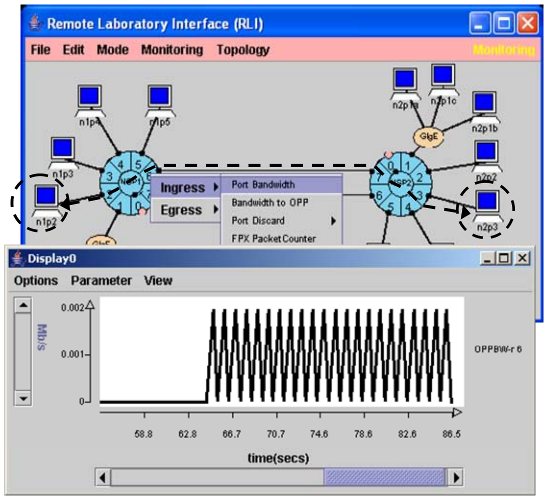


Figure 4. A Traffic Display.

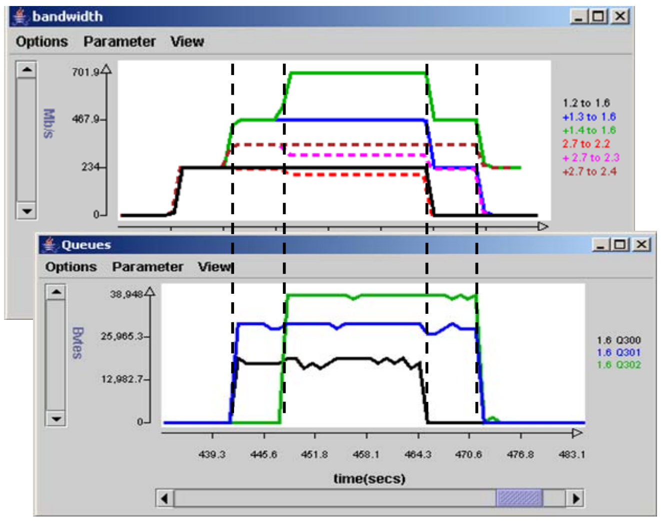


Figure 5. Traffic Bandwidth and Queue Lengths.

link coming from the first flow, the middle solid curve shows the bandwidth contributed by the first two flows, and the top solid curve shows the total bandwidth contributed by all three flows. The dashed curves show the bandwidth leaving the bottleneck link. Note that the bandwidths leaving the bottleneck link are smaller than those entering the bottleneck since the three sources are sending at an aggregate rate of over 700 Mbps, well over the 300 Mbps capacity of the bottleneck. The dashed curves indicate that the three UDP flows are receiving bandwidth in the proportion 4:2:1 when all three flows are active (middle section) and 2:1 (right end) when only qids 301 and 302 have packets. The bottom plot shows the queue length of the reserved flow queues and that the length of the three reserved flows is in the ratio 2:3:4 as required by the threshold settings.

Queueing and Interaction Exercises: The above example showed how multiple flows interact with one another and how their share of link bandwidth can be affected by the queueing subsystem. Variations of our example include: 1) the effect of different rates of input streams on packet loss; 2) the effect of individual queue parameters (e.g., queue length and bandwidth share); 3) the effect of link bandwidth; and 4) the behavior of TCP traffic in place of, or intermixed with UDP traffic. All of these effects can be easily observed using the RLI's monitors and charts.

5. ROUTER PLUGINS

A user can divert traffic to software plugins loaded into the SPCs to perform custom packet processing such as:

- Examine or modify packet headers and/or bodies
- Model packet delays, drops and modifications
- Produce additional packets
- Change the normal packet forwarding action

Figure 6 shows how a packet at an ingress port can flow from a link through the FPX Lookup module to an SPC plugin, back to the FPX and then finally out to the switch core. Packets can also be directed to the SPC on their way out of an egress port.

A user can select from a set of pre-existing, standard plugins. Examples include:

- stats: Collect statistics on packets
- pdelay: Delay packets
- strSubst: Modify packet contents
- multicast: Create multicast packets

The plugins can be used "as is" or their source code can be used as a basis for extension or modification.

In order to use an existing plugin, a user creates an instance of

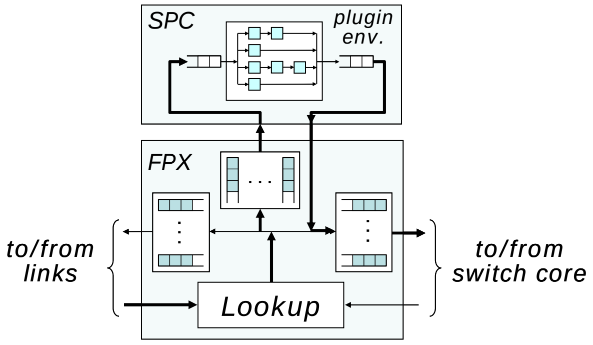
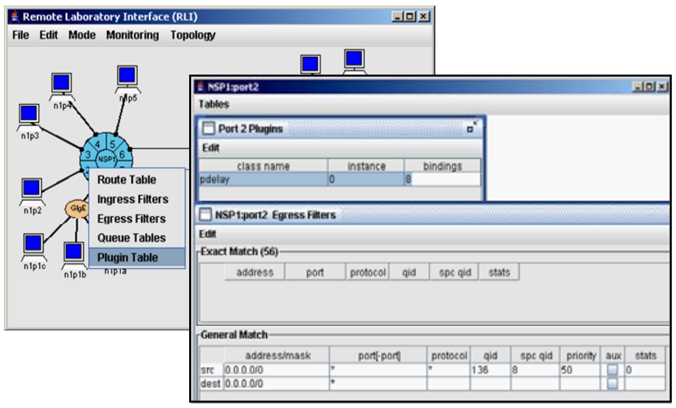


Figure 6. The Plugin Environment.



Port 2 Plugins

Edit

class name	instance	bindings
pdelay	0	8

NSP1:port2 Egress Filters

Edit

Exact Match (56)

address	port	protocol	qid	spc qid	stats

General Match

	address/mask	port[-port]	protocol	qid	spc qid	priority	aux	stats
src	0.0.0.0/0	*	*	136	8	50	☐	0
dest	0.0.0.0/0	*					☐	

Figure 7. Adding a Delay Plugin to Port 2.

the plugin at a port, creates a filter to divert traffic to the plugin instance and then binds the plugin instance to the filter. Figure 7 shows the panels used to create a plugin at the egress side of port 2 to delay TCP ACK packets. The GM filter in this example places all packets into queue 8 which is headed for the SPC where instance 0 of the *pdelay* plugin will delay packets it receives by 50 msec before forwarding them.

A user can send messages to plugins through the RLI. For example, the delay plugin can be told to change its delay, it can be queried for the number of packets that it has forwarded, and it can be told to reset its counters. Also, data from plugins can be easily displayed in real-time panels like any other data.

Plugin Usage Exercise: Students learn how to use the delay plugin and experiment with network parameters to obtain high throughput. Students send a TCP flow through a bottleneck link while using a delay plugin to delay the acknowledgements returned to the sender. They observe how this affects queueing, learn the relationship between the bandwith-delay product and the queue size needed to ensure high throughput. They use the RLI to modify the delay and observe the effect on traffic and queues.

6. A SYN FLOOD MITIGATION PROJECT

This section describes an advanced project that illustrates how students can add a plugin to mitigate the effects of a SYN flood attack and demonstrate the effectiveness of their plugin using the RLI's monitoring facilities. In a SYN flood attack [4], a malicious sender attempts to block new TCP connections at a target by sending SYN packets to begin the process of creating connections that it never intends to complete.

Figure 8 shows the essential elements of the experiment. A browser repeatedly makes legitimate TCP connections to a target Web site to send HTTP image requests. Concurrently, an attacker sends a flood of spoofed SYN packets that each begins but never completes the process of creating a TCP connection. Eventually,

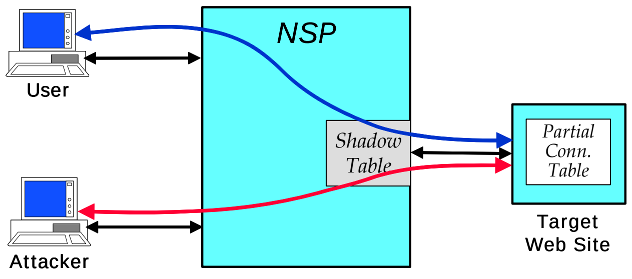


Figure 8. Mitigating a SYN Flood Attack.

the partial connections at the Web site exhaust the site's partial connection table blocking new connections from legitimate users.

A GM filter diverts TCP connection packets to a plugin at the egress port leading to the target web site. The plugin monitors partial TCP connections, records these connections in its *shadow table*, and clears those that don't complete in time. In order to monitor the connection and termination phases of TCP connections, two GM filters are installed which divert these packets through the plugin. If the three-way connect handshake succeeds, the plugin installs an EM filter to allow the web server's response packets to pass through the port without plugin processing. Since the reply traffic accounts for the bulk of the bandwidth usage at the web site, this keeps the amount of traffic that must be handled by the SPC relatively modest. However, when the plugin recognizes that a partial connection (from the attacker) has timed out, it sends a ReSeT packet to the server to release the resources consumed by the incomplete connection and deletes the entry from its shadow table. This ReSeT packet carries the source IP address that was used by the SYN packet that initiated the connection, making the attack mitigation mechanism completely transparent to the target web site.

Figure 9 shows several displays used to demonstrate the effectiveness of the plugin. On the right is a browser window with three Java applet panels. The applet in the top panel plays the role of a Web client sending HTTP image requests at a specified rate (every 3 seconds in the example) and displaying the reply images and response times. When an attack is successful, the image sequence freezes instead of displaying a new image every 3 seconds. The middle panel is used to control the attack daemon, and the bottom panel is used to enable/disable the plugin. The RLI displays on the left show monitored data. In both cases, the plugin has been disabled during the middle of the time interval shown. The bottom display shows the image traffic volume. During a successful attack when the plugin is disabled, image transfers stop shortly after the plugin has been disabled. The top display shows the number of incomplete connections as viewed by the Web server and the plugin. It shows that the Web server's connection table tops out (lower curve) while the plugin continues to see additional attacker packets.

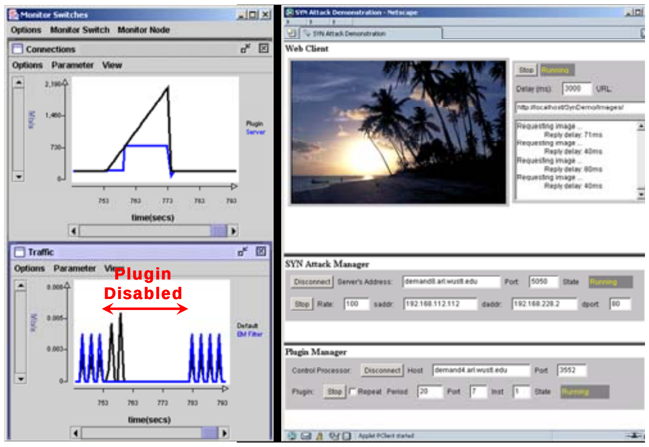
7. CONCLUSIONS

ONL is one of several experimental resources that have become available for networking educators and researchers. These include Emulab[5], PlanetLab [6], the Xbone [7], and Schooner [8]. While these various resources have certain overlapping capabilities, each offers unique features that make it better suited for certain types of experimental uses than others. Planetlab allows delivery of applications to a large geographically distributed user population. Emulab supports emulation of much larger network topologies than ONL. However, these resources either use commercial routers making them difficult to change or use commodity PCs as routers making it difficult to evaluate the performance and implementation complexity.

The strengths of ONL center on its ease of configuration, its extensive real-time displays and its use of extensible, high performance routers whose internal operations is fully open. The extensive real-time displays allow students to easily make a direct connection to classroom concepts through experimentation and observation, thus creating a more satisfying educational experience. ONL's combination of easily configurable packet processing hardware, and a tightly coupled software plugin environment makes it possible to experiment with capabilities in a realistic environment. These features make the ONL a compelling educational tool.

We plan to make it possible for users to modify the configurable logic in the FPX's FPGAs. While the essential technical capabilities needed to support this exist (we routinely load new configurable logic files in order to add features and correct errors), we need to develop mechanisms to ensure this can be done reliably, without risking damage to system components.

We believe that ONL can be an important complement to the set of resources available to networking educators and researchers. We hope that you and your collegues and students will check out the facility and its extensive tutorial pages and register for an account at our web site *onl.arl.wustl.edu*. A short video at the web site gives a quick overview of the RLI in action.



Delay (ms): 3000 URL: http://localhost/SynDemo/images/

Requesting image ...
Reply delay: 71ms
Requesting image ...
Reply delay: 40ms
Requesting image ...
Reply delay: 80ms
Requesting image ...
Reply delay: 40ms

Server's Address: demand8.arl.wustl.edu Port: 5050 State: Running
Rate: 100 saddr: 192.168.112.112 daddr: 192.168.228.2 dport: 80

Control Processor: Host: demand4.arl.wustl.edu Port: 3552
Plugin: Repeat Period: 20 Port: 7 Inst: 1 State: Running

Figure 9. SYN Flood Mitigation Displays.

8. REFERENCES

[1] John D. DeHart, William D. Richard, Edward W. Spitznagel, and Dave Taylor, "The Smart Port Card: An Embedded Unix Processor Architecture for Network Management and Active Networking," Washington University, Department of Computer Science Technical Memorandum WUCS-TM-01-15, July 2001.

[2] John W. Lockwood, Naji Naufel, Jon S. Turner, and David Taylor, "Reprogrammable Network Packet Processing on the Field Programmable Port Extender (FPX)," *Proc. ACM Intl. Symp. On Field Programmable Gate Arrays (FPGA'2001)*, Monterey, CA, Feb. 2001, pp. 87-93.

[3] http://dast.nlanr.net/Projects/iperf/.

[4] CERT, "TCP SYN Flooding and IP Spoofing Attacks," Advisory CA-1996-21, 1996.

[5] Brian White, Jay Lepreau, Leigh Stoller, et. al., "An Integrated Experimental Environment for Distributed Systems and Networks," *Proc. 5th Symp. on Op. Sys. Design & Implementation*, Dec. 2002, pp. 255-270.

[6] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, et.al., "Planetlab: An Overlay Testbed for Broad-Coverage Services," *ACM Computer Communications Review*, Vol. 33, No. 3, July 2003.

[7] Joe Touch, "Dynamic Internet Overlay Deployment and Management Using the X-Bone," *Computer Networks*, July 2001.

[8] http://www.schooner.wall.wisc.edu/.