

# ONL NP Router Plugin Tutorial

*This document was created by Patrick Crowley  
([pcrowley@wustl.edu](mailto:pcrowley@wustl.edu)) and last modified on 3/27/2008.  
All comments, suggestions and corrections are welcome.*

# ONL NP Router Demo

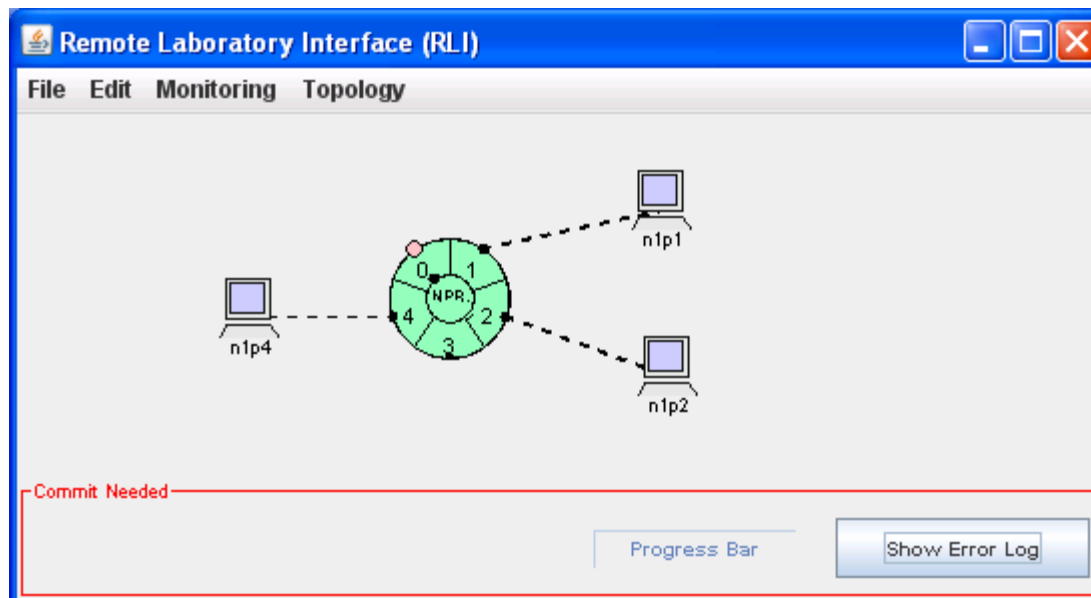
- Running the RLI
- Building a topology & routes
- Installing a plugin
  - Configuring a filter
- Generating simple traffic
- Building a new plugin
- Using Debug messages

# Getting Started

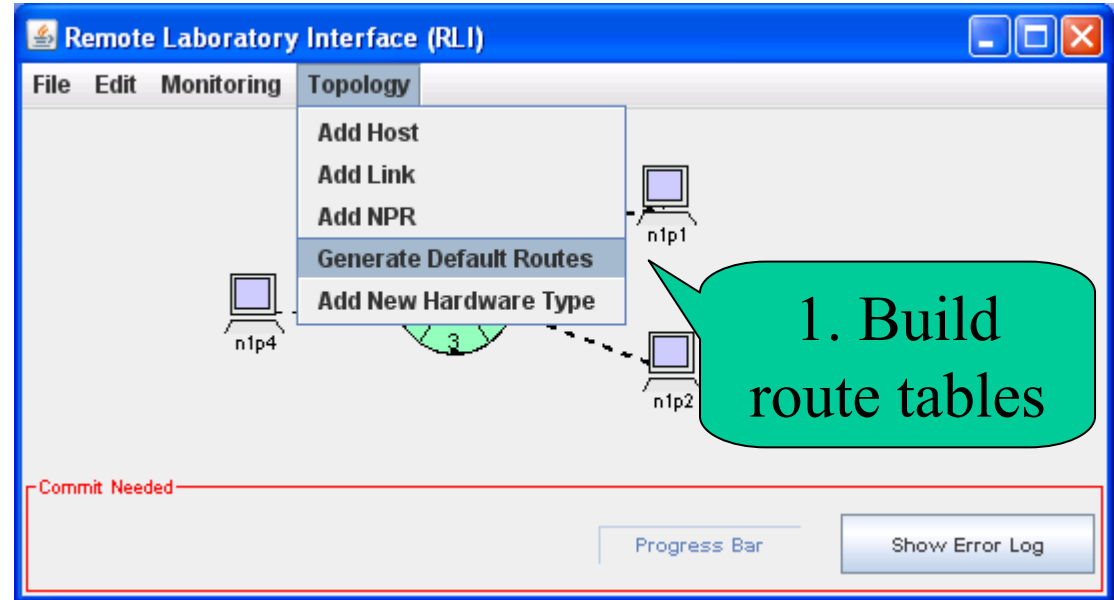
- Visit <http://onl.wustl.edu>
- You need:
  - An account
  - The RLI
  - An SSH client
- The site has ample installation instructions

# Build A Topology

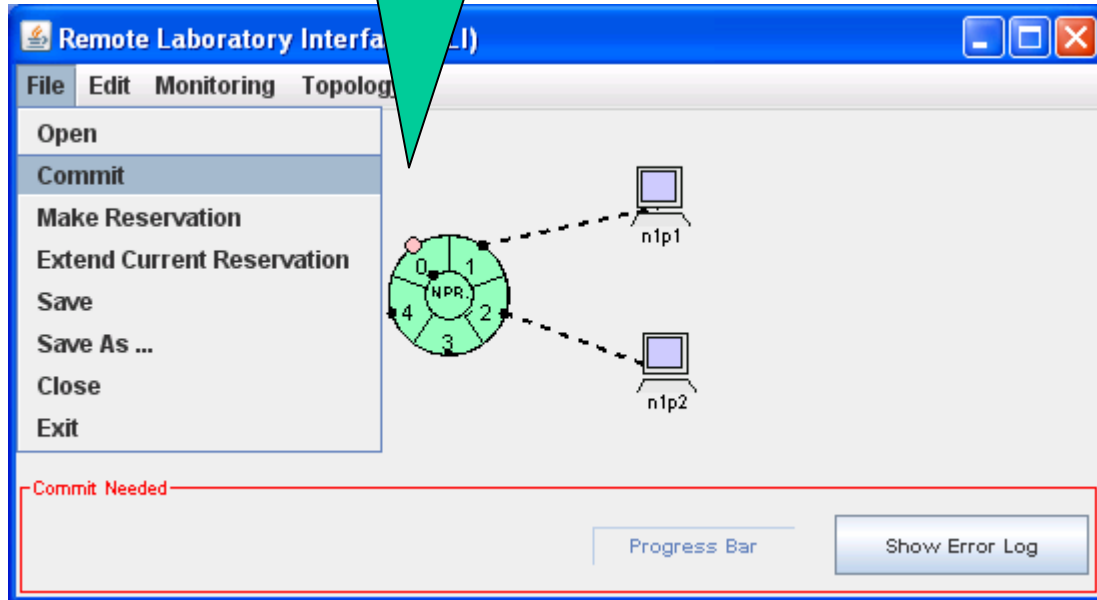
- Construct the topology below
- You can save it (File→Save As) for re-use later



# Build Routes & Commit



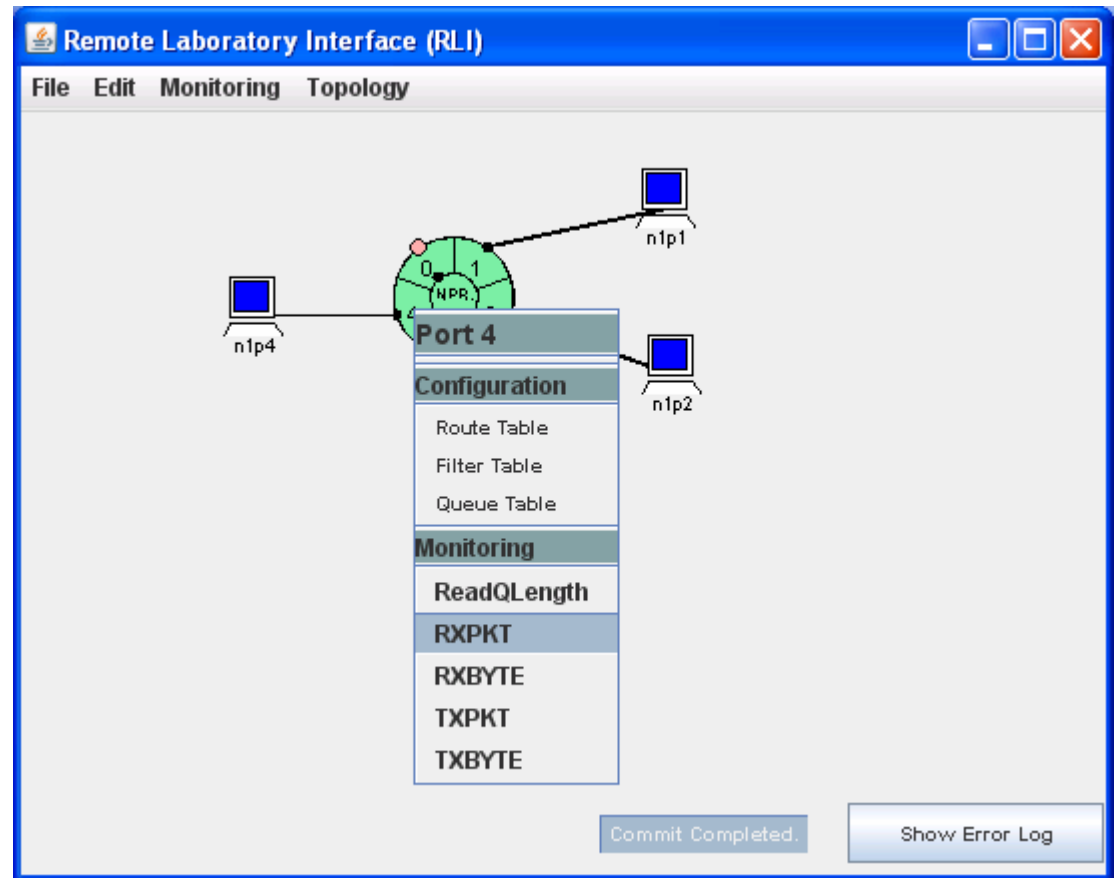
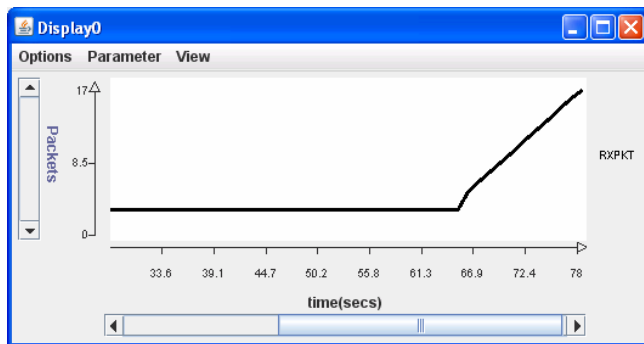
2. Commit to "go live"



You need a reservation before you can commit. You can get one at the web site or through the RLI.

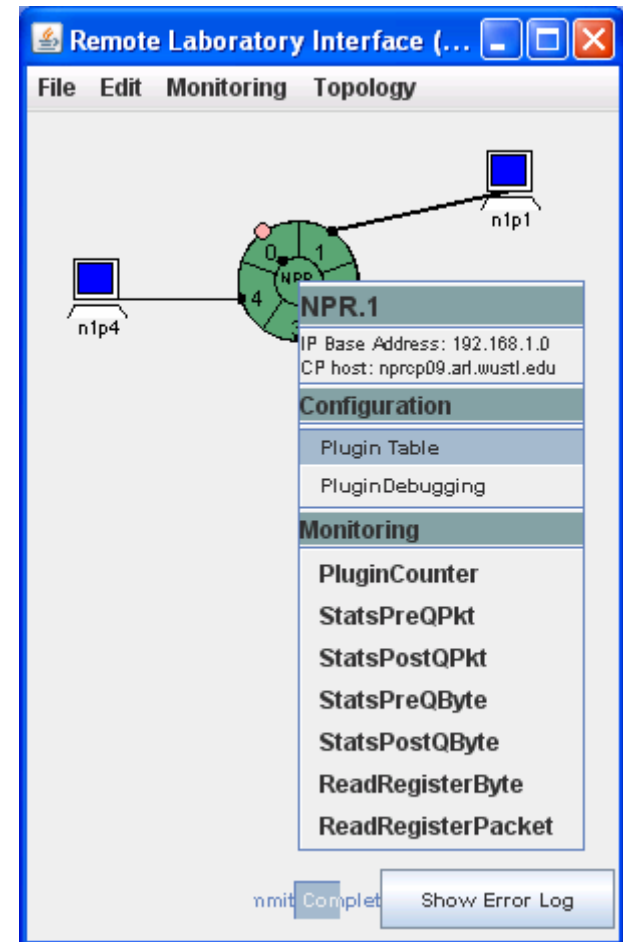
# Sanity Checks

- Once you have committed, it's a good idea to confirm that you can send and receive traffic
- Use a pair of SSH consoles to ping between hosts
  - SSH from onluser to n1p4 & n1p1
    - “source .topology”
    - “ssh \$n1p4”
    - “ssh \$n1p1”
  - On n1p4
    - “ping n1p1”
  - Monitor a RXPKT counter on port 4



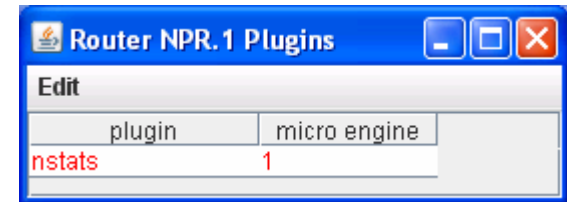
# Installing a Plugin

- There are two steps to using plugin
  - Install the plugin
  - Install a filter to direct packets to it
- To start, click the router's center circle
  - Choose “Plugin Table”



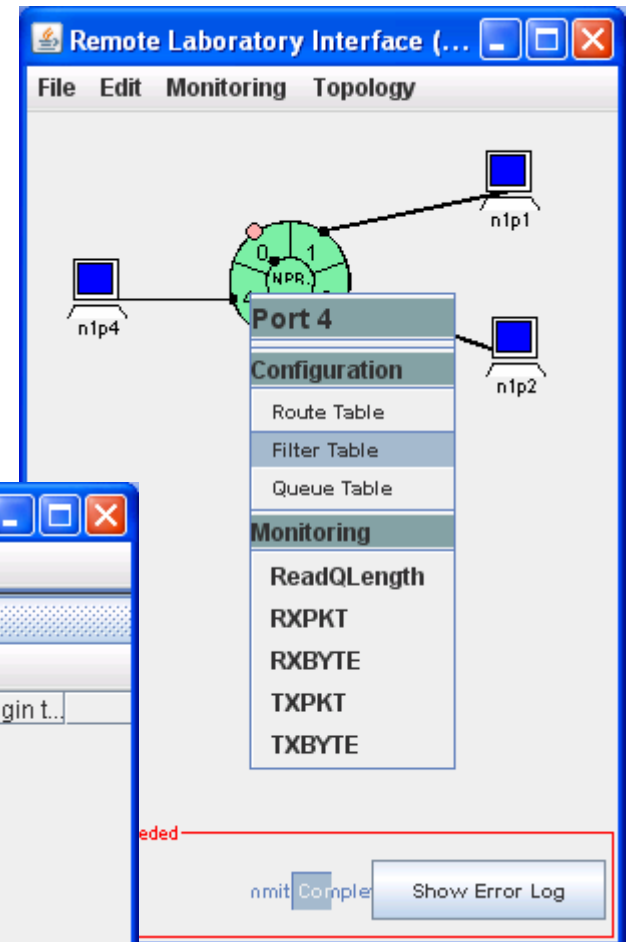
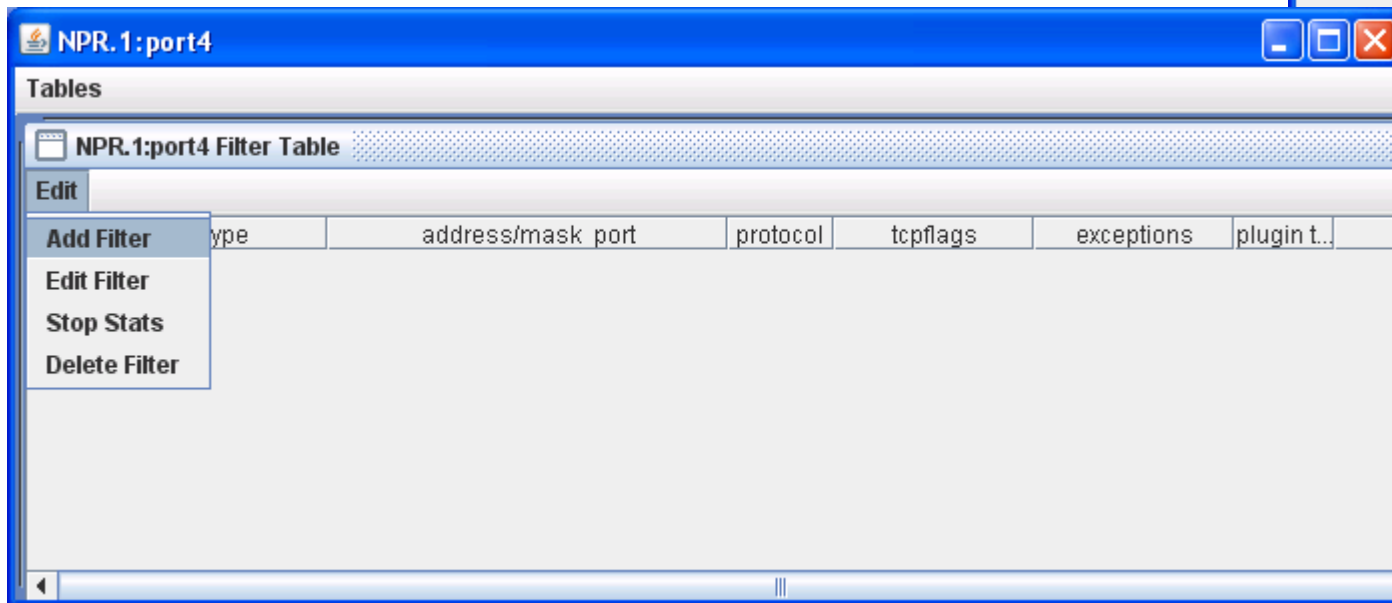
# Installing a Plugin (2)

- In the Plugin Table dialog, choose “Edit → Add Plugin → nstats”
- The plugin will be mapped to an available plugin ME, between 0 and 4



# Add a Filter

- Click port 4
- Choose “Filter Table”
- Choose “Edit → Add Filter”



# Add a Filter (2)

- Use the options shown
- Pay attention to:
  - Non-IP, ARP, IP Opt, & TTL
  - aux
  - port/plugin selection
  - output ports
  - output plugins

The screenshot shows the 'Add Filter' dialog box with the following configuration:

- source address/mask: 0.0.0.0/0
- source port: \*
- destination address/mask: 0.0.0.0/0
- destination port: \*
- protocol: \*
- tcpflags: tcpfin, tcpsyn, tcprst, tcppsh, tcpack, tcpurg
- exceptions: Non-IP (0), ARP (0), IP Opt (0), TTL (0)
- aux  multicast
- port/plugin selection: plugins only
- output ports: 1
- output plugins: 1
- qid: 0
- plugin tag(0-31 or \*): 0
- priority(0-63): 50
- sampling type (aux only): 100%

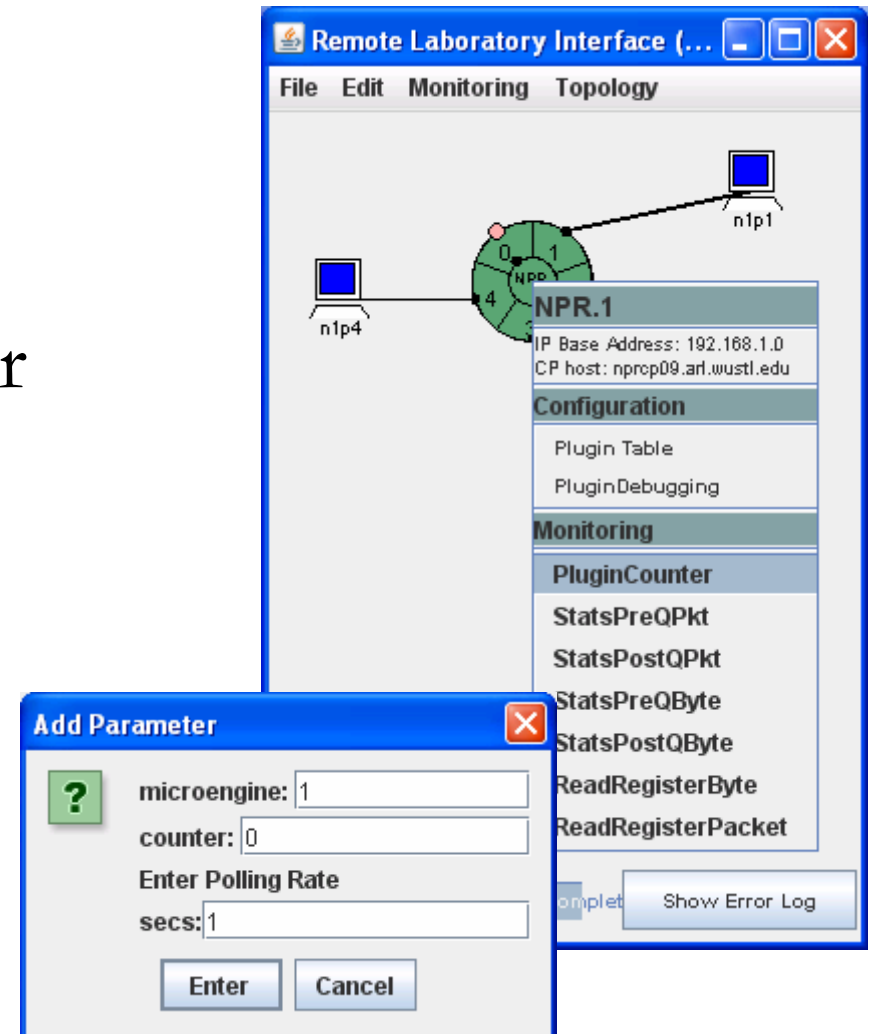
Buttons: Add, Cancel

# Commit Changes

- We have added a plugin and a filter
- “File → Commit” will update the router

# Monitor Plugin Counters

- Click the center circle and select “PluginCounter”
- Specify the ME, counter pair you want to monitor
- nstats has been mapped to plugin ME 1 and uses counters 0, 1, & 2 to count ICMP, TCP, & UDP, resp.
- Repeat to add them all



# Viewing nstats counters

- Generating traffic

- ICMP:

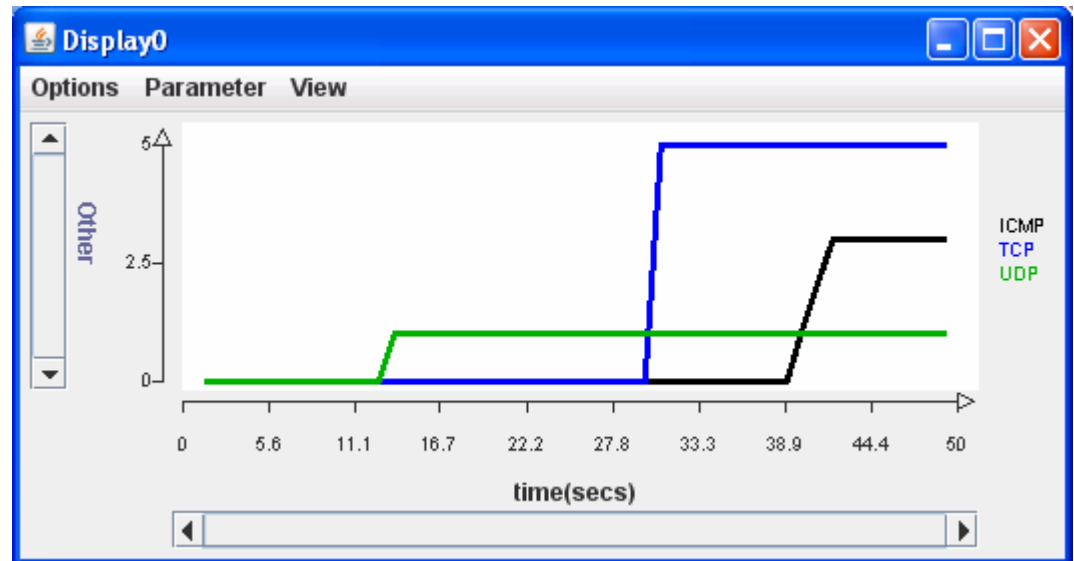
- n1p4: “ping n1p1”

- TCP:

- n1p1: “python receiver.py”
- n1p4: “python sender.py n1p1”

- UDP

- n1p1: “python udp\_receiver.py”
- n1p4: “python udp\_sender.py n1p1”



# Python UDP Example

## udp\_sender.py

```
import socket, sys

if len(sys.argv) > 1:
    dst = sys.argv[1]
else:
    dst = 'localhost'

# Create a socket
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# Open a connection to the local machine at port 7777
s.connect((dst,7777))
# Send some bytes through the connection.
s.send("Mr. Watson--come here--I want to see you.")
s.close()
print "Message sent."
```

# Python UDP Example (2)

## **udp\_receiver.py**

```
import socket
# Create a socket
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# Bind to port 7777 on the local machine
s.bind(('',7777))

# Dump data 1KB at a time
while 1:
    data, addr = s.recvfrom(1024)
    if not data: break
    print "From ", addr, ": ", data
s.close()
```

# Python TCP Example

## sender.py

```
import socket, sys

if len(sys.argv) > 1:
    dst = sys.argv[1]
else:
    dst = 'localhost'

# Create a socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Open a connection to the local machine at port 7777
s.connect((dst,7777))
# Send some bytes through the connection.
s.send("Mr. Watson--come here--I want to see you.")
s.close()
print "Message sent."
```

# Python TCP Example (2)

## receiver.py

```
import socket
# Create a socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Bind to port 7777 on the local machine
s.bind(('',7777))
# Listen for new connections
s.listen(1)
# Grab connection object and IP address of sender
conn, addr = s.accept()
print "Connected from: ", addr
# Dump data 1KB at a time
while 1:
    data = conn.recv(1024)
    if not data: break
    print data
conn.close()
```

# Building a Plugin

1. Create a “myplugins” directory in your ONL directory
2. Copy an existing plugin

```
[pcrowley@onlusr myplugins]$ cp -r /users/onl/npr/plugins/count/ .
```

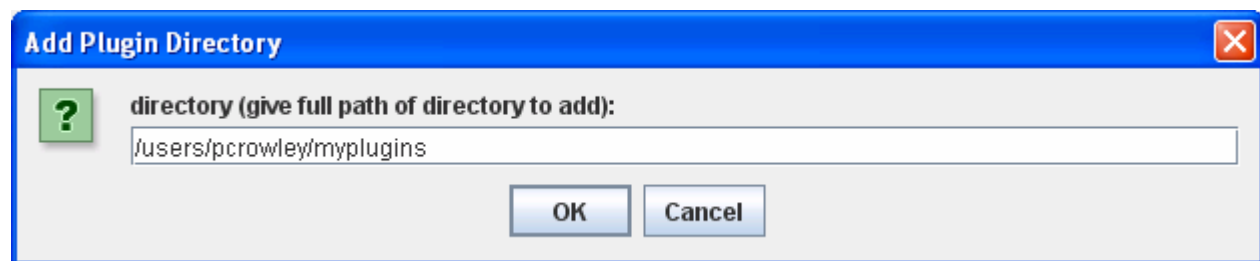
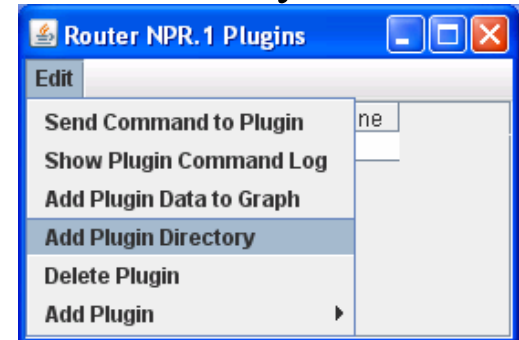
3. Change plugin name
  - Rename directory, “mycount”
  - In Makefile, rename plugin to “mycount”

# Building a Plugin (2)

4. Change count.c source
  - For example, in `handle_pkt_user()` repeat the counter increment line 9 more times for a total of 10 increments for each packet arrival
5. “make clean”
6. “make”

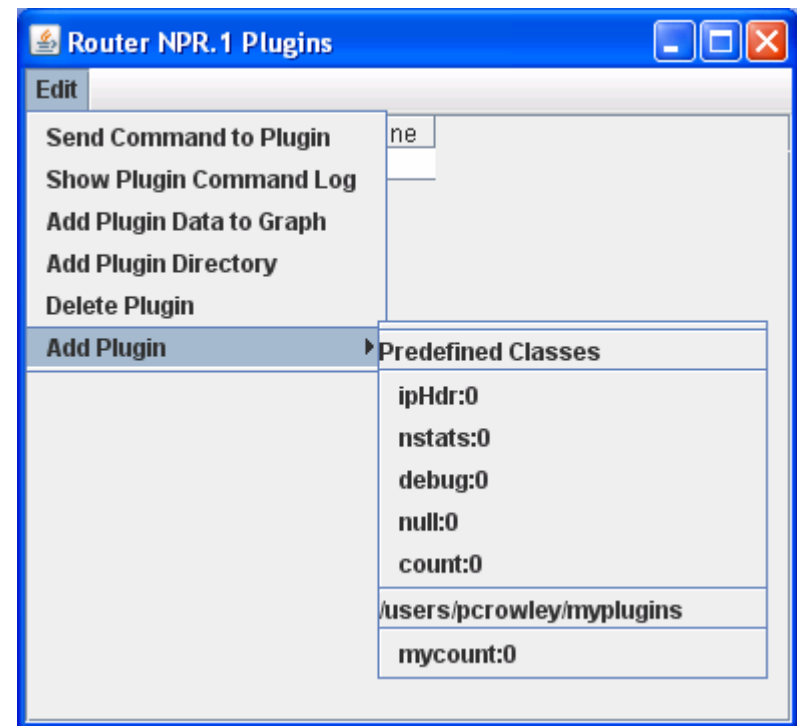
# Installing Your Plugin

- You must first tell the RLI where to find your plugins
  - Open the Plugin Table
  - “Edit → Add Plugin Directory”
- Use “pwd” on an ONL host to find and enter your fully-specified plugin directory



# Installing Your Plugin (2)

- Your plugins can now be installed like any others
- Be sure to add and monitor counters to make sure your changes took effect



# Debug Messages

- The plugin API allows you to write data into a text file in your local directory
- The “debug” plugin gives an example of this
- After you have installed the plugin, you must also specify the name of the file to receive the debug messages
- The file you specify will be created in your root directory

