Report Number: WUCSE-2005-42

2005-09-08

The Open Network Laboratory (a resource for high performance networking research)

John DeHart, Fred Kuhns, Jyoti Parwatikar, Jonathan Turner, and Ken Wong

The Open Network Laboratory (ONL) is a remotely accessible network testbed designed to enable network researchers to conduct experiments using high performance routers and applications. ONL™s Remote Laboratory Interface (RLI) allows users to easily configure a network topology, initialize and modify the routers™ routing tables, packet classification tables and queuing parameters. It also enables users to add software plugins to the embedded processors available at each of the routers™ ports, enabling the introduction of new functionality. The routers provide a large number of built-in counters to track various aspects of system usage, and the RLI software makes these available... **Read complete abstract on page 2.**

The Open Network Laboratory (a resource for high performance networking research)

John DeHart, Fred Kuhns, Jyoti Parwatikar, Jonathan Turner, and Ken Wong

Complete Abstract:

The Open Network Laboratory (ONL) is a remotely accessible network testbed designed to enable network researchers to conduct experiments using high performance routers and applications. ONL™s Remote Laboratory Interface (RLI) allows users to easily configure a network topology, initialize and modify the routers™ routing tables, packet classification tables and queuing parameters. It also enables users to add software plugins to the embedded processors available at each of the routers™ ports, enabling the introduction of new functionality. The routers provide a large number of built-in counters to track various aspects of system usage, and the RLI software makes these available through easy-to-use real-time charts. This allows researchers to expose what is happening ﬁunder the surfaceﬂ enabling them to develop the insights needed to understand system behavior in complex situations and to deliver compelling demonstrations of their ideas in a realistic operating environment. This paper provides an overview of ONL, emphasizing how it can be used to carry out a wide range of networking experiments.

2005-42

The Open Network Laboratory (a resource for high performance networking research)

Authors: DeHart, John; Kuhns, Fred; Parwatikar, Jyoti; Turner, Jonathan; Wong, Ken

September 8, 2005

Abstract: The Open Network Laboratory (ONL) is a remotely accessible network testbed designed to enable network researchers to conduct experiments using high performance routers and applications. ONL™s Remote Laboratory Interface (RLI) allows users to easily configure a network topology, initialize and modify the routers™ routing tables, packet classification tables and queuing parameters. It also enables users to add software plugins to the embedded processors available at each of the routers™ ports, enabling the introduction of new functionality. The routers provide a large number of built-in counters to track various aspects of system usage, and the RLI software makes these available through easy-to-use real-time charts. This allows researchers to expose what is happening fiunder the surfacefl enabling them to develop the insights needed to understand system behavior in complex situations and to deliver compelling demonstrations of their ideas in a realistic operating environment. This paper provides an overview of ONL, emphasizing how it can be used to carry out a wide range of networking experiments.

The Open Network Laboratory
a resource for high performance networking research

Abstract – The *Open Network Laboratory* (ONL) is a remotely accessible network testbed designed to enable network researchers to conduct experiments using high performance routers and applications. ONL's *Remote Laboratory Interface* (RLI) allows users to easily configure a network topology, initialize and modify the routers' routing tables, packet classification tables and queuing parameters. It also enables users to add software *plugins* to the embedded processors available at each of the routers' ports, enabling the introduction of new functionality. The routers provide a large number of built-in counters to track various aspects of system usage, and the RLI software makes these available through easy-to-use real-time charts. This allows researchers to expose what is happening "under the surface" enabling them to develop the insights needed to understand system behavior in complex situations and to deliver compelling demonstrations of their ideas in a realistic operating environment. This paper provides an overview of ONL, emphasizing how it can be used to carry out a wide range of networking experiments.

Keywords: Network Testbed, High Performance Router

Authors: John DeHart, (314) 935-7329, jdd@arl.wustl.edu
Fred Kuhns, (314) 935-6598, fredk@arl.wustl.edu
Jyoti Parwatikar, (314) 935-6110, jp@arl.wustl.edu
Jonathan Turner, (314) 935-8552, jst@arl.wustl.edu
Ken Wong, (314) 935-7524, kenw@arl.wustl.edu

Affiliation: The Applied Research Laboratory,
Department of Computer Science and Engineering,
Washington University in St. Louis

Contact: Ken Wong
Department of Computer Science and Engineering
Washington University in St. Louis
Campus Box 1045
St. Louis, MO 63130
(314) 935-7524, kenw@arl.wustl.edu

The Open Network Laboratory
a resource for high performance networking research

John DeHart, Fred Kuhns, Jyoti Parwatikar, Jonathan Turner 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,kenw}@arl.wustl.edu

Abstract – **The *Open Network Laboratory* (ONL) is a remotely accessible network testbed designed to enable network researchers to conduct experiments using high performance routers and applications. ONL's *Remote Laboratory Interface* (RLI) allows users to easily configure a network topology, initialize and modify the routers' routing tables, packet classification tables and queuing parameters. It also enables users to add software *plugins* to the embedded processors available at each of the routers' ports, enabling the introduction of new functionality. The routers provide a large number of built-in counters to track various aspects of system usage, and the RLI software makes these available through easy-to-use real-time charts. This allows researchers to expose what is happening "under the surface" enabling them to develop the insights needed to understand system behavior in complex situations and to deliver compelling demonstrations of their ideas in a realistic operating environment. This paper provides an overview of ONL, emphasizing how it can be used to carry out a wide range of networking experiments.**

I. INTRODUCTION

As the Internet has matured and become more complex, it has become increasingly difficult for networking researchers to conduct research that requires experimental modifications to the data path of high performance routers. The closed architectures of commercial routers makes them largely inaccessible for this type of research and in any case, the time and effort required to make experimental modifications to these systems, makes this type of work prohibitively difficult for most researchers. This is unfortunate, since many of the more exciting opportunities for advanced network services require the introduction of new functionality in the router data path. The *Open Network Laboratory* (ONL) has been designed as a resource for the networking research community, to enable researchers to conduct experimental research using high performance routers and applications. ONL dramatically reduces the "barrier-to-entry" for this kind of research 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).

ONL builds on an earlier effort at Washington University, in which *Gigabit Network Kits* [5] were produced for use by research groups at over thirty other universities. Each kit consisted of an open high performance switch, network interface cards and associated software. While the kits program was moderately successful, it became clear that most groups found it difficult to maintain the level of expertise needed to manage the experimental equipment and use it effectively. They found themselves spending far too much time on mundane system administration and too little time using the equipment for network experiments. The more recent, and highly successful development of Emulab [6], provided an alternate model for how to enable experimental network research. In developing our ideas for the Open Network Lab, we have directly borrowed the Emulab approach, although we have substituted high performance routers with packet forwarding in hardware, for Emulab's PC-based routers. This enables researchers to work directly with systems that are architecturally similar to commercial routers. The routers' packet forwarding and queueing mechanisms are implemented using configurable logic that can be dynamically reconfigured and each port has an embedded processor that hosts software plugins that can be added to provide new capabilities.

The RLI allows a remote user to easily configure experiments and monitor components (e.g., traffic, queues). The extensive support for real-time data visualization allows users to develop the insights needed to understand the behavior of new capabilities and allows researchers to deliver compelling demonstrations of their research ideas in a realistic operating environment.

Section II of the paper describes the architecture of ONL showing the technical components of the testbed. Section III describes the basic features of the Remote Laboratory Interface showing how an experiment can be remotely configured and monitored. Section IV discusses more advanced features such as packet filters and queue management. Then, Section V describes a demonstration that features the use of router plugins and hardware filters

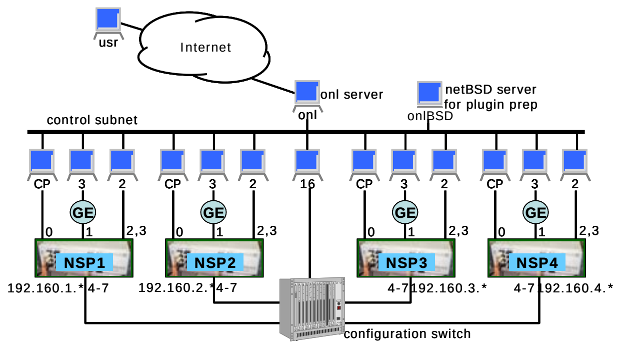


Fig. 1. Open Network Laboratory Configuration.

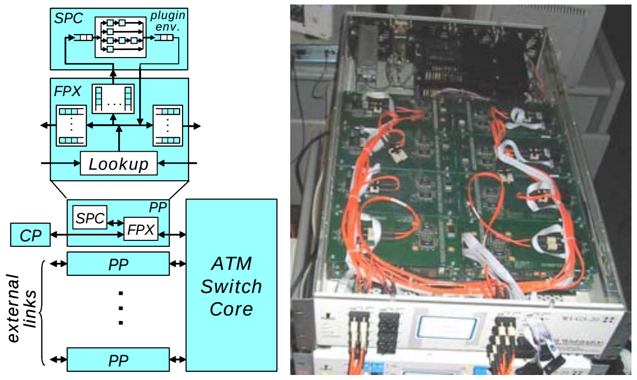


Fig. 2. NSP Hardware.

in a dynamic manner to mitigate a type of Denial of Service (DoS) attack.

II. ONL Architecture

The current equipment configuration for the Open Network Laboratory consists of four experimental routers called Network Service Processors (NSPs) plus 40 rack-mounted PCs that serve as end systems and control processors (Fig. 1). The hardware components are grouped into four *clusters* with each cluster consisting of a single NSP, a control processor (CP) that manages the NSP, a gigabit Ethernet subnet 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. The testbed can support simultaneous sessions by multiple users, so long as there are sufficient resources available. A second server (onlBSD) host is provided to facilitate preparation of software plugins for the NSPs' embedded processors.

The Configuration Switch is used to implement virtual network topologies linking the routers to one another and to hosts. When multiple experimental networks are present in the testbed, they operate completely independently. The configuration switch has a number of unused ports that we plan to use to deliver additional features in the future, such as programmable link delays and high rate traffic generators.

The core component of our testbed is a modular, gigabit router (Fig. 2). The system uses a cell-switched core and the per port interface hardware includes an embedded processor subsystem, called the *Smart Port Card* (SPC) [7], and a programmable logic board, called the *Field Programmable Port Extender* (FPX) [8,9], which 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, per virtual circuit traffic monitoring, support for multicast and two hardware priority levels. One port of the system is typically used by an external control processor for system management through in-band control cells.

Packets entering the system pass first to the FPX, which can be configured to do IP routing, flow classification and packet scheduling. Packets that require software processing can be diverted to the SPC on either the input or output side of the system. The system 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 [10]), 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 PC 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 the 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.

III. 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. Additional information can be found at http://arl.wustl.edu/projects/onl.

The first step in constructing an experiment is to define the network components and topology. The RLI has two viewing modes: topology configuration and monitoring. Fig. 3 shows the main RLI panel during the configuration phase 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 the components have not yet been bound to actual testbed resources. A cluster consists of an NSP, with its CP, two directly connected hosts and a gigabit Ethernet subnet with three more hosts. Additional hosts can be added and linked to other ports by selecting **Topology** ⇒ **Add Host** and **Topology** ⇒ **Add Link**. The **Generate Default Routes** item in the **Topology** menu initializes the NSPs' routing tables so that packets sent to any host will be routed to it along some minimum hop path.

To allocate and initialize physical resources in the testbed, the user selects **File** ⇒ **Commit**. Assuming the requested resources are available, the RLI display will be adjusted as shown in Fig. 4. Note that the links are now shown as solid lines and the components are displayed in a darker color. This signals that the components are now bound to actual hardware resources. The names and IP addresses of each host can be determined by right-clicking on the host as shown in Fig. 4. Each host has two names, an internal name (n1p5 in the example shown in Fig. 4) and a globally visible DNS name (onl23.arl.wustl.edu) that

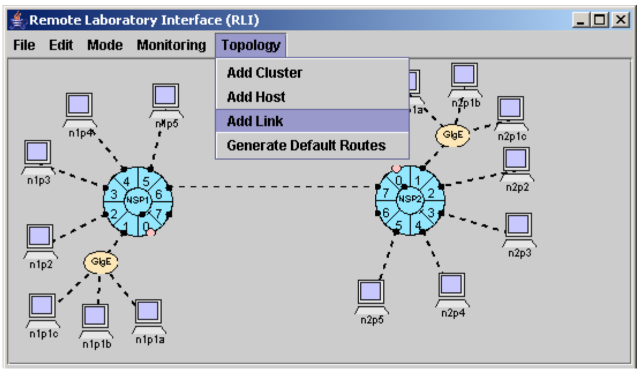


Fig. 3. Topology Construction.

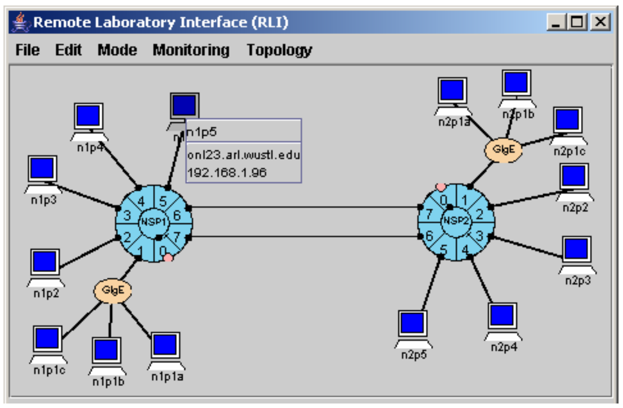


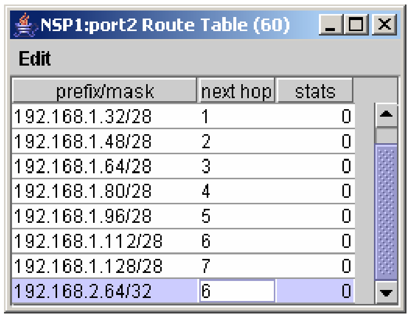
Fig. 4. RLI Display Following Resource Commit.

can be used to open SSH connections to the host for the purpose of running applications. The IP address shown in the RLI display is the address assigned to the host interface that is internal to the testbed. These addresses are not externally visible, and like the internal names are assigned algorithmically, making it possible to repeat an experiment in different sessions without having to modify names and addresses used in demonstration scripts.

A user can change the routing table at each router's input through the RLI. Fig. 5 shows a route being added at input 2 of NSP 1. Since this prefix has a specified length of 32, it takes precedence over the less specific matching prefix with length 28. Note that each entry includes an IP address prefix with mask length, the next hop port, and a statistics field.

The RLI can also be used to create traffic displays by switching to "Monitoring" mode by selecting **Mode** ⇒ **Monitoring**. This provides access to the various monitoring points within the NSPs. Fig. 6 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.

The NSPs provide mechanisms for monitoring a wide variety of measurements, including link bandwidth, bandwidth usage from inputs to outputs, the number of packets matching any given route or packet filter, and the number of packets discarded due to link overflows or



prefix/mask	next hop	stats
192.168.1.32/28	1	0
192.168.1.48/28	2	0
192.168.1.64/28	3	0
192.168.1.80/28	4	0
192.168.1.96/28	5	0
192.168.1.112/28	6	0
192.168.1.128/28	7	0
192.168.2.64/32	6	0

Fig. 5. Route Table at Port 2, NSP 1.

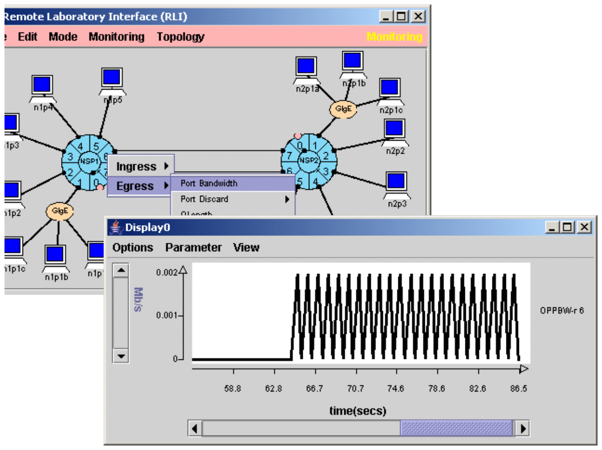


Fig. 6. A Traffic Display.

header errors. All can be connected to real-time displays, that can be customized in a variety of ways to best suit the user's needs. Experimental configurations can be saved to a file, making it relatively easy to return to an experiment in a later session.

IV. FILTERS, QUEUES AND BANDWIDTH

The RLI also provides the user access to more advanced features of the hardware such as packet classification, queueing and redirection, bandwidth sharing and configurable parameters (e.g., link capacity). This section describes a simple experiment in which UDP traffic from multiple sources flowing through a bottleneck link are given different bandwidth and queue shares. The real-time display capability is used to verify that the system behaves as expected.

The experiment uses the two NSP topology described in the previous section (Fig. 4), but instead of sending ping traffic, we use the *iperf* utility [11] 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. These flows will be mapped to separate reserved flow queues at port 6 of NSP 1.

In order to give special treatment to these three flows, we use General Match (GM) filters in the FPX to redirect the UDP flows to separate reserved queues. The default behavior at an egress port is to place packets in a common FIFO datagram queue for the egress link. However, the FPX has three parallel lookup tables at each port (Fig. 7): 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 EM filters differ from GM filters in two respects: GM filters allow wild-carding of any of the fields, and they have assignable priorities. When a packet matches multiple filters, the highest priority entry is chosen.

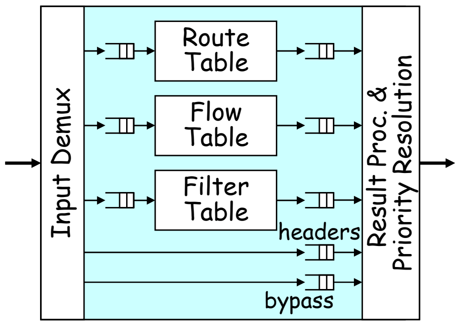
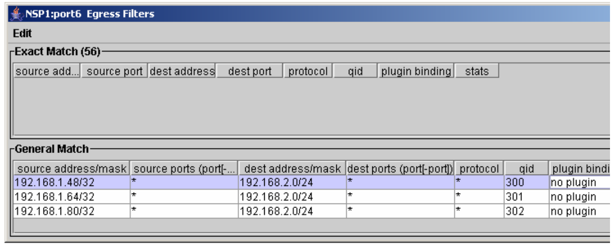


Fig. 7. Parallel Lookup Tables.

Fig. 8 shows the GM filters used to direct the three UDP flows to queues 300-302 respectively at egress port 6. Each of the source address/mask fields match the interfaces of the three sending hosts, and the destination address/mask fields match packets going toward the subnets associated with NSP 2. Since the only UDP flows are those from our traffic generators, we have wild-carded the application port fields.

Fig. 9 shows the configuration parameters for the queues at port 6 of NSP 1. Since each port handles traffic in both directions, the parameters for ingress and egress sides are shown. In this experiment, the egress link capacity has been set to 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. The Egress Queue table shows the three reserved flow qids (300-302) and the datagram queue. The relative values of the entries in the "quantum" field indicate the bandwidth shares of a Weighted Deficit Round Robin (WDRR) packet scheduling algorithm. In this example, the desired bandwidth ratios of queues 300-302 are 4:2:1 (i.e., 8000:4000:2000). The "threshold" column indicates the *discard threshold*; i.e., the queue level above which arriving packets for that queue are dicarded.

Fig. 10 shows two plots. The top plot shows the bandwidths in incremental form. Specifically, the first solid curve shows the bandwidth entering the bottleneck link coming from the first flow, the second solid curve shows the bandwidth contributed by the first two flows and the third shows the total bandwidth contributed by all three flows. The dashed curves show the bandwidth leaving the bottleneck link. Note that the three sources are sending at an aggregate rate of over 700 Mbps, well over the 300



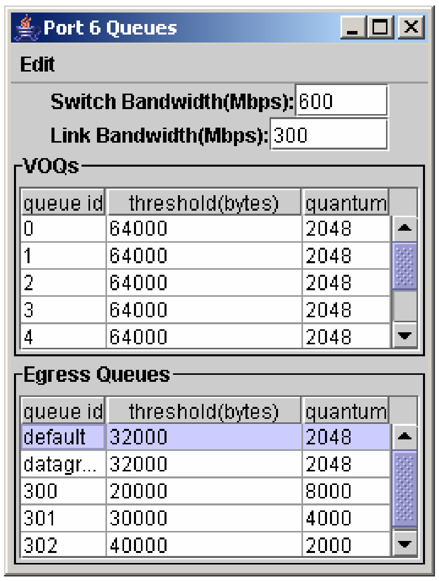
Exact Match (56)

source add...	source port	dest address	dest port	protocol	qid	plugin binding	stats

General Match

source address/mask	source ports (port[-...	dest address/mask	dest ports (port[-port])	protocol	qid	plugin bindi
192.168.1.48/32	*	192.168.2.0/24	*	*	300	no plugin
192.168.1.64/32	*	192.168.2.0/24	*	*	301	no plugin
192.168.1.80/32	*	192.168.2.0/24	*	*	302	no plugin

Fig. 8. Filter Table (GM Filters).


Port 6 Queues

Edit

Switch Bandwidth(Mbps): 600

Link Bandwidth(Mbps): 300

VOQs

queue id	threshold(bytes)	quantum
0	64000	2048
1	64000	2048
2	64000	2048
3	64000	2048
4	64000	2048

Egress Queues

queue id	threshold(bytes)	quantum
default	32000	2048
datagr...	32000	2048
300	20000	8000
301	30000	4000
302	40000	2000

Fig. 9. Port 6 Queue Parameters.

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.

Filters can also be used to re-direct individual flows or flow aggregates to different outgoing links than those specified by the routing tables. GM filters can also be configured to replicate matching packets and direct the copies to a different location. This is useful for passive monitoring of a flow.

V. A DoS Attack Mitigation Plugin

In a SYN flood attack [12], 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. Although host-based techniques for dealing with SYN flood attacks have now been widely adopted, this nonetheless provides a useful illustration of how new capabilities can be added to an NSP using software plugins.

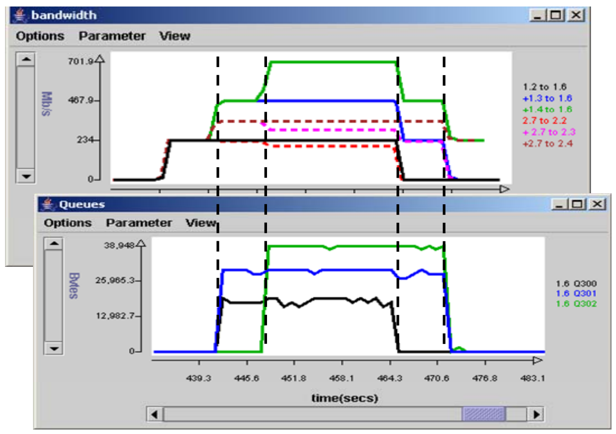


Fig. 10. Traffic Bandwidth and Queue Lengths.

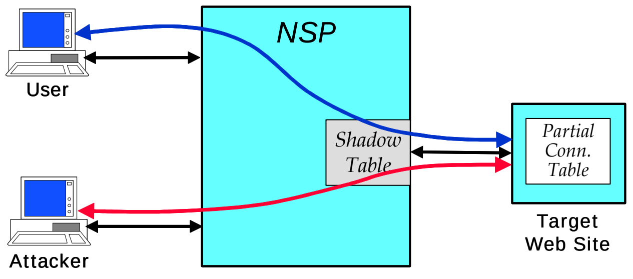
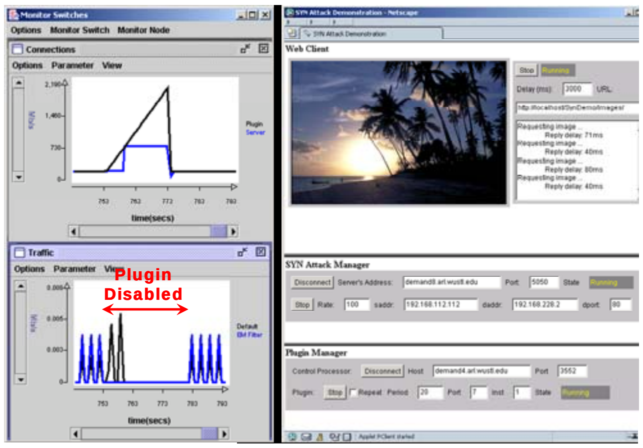


Fig. 11. Mitigating a SYN DoS Attack.

Fig. 11 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, the partial connections at the target Web site exhaust the site's partial connection table blocking new connections from legitimate users.

The NSP's plugin facility can be used to mitigate the effects of this type of DoS attack. TCP connection packets are diverted 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.

Fig. 12 shows several displays used to demonstrate this new feature. 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



State: Running
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

Fig. 12. DoS Attack Mitigation Displays.

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 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.

VI. Conclusions

We have described the Open Network Laboratory, a high performance, remotely accessible network testbed provided as a resource for networking research. We have shown how ONL's Remote Laboratory Interface (RLI) allows users to easily create a network topology, configure the routers in the network and attach the system's extensive traffic monitoring mechanisms to real-time displays that yield insight and help researchers produce compelling demonstrations. We have shown how the functionality of the routers can be extended through the addition of software plugins, providing a rich experimental environment for developing and evaluating advanced services. We believe that ONL can be an important addition to the set of resources available to systems researchers in networking, complementing existing testbeds, such as Emulab and Planetlab.

We have a number of plans for the continued development of ONL. In particular, we plan to add mechanisms to regulate the flow of data through the routers so as to avoid congestion at output ports that can lead to packet loss. The essential mechanisms needed to support this already exist. The remaining step is to add a software component to the SPCs to periodically exchange information about backlogs in VOQs and adjust VOQ pacing rates. We also expect to extend the queueing subsystem to allow dynamic sharing of memory space used by different queues.

In addition, 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.

References

[1] Corporation for National Research Initiatives, *Gigabit Testbeds Final Report*. http://www1.cnri.reston.va.us/gigafr/, 1996.

[2] Very High Performance Network Service, http://www.vbns.net.

[3] Internet 2. http://www.internet2.edu, 2000.

[4] Information Sciences Institute, "Collaborative Advanced Internet Research Network (CAIRN)," http://www.isi.edu/CAIRN.

[5] Kits Gigabit Kits Technology Distribution Program., http://www.arl.wustl.edu/gigabitkits, 1998.

[6] 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.

[7] 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.

[8] John W. Lockwood, Jon S. Turner and David E. Taylor, "Field Programmable Port Extender (FPX) for Distributed Routing and Queueing," *Proc. ACM Intl. Symp. On Field Programmable Gate Arrays (FPGA'2000)*, Monterey, CA, Feb. 2000, pp. 137-144.

[9] 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.

[10] Zubin Dittia, Guru Parulkar and Jerry Cox, "The APIC Approach to High Performance Network Interface Design: Protected DMA and Other Techniques," *Proc. Infocom*, Kobe, Japan, 1997.

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

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