The Open Network Lab

Ken Wong
Applied Research Laboratory
Computer Science and Engineering Department
http://www.arl.wustl.edu/~kenw
kenw@arl.wustl.edu
http://www.onl.wustl.edu (ONL)
National Science Foundation ANI-023826, CNS-0551651, REL-0632580

ONL Lab Overview

- Gigabit routers
  - easily configured thru Remote Lab Interface
  - embedded processors for adding new features
- PCs serve as hosts
  - half on shared subnets
- Net configuration switch
  - link routers in virtual topologies
  - traffic generation
- Tools for configuration and collecting results
  - monitoring traffic
  - data capture and playback
- Open source
  - all hw & sw sources on web
People Who Make it Happen

- Jon Turner
  Principle Investigator
- Ken Wong
  Admin, Web site, Dist. Sched.
- Jyoti Parwatikar
  RLI, Software development
- Charlie Wiseman
  Web site, Ops Dist. Sched.
- Fred Kuhns
  SPC software FPX hardware
- John Dehart
  FPX hardware System integration

Instructional Objectives

- Deepen understanding of networking
  » *Fundamentals*: routing, queueing, pkt classification, error and congestion control
  » *Advanced*: pkt scheduling, protocols, services (e.g., guaranteed)

- Develop experimental CS skills
  » measurement and data analysis
  » selection of exploration trajectories

- Experience with adv. network technology
  » distributed router with distributed scheduler
  » router plugins
  » network processors (to come)
Sample ONL Session

- Network Configuration
- Routing Tables
- Queue Length
- Bandwidth Usage
- Queue Parameters

ONL Hardware

- Gigabit Router
- Smart Port Card
- Field Prog. Port Extender
Testbed Organization

Testbed Organization

Major Software Components

Getting Started

- Ken Wong– 9/13/2006

**tutorial**
- [get an account](onl.arl.wustl.edu, onl.wustl.edu, www.onl.wustl.edu)

After Logging in

- Ken Wong– 9/13/2006

**extra links**
- [getting started](onl.arl.wustl.edu, onl.wustl.edu, www.onl.wustl.edu)
- [testbed status](onl.arl.wustl.edu, onl.wustl.edu, www.onl.wustl.edu)
- reservations

**download**
- Remote Lab Interface Software

**install**
- Java runtime environment

**verify**
- that you can SSH to onl.arl.wustl.edu

**configure**
- SSH tunnels
SSH Tunnel Configuration

- Unix or Unix-like (e.g., cygwin)
  » `ssh -L 7070:onlsrv:7070 onl.arl.wustl.edu`
  » I use this (defined as an alias) from Linux

- MS Windows PuTTY (GUI-based)
  » I use this from MS XP laptop
  » See Tutorial for URL (free software)

- MS Windows SSH client tool from SSH Corp

- Follow instructions on ONL web pages
  » 100s have successfully installed the tunnel
  » But don’t agonize over it
  » If problems, send email to your grader/consultant

PuTTY SSH Tunnel Configuration

- local port 7070
- remote host onlsrv
- remote port 7070
- session name = rli
- external host name = onl.arl.wustl.edu
Topology Demo 1/2

- Topology
- Add Cluster
- Add Host
- Generate Default Routes
- Spin handle and Port 0

Configuring Topology

Cluster includes:
- NSP router,
- GE switch,
- fixed set of hosts

Add Cluster
Add Host
Add Link
Generate Default Routes

Port 0 used for Control Processor. Spin handle rotates ports.

Add hosts and links as needed.

Drag icons to improve visual layout.

For all ports

Commit Needed
No Status

Actual hardware NOT been allocated until after commit.
Save Configuration

You do NOT need to be connected to the ONL testbed until you commit!!

Resource Reservation

Make reservation

Current reservations
Commit and Testbed Status

Allocate hardware and configure

Commit and Testbed Status

Commit Done

Can SSH to control net hostname from onlusr (no password)

Logical names bound to actual hardware

Dark links: commit done
Verifying Host Configuration

- `nestat -i:`
  - Display interface summary

- `ifconfig eth0`
  - Display interface config info

MAC address:
- IP address of testbed interface

Internal Interface IP Addresses

- **Control Network:** 10.0.0.X
- **Data Network:** 192.168.N.Y
  - \( N = \text{NSP#} \)
  - \( Y = 16 \times (1 + \text{port#}) \)
**Default Route Tables**

CIDR network addresses

RTs are really Forwarding Tables

Select Port 2 => Route Table

**Semantics of Route Table Entry**

RT implementation uses space-efficient variant of multibit trie.
Packet Processing Example

- Packet Scheduler (PS)
  - Weighted Deficit Round Robin (WDRR) or Distributed Queueing (DQ) on ingress side
  - WDRR on egress side
- Shim is a special header
  - Contains internal metadata

How Fast Can a Queue Overflow?

- Given
  - Q = 250,000-byte queue
  - All pkts are L = 1,500 bytes long
  - Average input traffic rate $R_{in} = 500$ Mbps
  - Link rate $R_{out} = 1$ Mbps

- Fluid Model
  - Queue Backlog $B(t)$
  - Input Volume $V_{in}(t)$
  - Output Volume $V_{out}(t)$

- Cases
  - Constant (deterministic) input rate
  - Deterministic, bursty input rate
  - Stochastic (Poisson) input rate
Pause for Topology Demo 2/2

- Simple traffic monitoring
- Add links

Add Monitoring Display

chart name

poll every 0.3 sec
Ping Chart

- Select max y-value = 0.004
- Window width halved

Adding A Link

- Drag from port 7 to port 6
Delete Route Group

1. Press `shift+select` to highlight the routes you want to delete.
2. Press `route` to delete the highlighted routes.

Add Route To Loopback

1. Select the loopback address for which you want to add a route.
2. Use the `match` command to specify the criteria for adding the route.
3. Use the `route` command to add the route.
Pause for Traffic Demo 1/2

- Traffic monitoring
- iperf traffic generator

Bandwidth Monitoring
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 -w 4m`
  - run as TCP server on port 5001
  - set max window to 4 MB
- `iperf -c server -w 4m -t 300`
  - run as client, sending as fast as possible, with max window 4 MB

iperf Traffic Generator

Start UDP server (receiver)

Start client (sender)

200 Mbps bw for 10 sec

Client sending rate

Server receive rate (bottleneck)
Built-in Monitoring Points

- Easy to monitor traffic using real-time displays
- Built-in monitoring points
  - Link bandwidth
  - Port-to-port switch bandwidth
  - Queue lengths
  - Packet loss
  - and much more
- Can use same monitoring interface to display data generated by
  - End hosts (e.g., TCP window size)
  - Software plugins installed at router ports

Pause for Traffic Demo 2/2

- Link rate (capacity -- Mbps)
- General match filters
- Queue size (capacity -- bytes)
- Emulating propagation delay (delay plugin)
Some Bottleneck Experiments

- **Configuration**
  - Senders: n1p2, n1p3
  - Receivers: n1p4, n1p5
  - Bottleneck link 1.7-1.6 (300 Mbps)
  - Queue 300: 150,000 bytes
  - Forward path through 1.7-1.6
  - Return path through 1.6-1.7

- **Flows**
  - 8 second staggered starting time (shell script)
  - Use `iperf` traffic generator
  - Two 200 Mbps UDP flows through bottleneck
    - Pkt scheduling: Equal rate vs. 1:3 rate
  - Two TCP flows through bottleneck
    - With and without 50 msec ACK delay

Configuring For UDP Experiment

- At egress port 7 (the bottleneck)
  - Install GM filter to direct traffic to queue 300
  - Configure queue 300 to be 150,000 bytes
- Monitor **VOQ bw, queue 300 length, and pkt drops**
Monitoring Points

Queue Table
Adding a General Match Filter

- GM filter
  - matches all pkts
  - protocol (*) matches any protocol
  - Queue 300
  - Priority 50 (higher than RTs 60)
- Queue 300
  - 150,000 bytes
- Egress link rate = 300 Mbps

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 – general match
    - (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 *(best match)* – 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 match)* – ingress or egress
  - 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 of pkt is made.

Iperf Server And Client Scripts

```bash
#!/bin/bash

# Start udp clients
for i in 1 2 3 4 5; do
  ssh $n1p$i /usr/local/bin/ipercf -c n1p$i -u -b 200m -t 20s
done

# Start udp server
iperf -s
```

Puts $n1p2, etc. into Linux environment

Run "iperf -s -u" on hosts $n1p4 and $n1p5

Run as UDP server

Run as UDP client sending to n1p4

Send at 200 Mbps for 20 seconds
Running Receiver and Sender Scripts

Run receiver script

Run sender script

Sending (client) rate and receive (server) rate for two flows

VOQ Bandwidth and Q300 Length

- Both senders send at 200 Mbps
  - 1.2 to 1.7 and 1.3 to 1.7
- Only n1p2 traffic
  - 200 Mbps goes to n1p4
- n1p2 and n1p3 traffic
  - 120 Mbps to n1p4 (1.6-1.4)
  - 180 Mbps to n1p5 (1.6-1.5)
- Queue 300 is full
- Only n1p3 traffic
  - 200 Mbps goes to n1p5
Separate Queues For Each Flow

- 150 Mbps each flow
- Q300 and Q301 filled

TCP Scripts

- 4 MB receiver window to handle bandwidth-delay product
Both TCP Flows Compete Equally

Delay ACK Pkts By 50 Msec

- Define GM filter at egress port 6
  - matches all pkts
  - directs traffic to SPC delay plugin
- Define standard delay plugin
  - default delay is 50 msec
  - change delay thru msg interface
- Binding between filter and plugin
  - SPC QID (a value between 8 and 127)
Working Smarter

- Tutorial ➔ Summary Information ➔ Password-Free SSH
- Traffic generation shell scripts
  - e.g., ~kenw/bin/{[ut]sndrs-1nsp,[ut]rcvrs-1nsp}
- Reference Web pages
  - Tutorial ➔ Summary Information
- FAQs by category (to come)

Open Network Laboratory

- Web Site
  - onl.arl.wustl.edu, www.onl.wustl.edu, onl.wustl.edu
  - Look at tutorial
  - Look at video
  - Register and try it out
- Lab Questions?
  - See your grader(s) and instructor
- ONL Problems?
  - Verify that the problem persists, then:
    - Email testbed-ops@onl.arl.wustl.edu
    - Give concise description of problem
    - Give instructions for reproducing problem
    - It would help if you can reduce the experiment to the simplest case