The Open Network Laboratory
A resource for networking research and education

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 designed to enable networking faculty, students and researchers to conduct experiments using high performance routers and applications. The system is built around a set of extensible, high-performance routers and has a graphical interface that enables users to easily configure and run experiments remotely. ONL's *Remote Laboratory Interface* (RLI) allows users to easily configure a network topology, configure routes and 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 remote visualization features of the RLI make 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 provides a simple environment for software *plugins* allowing users to extend the system's functionality. This paper describes the general facilties and some networking experiments that can be carried out. We hope that you and your collegues and students will check out the facility and register for an account at our web site onl.arl.wustl.edu.

Categories and Subject Descriptors

C.2 [Internetworking]: Routers

General Terms

Experimentation, Measurement

Keywords

Experimental testbed, Education, Real-time displays

1. 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 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 and educational communities, to enable users to conduct experiments 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* [1] were produced for use by research groups at over thirty other universities. 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. The more recent, and highly successful development of Emulab [2], 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.

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 users 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 router plugins that are software modules that can be inserted along a router's data path to provide custom processing.

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

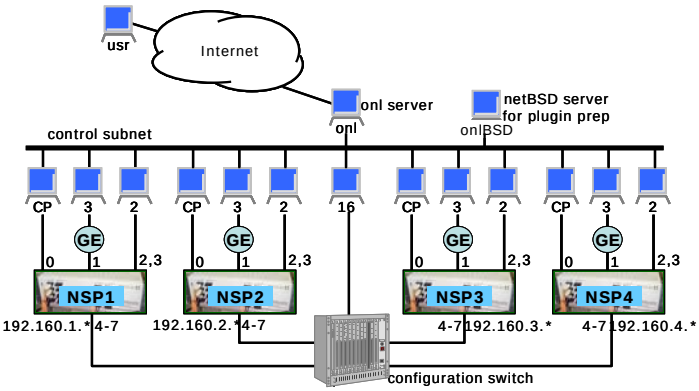

Figure 1. Open Network Laboratory Configuration.

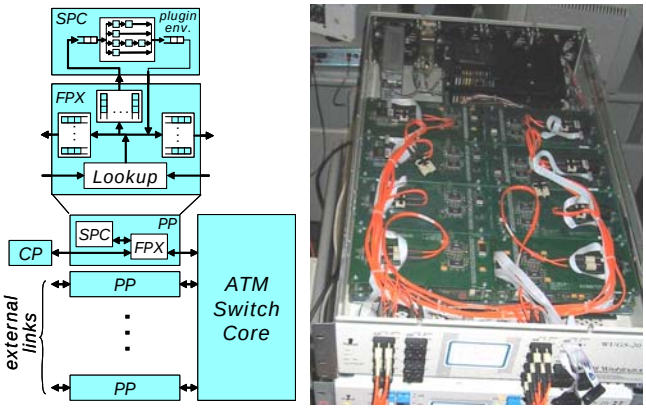

Figure 2. NSP Hardware.

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 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. A second server (onlBSD) host is provided to facilitate preparation of software plugins for the NSPs' embedded processors.

The core component of our testbed is a modular, gigabit router (Figure 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) [3], and a programmable logic board, called the *Field Programmable Port Extender* (FPX) [4], 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 first pass 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), 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 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.

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.

The first step in constructing an experiment is to define the network components and topology. Figure 3 shows the main RLI panel during the configuration phase with its main drop-down

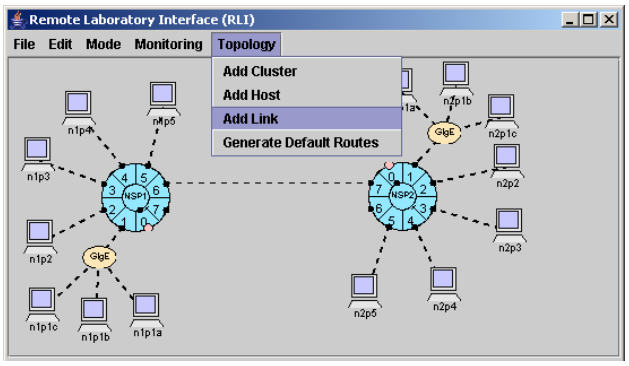

Figure 3. Topology Construction.

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 Control Processor (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. Experimental configurations can be saved to a file by selecting **File ⇒ Save As**, making it relatively easy to return to an experiment later.

Although in this example we accepted default values for parameters such as link rates and queue sizes and accepted default routing, the user can modify these settings as well as give special treatment to flows, and install plugins for special packet processing. 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 user must reserve resources and the system must bind the logical components to actual resources.

A user can either reserve resources in advance through a web-based reservation system that is modeled after a restaurant reservation system or reserve the resources during the resource binding process. To allocate, bind and initialize physical resources in the testbed, the user connects to an ONL server with an ssh tunnel and selects **File ⇒ Commit** in the RLI. 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).

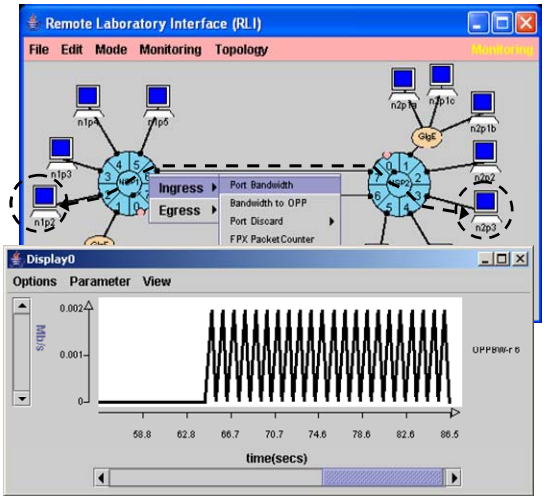

Figure 4. A Traffic Display.

The RLI can also be used to visually display traffic moving through various monitoring points within the NSPs. 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.

The NSPs provide a wide variety of monitoring points, such as link bandwidth, the number of packets matching any given route or packet filter, queue lengths and the number of packets discarded due to link overflows or 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.

4. FILTERS, QUEUES AND BANDWIDTH

The RLI also allows the user to access more advanced features of the hardware such as packet classification, queueing and redirection, and bandwidth sharing. 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 (Figure 4), but instead of sending ping traffic, we use the *iperf* utility [5] 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.

In order to give special treatment to these three flows, we use General Match filters in the FPX at the egress side of port 6 of NSP 1 to redirect the flows to separate reserved queues. 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 EM filters differ from GM filters in two respects: GM filters allow wild-carding of these fields, and they have assignable priorities. When a packet matches multiple filters, the highest priority entry is chosen.

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 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,

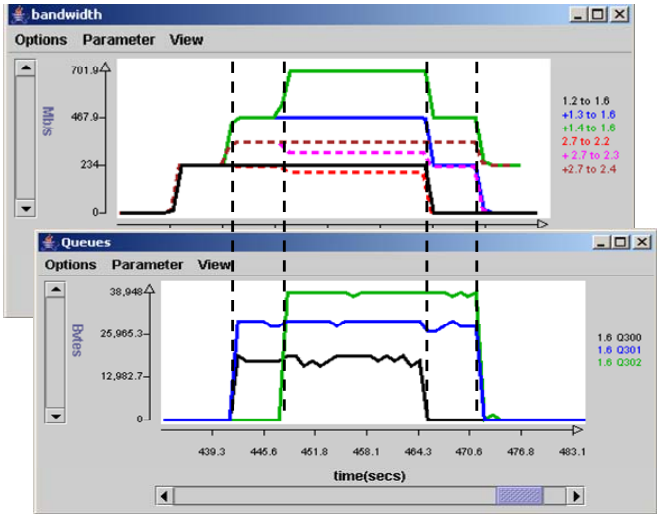

Figure 5. Traffic Bandwidth and Queue Lengths.

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 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.

5. ROUTER PLUGINS

A user can divert traffic to 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 flows from a link through the FPX to an SPC plugin, back to the FPX and then finally out to the switch core.

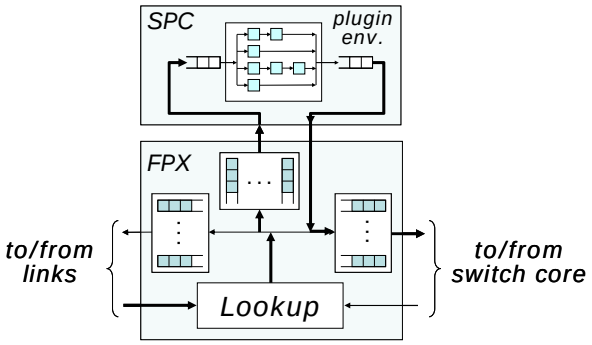

Figure 6. The Plugin Environment.

In order to use an existing plugin, a user creates an instance of 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 select from a set of standard plugins or write his/her own plugin.

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

6. CONCLUSIONS

We have described the Open Network Laboratory and 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. We have also shown how the functionality of the routers can be extended through the addition

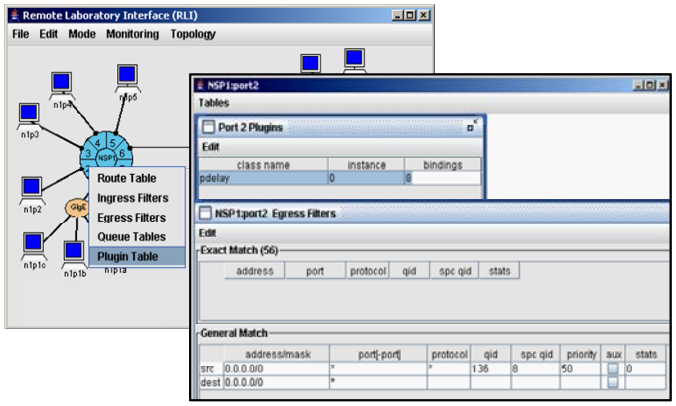

Figure 7. Adding a Delay Plugin to Port 2.

of software plugins, providing a rich experimental environment for developing and evaluating advanced services. The ONL has other features (e.g., user-data displays, debugging, plugin writing) which have been omitted for the sake of brevity here.

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 addition to the set of resources available to systems researchers in networking, complementing existing testbeds, such as Emulab and Planetlab. We hope that you and your collegues and students will check out the facility and register for an account at our web site *onl.arl.wustl.edu*.

7. REFERENCES

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

[2] 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.

[3] 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.

[4] 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.

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