Using the Open Network Lab

Jon Turner
Applied Research Laboratory
Computer Science and Engineering Department
http://www.arl.wustl.edu/arl

Motivation

■ What is ONL?
  - remotely accessible networking lab
  - gigabit routers with configurable hardware packet forwarding and embedded processors at each port
  - routers can be remotely configured through intuitive GUI
  - extensive support for traffic monitoring/visualization
  - resource for network research community

■ Why did we build ONL?
  - difficult to experiment with high performance routers
    - commercial routers are not open
    - open PC routers have limited performance & experiments using them may have limited relevance to high performance routers
  - net research community needs better experimental resources

■ What can you do with ONL?
  - evaluate new and existing protocols & apps in realistic testbed
  - add new features to routers (embedded processors, hw mods)
  - mount compelling demonstrations using real-time visualization
ONL Lab Overview

- Gigabit routers with embedded processors.
- PCs serve as hosts.
  - half on shared subnets
- Net configuration switch.
  - link routers in virtual topologies
  - traffic generation
  - programmable delay
- Tools for configuration and collecting results.
  - monitoring traffic
  - data capture and playback
- Open source
  - all hw & sw sources on web

Mitigating Denial of Service Attacks

Users request connections to communicate with web site

Extensible Router

Partial Conn. Table

Attacker

Users request connections to communicate with web site

Extensible router observes partial connections and clears those that don’t complete

Table fills up blocking legitimate users.

Requires temporary entry in table of partial connections

Shadow Table

Attacker repeatedly start connection process but don’t complete it.
Attack Mitigation Displays

Conn. Table fills when plugin off
Image xfer blocked
Table clears when plugin
Image xfer resumes

People Who Make it Happen

Ken Wong
Lab administration
Web site manager

Jyoti Parwatikar
RLI software development

Fred Kuhns
SPC software
FPX hardware

John Dehart
FPX hardware
System integration
Gigabit Router Architecture

- Scalable architecture built around ATM switch core.
  - core provides 2 Gb/s bandwidth per port (2x speedup)
- Port processors (PP) implement packet processing
  - Field Programmable Port Extender (FPX) implements routine packet processing
  - Smart Port Card (SPC) hosts programmable extensions
- Control Processor (Linux PC) handles configuration
  - can support routing protocols, OA&M, etc.

Field Programmable Port Extender (FPX)

- **Network Interface Device (NID)** routes cells to/from RAD.
- **Reprogrammable Application Device (RAD)** functions:
  - implements core router functions
    - Xilinx Virtex 1E family
      - 38K logic cells (LUT4 + flip flop)
      - 160 block RAMs, 512 bytes each
  - Core router functions include
    - packet classification & route lookup
    - packet storage manager
    - queue manager
      - link queues (datagram, reserved)
      - per flow SPC queues
      - virtual output queues to switch
    - control cell processing
      - access status & control registers
      - update route tables, packet filters
Packet Processing in the FPX

- Input/Output Segmentation and Reassembly (ISAR/OSAR)
  - separate reassembly context for link, SPC and each input port
  - IP packets extracted and stored in memory "chunks" by PSM
  - headers passed to "control path"
  - packets retrieved from memory on output and segmented

- Packet Storage Manager (PSM)
  - stores packets in one of two SDRAMs based on where arriving from

- Classification and Route Lookup (CARL)
  - route lookup (best prefix match) using external SRAM
  - flow lookup (exact 5-tuple match) using external SRAM
  - packet classification (general match) using on-chip resources

- Queue manager (QM) implements three sets of queues
  - link queues per-flow and datagram queues using weighted DRR
  - virtual output queues to switch with controllable output rates
    - can be adjusted by control process in SPC
  - SPC queues using weighted DRR

- Control Cell Processor (CCP)
  - access to traffic counters, updates to lookup tables & control registers
Classification and Route Lookup (CARL)

- Three lookup tables.
  - route table for routing datagrams – best prefix
  - flow table for reserved flows – exact match
  - filter table for management (adr prefixes, proto, ports)

- Lookup processing.
  - parallel check of all three
  - return highest priority primary entry and highest priority auxiliary entry
  - each filter table entry has assignable priority
  - all flow entries share same priority, same for routes

Route lookup & flow filters
- share off-chip SRAM
- limited only by memory size

General filters done on-chip
- total of 32

Lookup Contents

- Route table – ingress only
  - output port, Queue Identifier (QID)
  - packet counter
    - incremented when entry returned as best match for packet

- Flow table (exact match) – both ingress and egress
  - output port – for ingress
  - Queue Identifier (QID) – for egress or SPC
  - packet and byte counters
    - updated for all matching packets

- Filter table (general) – ingress or egress (rate limits)
  - for highest priority primary filter, returns QID
    - packet counter incremented only if used
  - same for highest priority auxiliary filter

If packet matches both primary and auxiliary entries, copy is made.
Controlling the Queue Manager

- All queues are configurable.
  - discard threshold
  - WDRR quota
- Virtual Output Queues (QIDs 504-511)
  - all packets going to switch placed in VOQ for target output
- Datagram output queues (QIDs 440-503)
  - packets going to link with no special queue assignment are hashed to one of these 64 queues
- Reserved output queues (QIDs 256-439)
- SPC queues (QIDs 1-127, 128-255)
  - assigned in pairs \((q, q+128)\)
  - packets to SPC use 1-127
  - packets returning from SPC, going to link use 128-256
FPX Traffic Counters, Status Info.

- Packet and byte counters are read via control cells
  - returned value includes counter value and timestamp
  - timestamps used by software to compute rates
- Port level packet counters
  - received from/sent to link (2 counters)
  - received from/sent to SPC on ingress/egress side (4 counters)
  - received from/sent to router input/output ports (16 counters)
- Packet drop counters
  - ISAR dropped cell counter
  - ISAR invalid packet counter (CRC failure, etc.)
  - QM dropped packet for link queues, switch queues, SPC queues
- And many others,

Selected FPX Counters

- 00 packets from link
- 04 packets from ingress port 0
- 05 packets from ingress port 1
- . . .
- 11 packets from ingress port 7
- 12 ingress-side packets from SPC
- 13 egress-side packets from SPC
- 16 packets to link
- 20 packets to egress port 0
- . . .
- 27 packets to egress port 7
- 28 ingress-side packets to SPC
- 29 egress-side packets to SPC
- 64 ISAR input cell drops
- 65 ISAR invalid packet drops
- 66 QM packet drops for link
- 67 QM packet drops for switch
- 68 QM packet drops for SPC
Smart Port Card

- FPGA routes data straight-thru or to/from SPC.
  - 2 Gb/s data paths
- APIC is Network Interface Chip
  - segments packets into cells on transmission
  - reassembles in memory on reception
- 500 MHz Pentium III processor
  - 100 MHz EDO memory
  - 32 bit PCI at 33 MHz
  - flash disk
  - standard BIOS
- Hosts software plugins
  - options processing
  - application-specific processing

Core ATM Switch

- Recycling Buffer
- Virtual Circuit/Path Lookup Table
- 4 Parallel Switch Planes each cell split into 4 pieces
- Resequencing Buffer
- Dual Priority Transmit Buffer
PVCs for Inter-port Traffic

- Permanent Virtual Circuits carry traffic between FPXs.
- Egress FPX maintains separate reassembly buffers.
- Can use cell counters in switch to monitor traffic.
- Port-level cell counters also available.

Switch Congestion

- Causes
  - switch provides bandwidth of about 2 Gb/s per port
  - so, easy for multiple inputs to overload an output causing congestion in switch and lost packets
  - problem can be exacerbated by fragmentation effects

- Congestion avoidance
  - plan experiments to avoid excessive overloads
  - by default, most link rates are limited to 600 Mb/s to reduce opportunities for congestion

- VOQ rate controls
  - rate limits for virtual output queues can be used to ensure outputs are not overloaded
  - automated configuration of rate limits is planned
    - periodic exchange of VOQ backlog information by SPCs
    - distributed allocation of switch bandwidth
### Testbed Organization

- **Internet**
- **Control Subnet**
- **ONL Server**
- **NetBSD Server** for plugin prep
- **NSP10**, **NSP2**, **NSP3**, **NSP4**

### Configuring Topology

- **Cluster includes router GE switch and fixed set of hosts**
- **Add hosts can as needed.**
- **Drag graphic elements to prettify display.**
- **Port 0 used for Control Processor. Spin handle rotates ports.**
Configuring Topology (cont.)

Add links as needed. These are implemented using configuration switch.

Select “Commit” item to transfer config changes to hardware. Note: first time is slow.

Save config. to a file for use in later session.

Right-click on host to get host name and IP address.

Note color change following commit. Indicates RLI connected to lab hw.
Verifying Host Configuration

Directly connected hosts use ATM interface.

Verify that IP address of interface matches displayed address.

/sbin/ifconfig -a displays info on configured interfaces.

Configuring Routes

Click on port to access route table (and other stuff).

Default routes can be generated for local hosts.

Entry defined by address prefix and mask. Specifies router output port.
What does This Mean for Router?

Route table implemented using space-efficient variant of multibit trie.

192.168.1.16/28 ⇒ 0
192.168.1.32/28 ⇒ 1
192.168.1.48/28 ⇒ 2
192.168.1.64/28 ⇒ 3
192.168.1.80/28 ⇒ 4
192.168.1.96/28 ⇒ 5
192.168.2.0/24 ⇒ 6

Adding More Routes

Causes packets received at port 2 for specified host to be routed thru output 6.
Routes for 2-Way Communication

- Commit routing changes to make effective.
- Second hop of east-bound path.
- First hop of east-bound path.
- Second hop of west-bound path.
- First hop of west-bound path.

Verifying Routes

- Secure shell session to onl19.arl.wustl.edu.
- Ping packets passing through ONL routers.
Monitoring Traffic

Specify monitoring view.

Monitor traffic.

Select desired monitor variable.

Peak per ping packet.

Monitoring Other Data

To add separate chart.

0 for packets from link, 16 for packets to link.

CLICK TO change label.

To select FPX packet counter.

Shows packets/sec for entering/exiting traffic.
Monitoring Still Other Data

- Set focus, so new trace goes here.
- Monitor bandwidth use on virtual circuit entering ATM core.
- Specify target output port.
- New traces from 2 to outputs 6 and 7.

Changing Routes

- Commit route change to make effective.
- Changing next hop to 7 re-routes flow thru bottom link.
- Now see traffic from input 2 to output 7.
- No traffic from input 2 to output 6 or east-bound on top link.
Pause for Live Demo

- Configuring topology
- Configuring routing tables
- Verifying routes using ping
- Monitoring traffic

Using Flow Tables (Exact Match)

- Select ingress filter tables for port 2
- Priority allows flow table entry to override route
- Traffic switches from port 7 to port 6
- Add filter
- Port numbers ignored for ICMP
- Enter 1 for protocol (ICMP)
- Specifies top link
- Bandwidth graph
Using General Filter Tables

- Protocols and ranges may be "don’t-care".
- Bottom link addresses may be specified as prefixes.
- Priority allows filter table entry to override flow table entry.

Using Auxiliary Filter to Copy Flow

- Auxiliary filter replicates data stream.
- Lower priority irrelevant, since auxiliary.
- Flow being sent to both 6 and 7.
- Top east 1.2 to 1.7
  Top west 1.2 to 1.6
Generating Traffic with Iperf

Sample uses
- `iperf -s -u`
  - run as UDP server on port 5001
- `iperf -c server -u -b 20m -t 300`
  - run as client sending UDP packets to server at 20 Mb/s for 300 secs.
- `iperf -s`
  - run as TCP server on port 5001
- `iperf -c server -w 4m -t 300`
  - run as client, sending as fast as possible – make rcv window 4 MB

Using Iperf

Single UDP stream
Multiple Iperf Streams

- Jonathan Turner – 3/14/2005
Displaying Incremental Bandwidth

- Select Add Formula
- Select measures for inclusion
- Resulting formula
- Resulting curve
- Name curve
Displaying Incremental Bandwidth

1. Select Add Formula
2. Name: [-1.3 to 1.6]
3. Select measures for inclusion
4. Resulting formula
5. Resulting curve

Modifying Link Rate

1. Select Queue Tables
2. Modify link bandwidth and commit
3. Total received bandwidth limited
Mapping Flows to Single Queue

Packets from each source mapped to common reserved flow queue.

Monitoring Queue Length

Select Egress Qlength when two or more active flows.
Changing Queue Size

- Adding an entry to the egress queue
- Displaying the queue entry
- Changing the discard threshold and committing
- Entering the number of the queue of interest
- Larger effective buffer

Mapping Flows to Separate Queues

- Varying queue lengths
- Varying queue lengths
Changing Bandwidth Shares

- Variable WDRR (proportional bandwidth allocation)
- Varying quantum
- Packet discards

Changing VOQ Rates

- Select Port 2 Queue Tables
- Changing bandwidth to switch
- Reduced input rate
- Reduced queueing
Using Exact Match Filters

Start long iperf session to run netstat

Interrupt to run netstat

Port numbers

Exact match filter to different queue

Lower priority for general match filter

New queue used

Using Iperf with TCP

Start TCP sender

Interrupt with ctrl-Z

Uses all link bandwidth

Uses most of 20 KB queue
Competing TCP Flows

- senders adjust rate to match available bandwidth
- queue size increased by 100x, but only small increase in queueing
- using most of available queue space (20KB, 30KB, 40KB)
- slightly better match of sending/receiving rates

Pause for Live Demo

- Using Iperf with UDP
- Using filters
- Monitoring queues
- Using Iperf with TCP