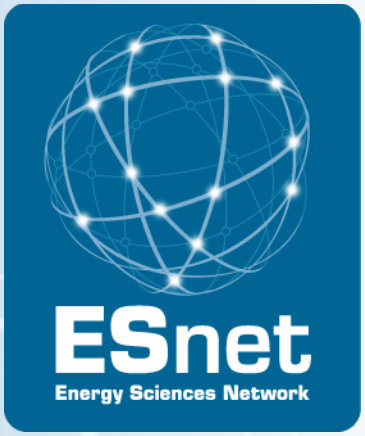



# Using TCPDump, TCPTrace, & XPlot to Debug Network Problems

Jason Zurawski - **ESnet Science Engagement**

http://fasterdata.es.net

October 16th, 2013

perfSONAR powered

U.S. DEPARTMENT OF ENERGY
Office of Science

BERKELEY LAB

With contributions from S. Balasubramanian, E. Dart, B. Johnston, A. Lake, E. Pouyoul, L. Rotman, B. Tierney and others @ ESnet

# Agenda

- Introduction
- TCPDump – Isolate & Capture Traffic
- TCPTrace – Analyze Captured Traffic
- XPlot – Graphical Display of Flows
- Example of Use
- Real Example (Performance Through a Firewall)

# Introduction

The TCP protocol hides performance details from the user

- Unless the Kernel is instrumented with Web100/ Web10G, getting a per flow record of occurrences such as retransmissions and timeouts is not readily available

There are two methods to learn more about the TCP stream of an application:

- ***Application Instrumentation*** – internal record keeping
- ***Passive Observation*** – from external applications

perfSONAR powered

# Introduction

## Goals:

- Record of key metrics related to TCP
- Convert into statistics about a flow (e.g. count of packet types, time spent in sending vs stalled state, etc.)
- Graphical representation to illustrate performance over time
- Aid in debugging end to end behaviors of a target use/ application

## Tools:

- TCPDump
- TCPTrace
- XPLot

perfSONAR powered

# Agenda

- Introduction
- TCPDump – Isolate & Capture Traffic
- TCPTrace – Analyze Captured Traffic
- XPlot – Graphical Display of Flows
- Example of Use
- Real Example (Performance Through a Firewall)

# TCPDump

Application designed to capture packets (headers normally, or with certain options entire data streams)

Designed to run on a target interface, with an expression that matches a specific pattern:

- E.g. *capture all traffic on interface eth0 destined for subnet 192.168.1.0/24*
- or *capture all traffic destined for port 5001*

Goal is to capture all behavior

- Can isolate specific flows to a src/dst pair
- Can isolate specific applications if they are using specific port profiles

perfSONAR powered

# TCPDump

Note: Will require interface to enter 'promiscuous' mode.

- SELinux may not like this
- Firewall/IDS on machine may not like it either
- Typically need to be root.

Common Options:

- -A: ASCII output of packets (good if you are inspecting data contents)
- -c: Exit after 'c' packets (rather than waiting for SIGTERM)
- -I $INTERFACE: Specific interface to listen on (vs entire machine)

perfSONAR powered

# TCPDump

## Common Options (cont.):

- -n: Don't convert addresses to names (e.g. useful if you want to see raw IPv4, IPv6 addresses, and ports)
- -w $FILE: Write output to $FILE instead of STDOUT. You will want to use this option if you plan to process the packets through TCPTrace
- -x: Print packet data (in HEX). Add another to see link layer header.
- -X: Print packet data (in HEX and ASCII). Add another to see link layer header.

# TCPDump

## Invocation:

- **`sudo tcpdump –i eth0 192.168.0.1`**
  - or
- **`sudo tcpdump –i eth0 host 192.168.0.1`**
  - Capture traffic from specific host (incoming and outgoing) on target interface.
- **`sudo tcpdump –i eth0 net 192.168.0`**
  - Capture traffic from /24 subnet (incoming and outgoing) on target interface.
- **`sudo tcpdump –i eth0 port 5001`**
  - Capture traffic from specific port (incoming and outgoing) on target interface.
- **`sudo tcpdump –i eth0 portrange 100-200`**
  - Capture traffic from specific ports (incoming and outgoing) on target interface.

perfSONAR powered

# TCPDump

## Invocation (cont.):

- **`sudo tcpdump –i eth0 src 192.168.0.1`**
  - Capture traffic where host is the src, on specific interface
- **`sudo tcpdump –i eth0 dst net 192.168.0`**
  - Capture traffic where /24 subnet is the dst, on specific interface
- **`sudo tcpdump –i eth0 src or dst port 5001`**
  - Capture traffic where port 5001 is the destination, on specific interface
- **`sudo tcpdump –i eth0 tcp port 5001`**
  - Capture traffic where port 5001 is used for TCP protocol only, on specific interface

## Read man pages for more information

perfSONAR powered

# Agenda

- Introduction
- TCPDump – Isolate & Capture Traffic
- TCPTrace – Analyze Captured Traffic
- XPlot – Graphical Display of Flows
- Example of Use
- Real Example (Performance Through a Firewall)

# TCPTrace

TCPDump captures packets according to specific filters, TCPTrace will analyze the gathered data

- Processes entire dump file
- Splits contents into specific flows (e.g. specific src/dst/ port pairs based on dump contents). Multiple flows per file is possible
- Outputs statistics
- Produces XPlot ready graphs

Goal is to organize the haphazard nature of a dump file into specific interactions

- packets arrive in pseudo-order for a given flow, multiple flows in the same file may interleave
- Track individual operations (e.g. sending packets in a TCP window, tracking timers, retransmissions, duplicate ACKs, etc.)

perfSONAR powered

# TCPTrace

Common Options:

- -b: Brief output
- -l: Long Output (suggested)
- -W: Report on congestion window
- -G: Output all graphs (recommended, although the TSG and TLine are the most commonly used)

# TCPTrace

## Invocation:

- **`tcptrace –lW ~/iperf.dmp`**
  - Output long analysis  with congestion window information for target dump file
- **`tcptrace –G ~/iperf.dmp`**
  - Generate all graphs for target dump file
  - Note there are a couple of graph types:
    - Time Sequence Graph (TSG)
    - Throughput Graph (TPUT)
    - RTT Graph (RTT)
    - Outstanding Data Graph (OWIN)
    - Segment Size Graph (SSIZE)
    - Time-Line Graph (TLINE)
  - http://www.tcptrace.org/manual/index.html

## Read man pages for more information

perfSONAR powered

# Agenda

- Introduction
- TCPDump – Isolate & Capture Traffic
- TCPTrace – Analyze Captured Traffic
- XPlot – Graphical Display of Flows
- Example of Use
- Real Example (Performance Through a Firewall)

# XPlot

Visualization tool for plotting complex data sets.

- Supports multiple plots on a single graph
- Color or mono options

Simple interface to allow click/drag zooming over interesting regions.

Invocation:

- **`xplot graph.xpl`**
  - Display graph generated from tcptrace ('-G option over target dumpfile').
  - The TSG (Time Series Graph) is the most useful to see events over time
- **`xplot -y 'yrange' graph1.xpl graph2.xpl`**
  - Open 2 graphs, use a common 'y' axis (assists with seeing a relative slope for 'throughput'

Read man pages for more information

perfSONAR powered

# Agenda

- Introduction
- TCPDump – Isolate & Capture Traffic
- TCPTrace – Analyze Captured Traffic
- XPlot – Graphical Display of Flows
- Example of Use
- Real Example (Performance Through a Firewall)

# Example Use Case

Select 2 Hosts

Run “iperf” server on the first, client on the second

Run “tcpdump” on the client side or server side (its often good to do both, and compare)

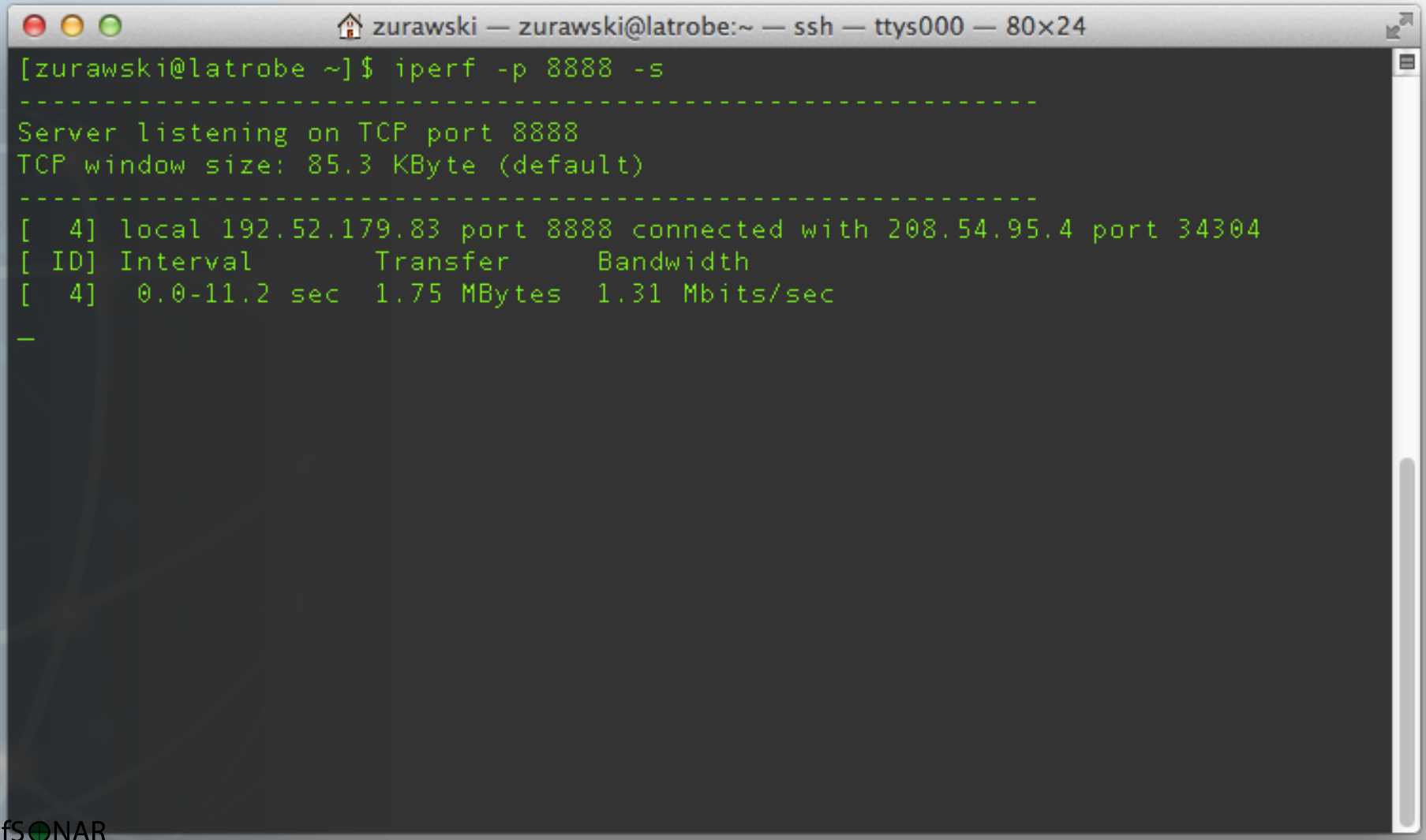
Stop “tcpdump” after iperf session has completed

Run “tcptrace -G dumpfile.dmp” over the collected “tcpdump” data

Use “xplot” to view TSG (Time Series Graph) and TLine (Time Line Graph)

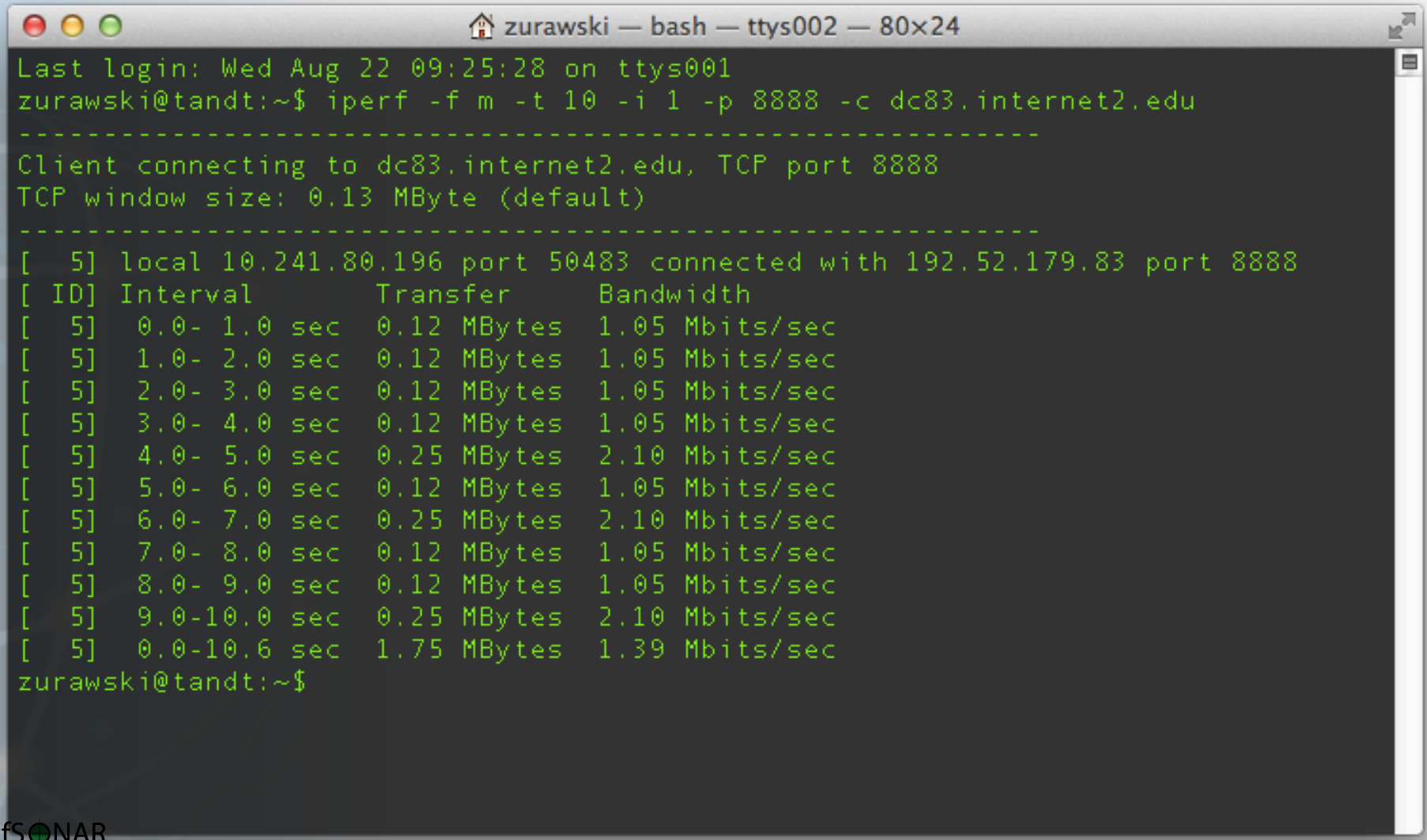
perfSONAR powered

# Example Use Case – Iperf Server



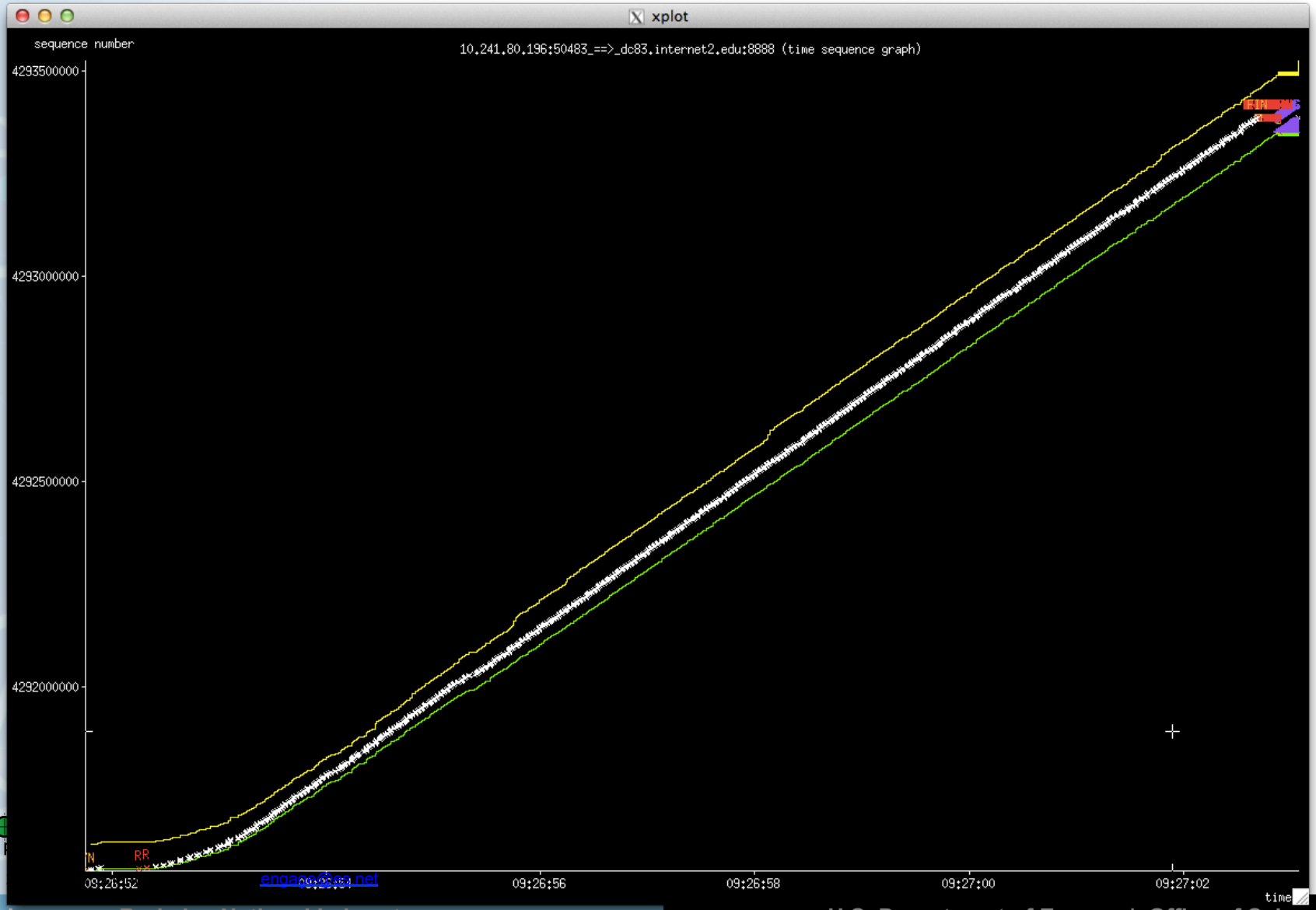

perfSONAR powered

# Example Use Case – Iperf Client

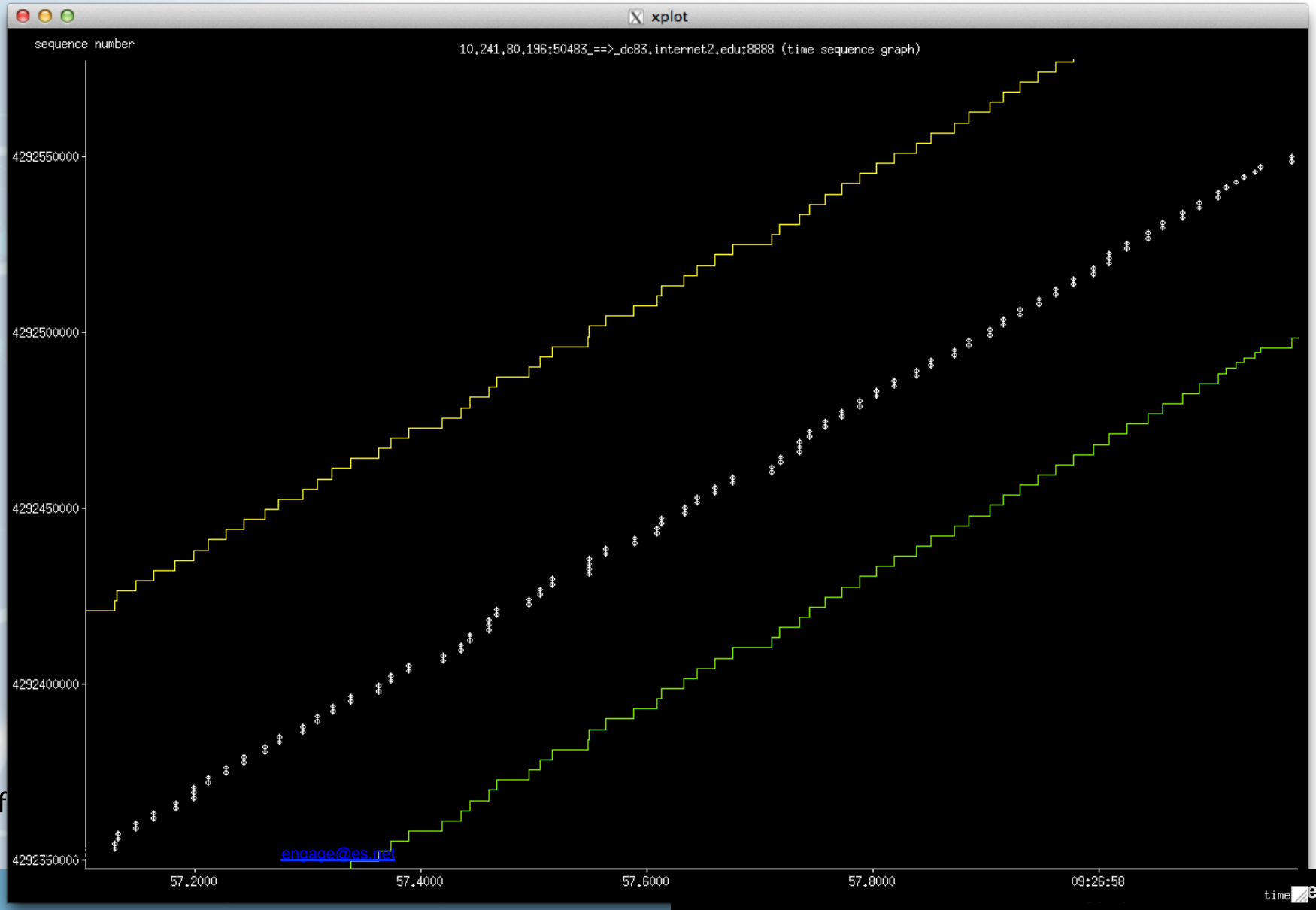



perfSONAR powered

# Example Use Case – XPlot (TSG)

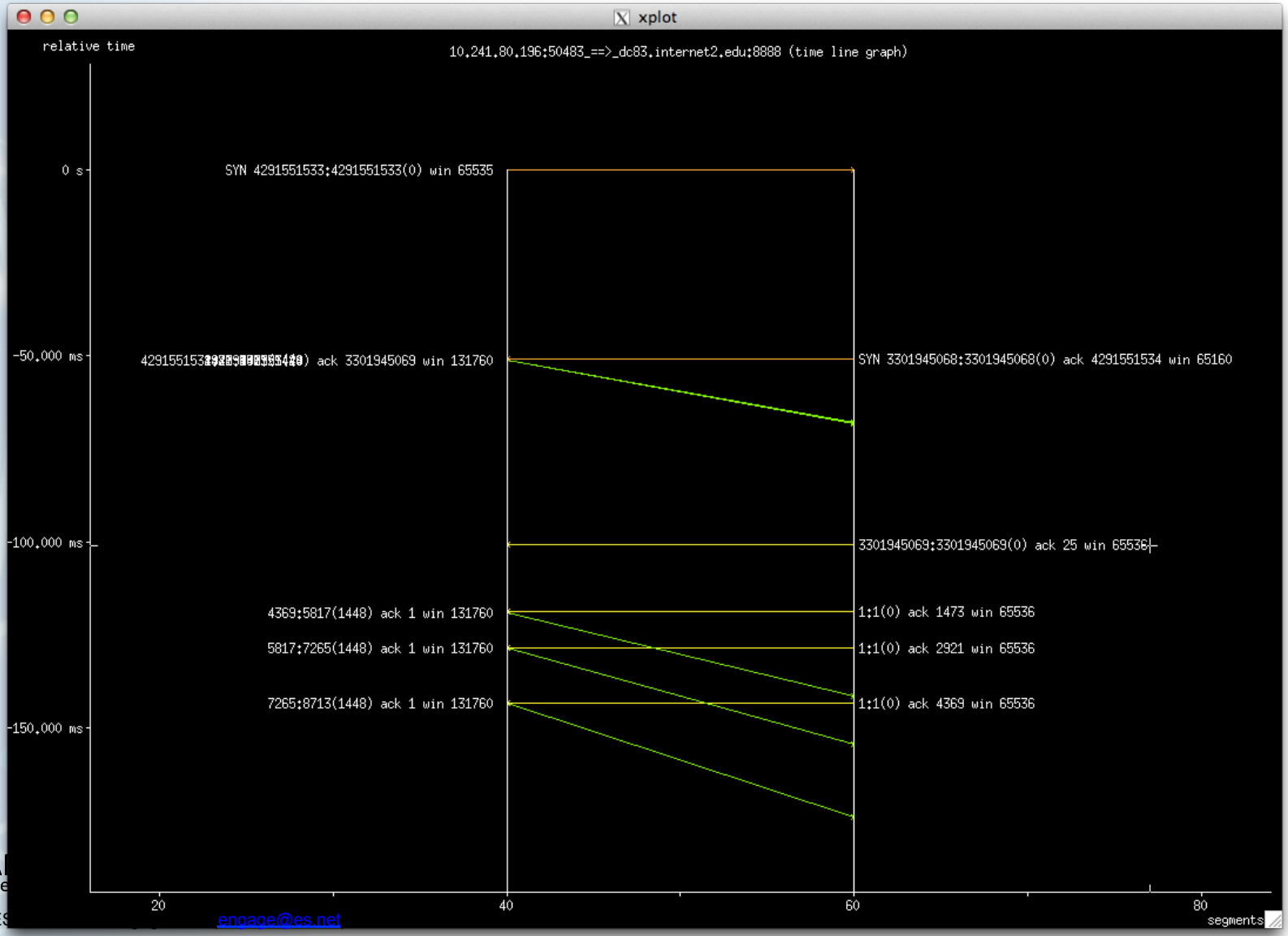


# Example Use Case – XPlot (TSG - zoom)

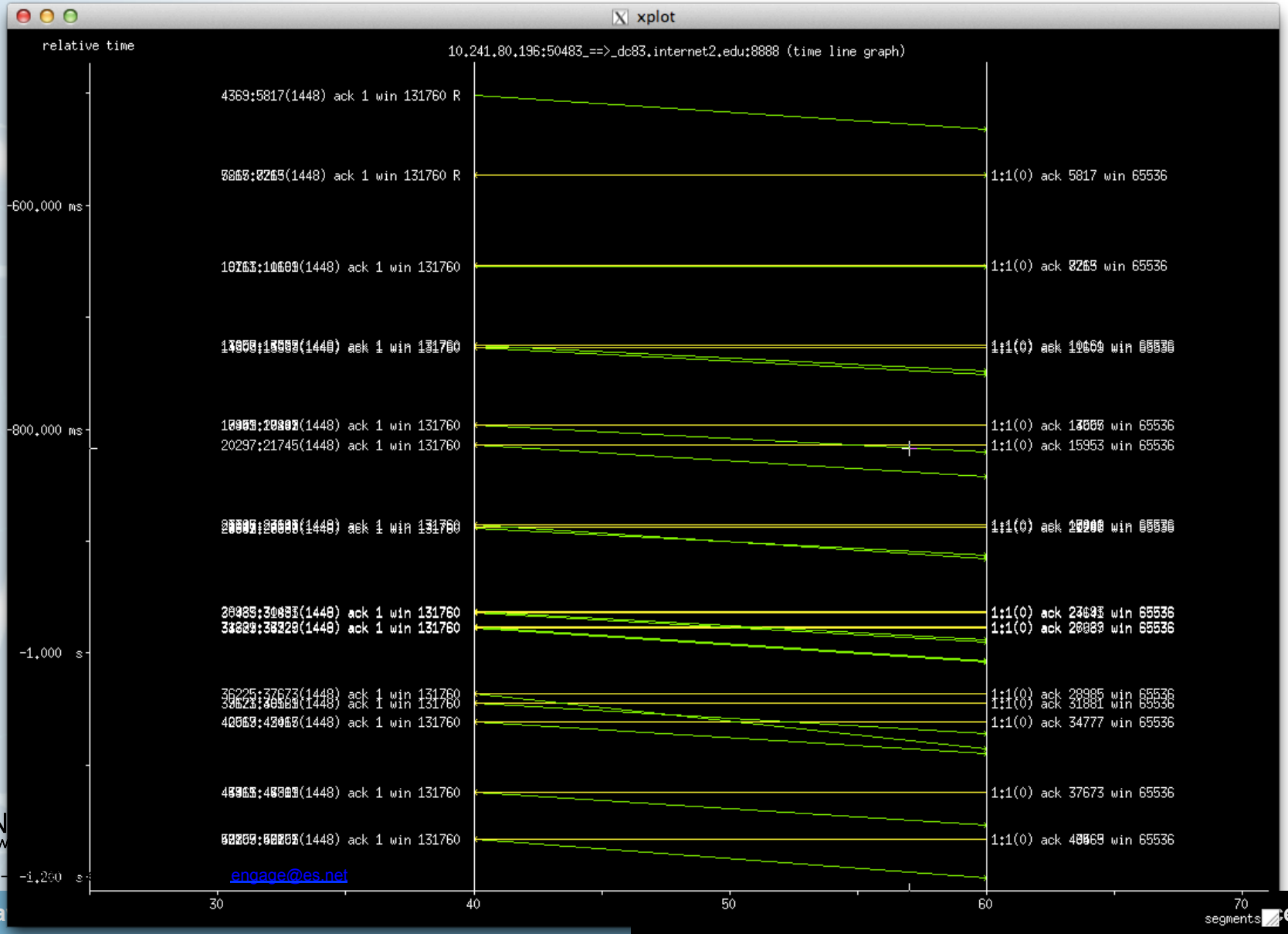



perf

# Example Use Case – XPlot (TLine)



# Example Use Case – XPlot (TLine - zoom)





engage@es.net

# Agenda

- Introduction
- TCPDump – Isolate & Capture Traffic
- TCPTrace – Analyze Captured Traffic
- XPlot – Graphical Display of Flows
- Example of Use
- Real Example (Performance Through a Firewall)

# Experiment Overview

2 Situations to simulate:

- "Outbound" Bypassing Firewall
  - Firewall will ***normally*** not impact traffic leaving the domain. Will pass through device, but should not be inspected
- "Inbound" Through Firewall
  - Statefull firewall process:
    - Inspect packet header
    - If on cleared list, send to output queue for switch/router processing
    - If not on cleared list, inspect and make decision
    - If cleared, send to switch/router processing.
    - If rejected, drop packet and blacklist interactions as needed.
  - Process slows down all traffic, even those that match a white list

# Server (Outbound)

## Run “nuttcp” server:

- `nuttcp -S -p 10200 --nofork`

# Client (Outbound)

Start “tcpdump” on interface (note – isolate traffic to server’s IP Address/Port as needed):

- `sudo tcpdump -i eth1 -w nuttcp1.dmp net  64.57.17.66`
- `tcpdump: listening on eth1, link-type EN10MB (Ethernet), capture size 96 bytes`

Run “nuttcp” client:

- `nuttcp -T 10 -i 1 -p 10200 bwctl.newy.net.internet2.edu`
- `  92.3750 MB /   1.00 sec =  774.3069 Mbps     0 retrans`
- `111.8750 MB /   1.00 sec =  938.2879 Mbps     0 retrans`
- `111.8750 MB /   1.00 sec =  938.3019 Mbps     0 retrans`
- `111.7500 MB /   1.00 sec =  938.1606 Mbps     0 retrans`
- `111.8750 MB /   1.00 sec =  938.3198 Mbps     0 retrans`
- `111.8750 MB /   1.00 sec =  938.2653 Mbps     0 retrans`
- `111.8750 MB /   1.00 sec =  938.1931 Mbps     0 retrans`
- `111.9375 MB /   1.00 sec =  938.4808 Mbps     0 retrans`
- `111.6875 MB /   1.00 sec =  937.6941 Mbps     0 retrans`
- `111.8750 MB /   1.00 sec =  938.3610 Mbps     0 retrans`
- `1107.9867 MB /  10.13 sec =  917.2914 Mbps 13 %TX 11 %RX 0 retrans 8.38 msRTT`

Complete “tcpdump”:

- `974685 packets captured`
- `978481 packets received by filter`
- `3795 packets dropped by kernel*`
- `* Older versions of TCPdump have buffer problems, drops occur at high rates`

# Analysis (Outbound)

## Perform “tcptrace” analyses:

- `tcptrace -l nuttcp1.dmp`

- `1 arg remaining, starting with 'nuttcp1.dmp'`
- `Ostermann's tcptrace -- version 6.6.7 -- Thu Nov  4, 2004`

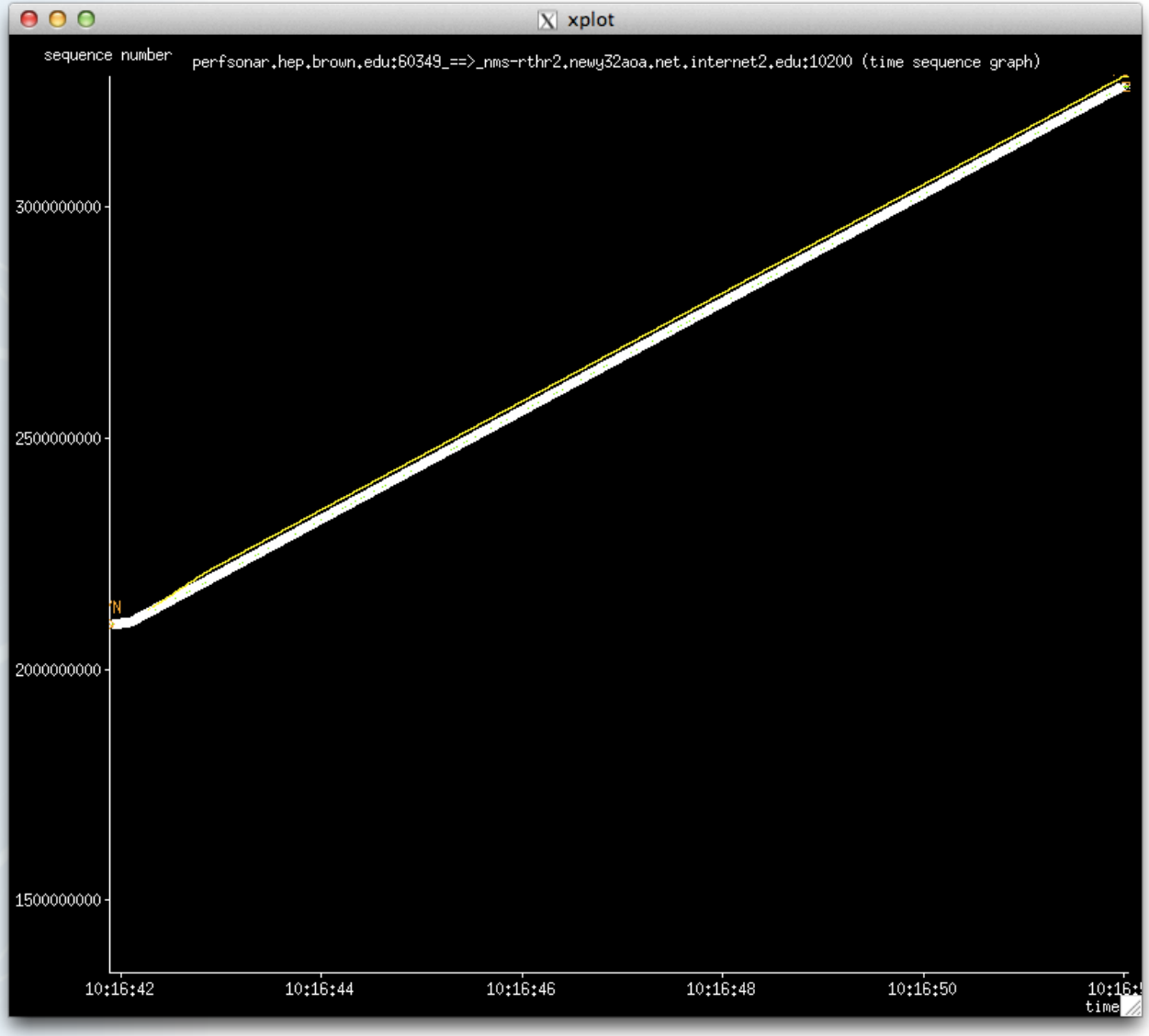
- `974685 packets seen, 974685 TCP packets traced`
- `elapsed wallclock time: 0:00:00.622323, 1566204 pkts/sec analyzed`
- `trace file elapsed time: 0:00:10.215806`
- `TCP connection info:`
- `2 TCP connections traced:`

## Perform “tcptrace” graph generation:

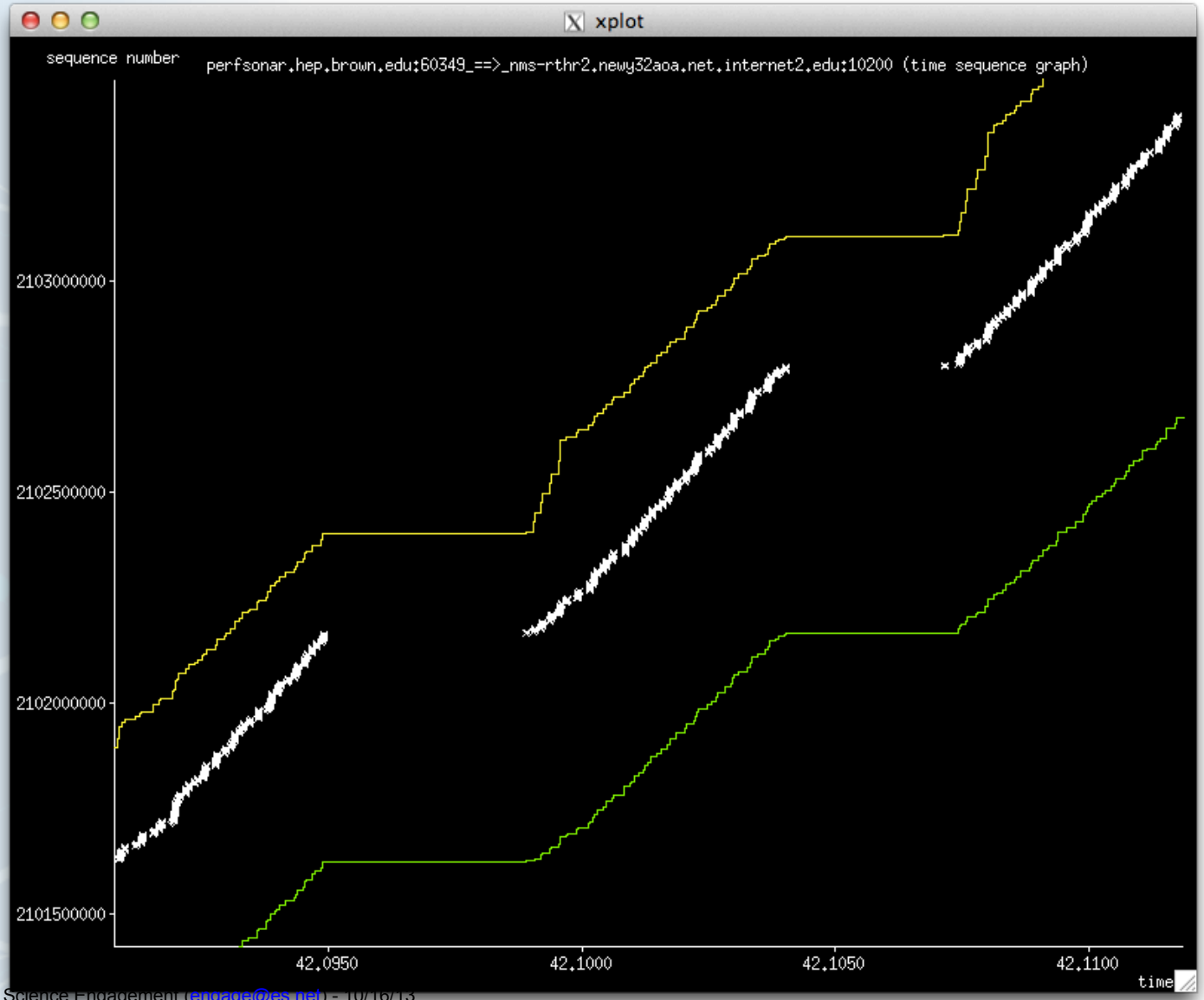
- `tcptrace -G nuttcp1.dmp`
- `1 arg remaining, starting with 'nuttcp1.dmp'`
- `Ostermann's tcptrace -- version 6.6.7 -- Thu Nov  4, 2004`

- `974685 packets seen, 974685 TCP packets traced`
- `elapsed wallclock time: 0:00:33.083618, 29461 pkts/sec analyzed`
- `trace file elapsed time: 0:00:10.215806`
- `TCP connection info:`
- `1: perfsonar.hep.brown.edu:47617 - nms-rthr2.newy32aoa.net.internet2.edu:5000 (a2b)    18>   17< (complete)`
- `2: perfsonar.hep.brown.edu:60349 - nms-rthr2.newy32aoa.net.internet2.edu:10200 (c2d) 845988> 128662< (complete)`

# Plotting (Outbound)



# Plotting Zoomed (Outbound)



# Server (Inbound)

## Run “nuttcp” server:

- `nuttcp -S -p 10200 --nofork`

# Client (Inbound)

Start “tcpdump” on interface (note – isolate traffic to server’s IP Address/Port as needed):

- `sudo tcpdump -i eth1 -w nuttcp2.dmp net  64.57.17.66`
- `tcpdump: listening on eth1, link-type EN10MB (Ethernet), capture size 96 bytes`

Run “nuttcp” client:

- `nuttcp -r -T 10 -i 1 -p 10200 bwctl.newy.net.internet2.edu`
- `    4.5625 MB /   1.00 sec =   38.1995 Mbps    13 retrans`
- `    4.8750 MB /   1.00 sec =   40.8956 Mbps     4 retrans`
- `    4.8750 MB /   1.00 sec =   40.8954 Mbps     6 retrans`
- `    6.4375 MB /   1.00 sec =   54.0024 Mbps     9 retrans`
- `    5.7500 MB /   1.00 sec =   48.2310 Mbps     8 retrans`
- `    5.8750 MB /   1.00 sec =   49.2880 Mbps     5 retrans`
- `    6.3125 MB /   1.00 sec =   52.9006 Mbps     3 retrans`
- `    5.3125 MB /   1.00 sec =   44.5653 Mbps     7 retrans`
- `    4.3125 MB /   1.00 sec =   36.2108 Mbps     7 retrans`
- `    5.1875 MB /   1.00 sec =   43.5186 Mbps     8 retrans`
- `   53.7519 MB /  10.07 sec =   44.7577 Mbps 0 %TX 1 %RX 70 retrans 8.29 msRTT`

Complete “tcpdump”:

- `62681 packets captured`
- `62683 packets received by filter`
- `0 packets dropped by kernel`

perfSONAR powered

# Analysis (Inbound)

## Perform “tcptrace” analyses:

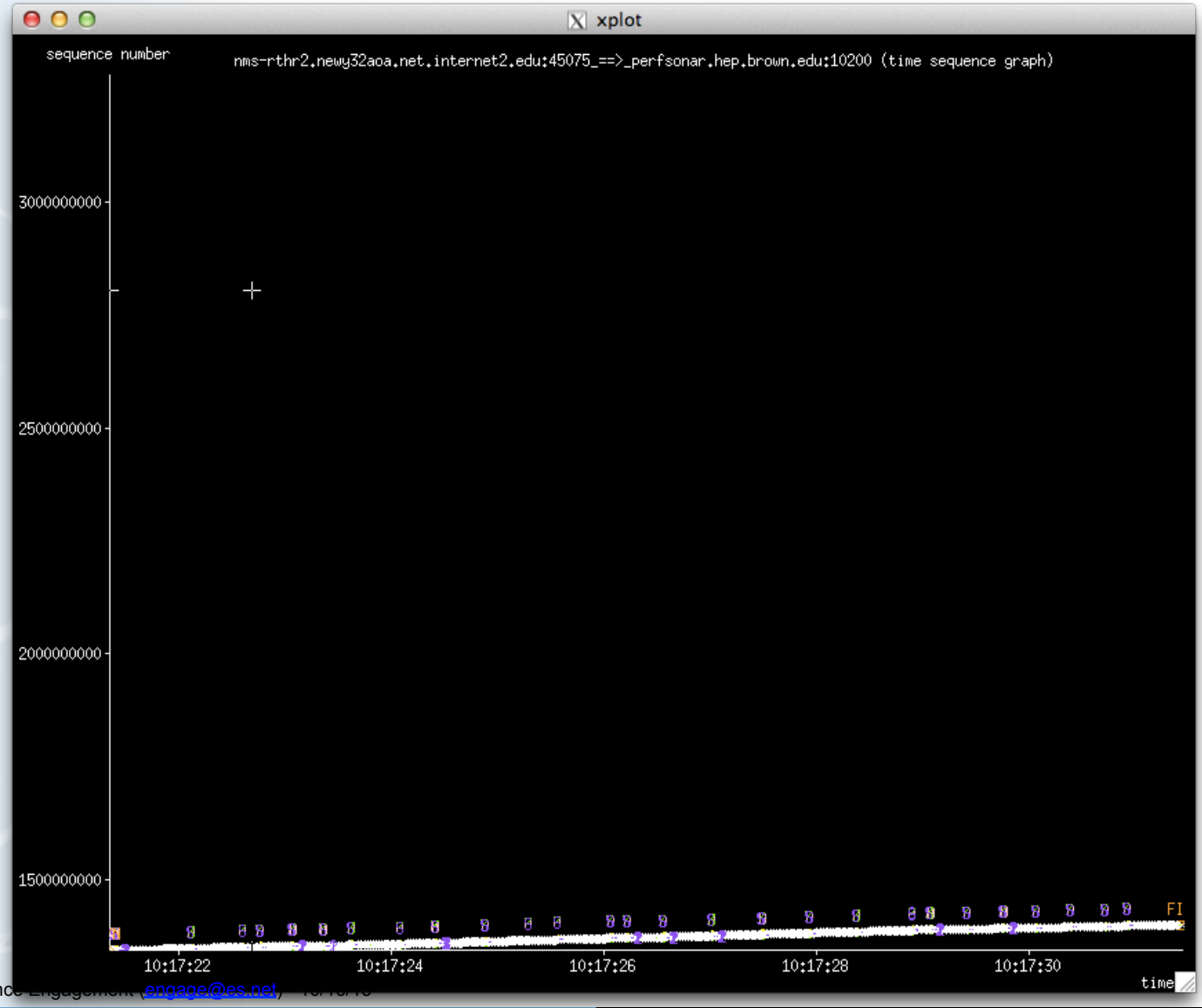
- `tcptrace -l nuttcp2.dmp`
- `1 arg remaining, starting with 'nuttcp2.dmp'`
- `Ostermann's tcptrace -- version 6.6.7 -- Thu Nov  4, 2004`
- `62681 packets seen, 62681 TCP packets traced`
- `elapsed wallclock time: 0:00:00.126221, 496597 pkts/sec analyzed`
- `trace file elapsed time: 0:00:10.188876`
- `TCP connection info:`
- `2 TCP connections traced:`

## Perform “tcptrace” graph generation:

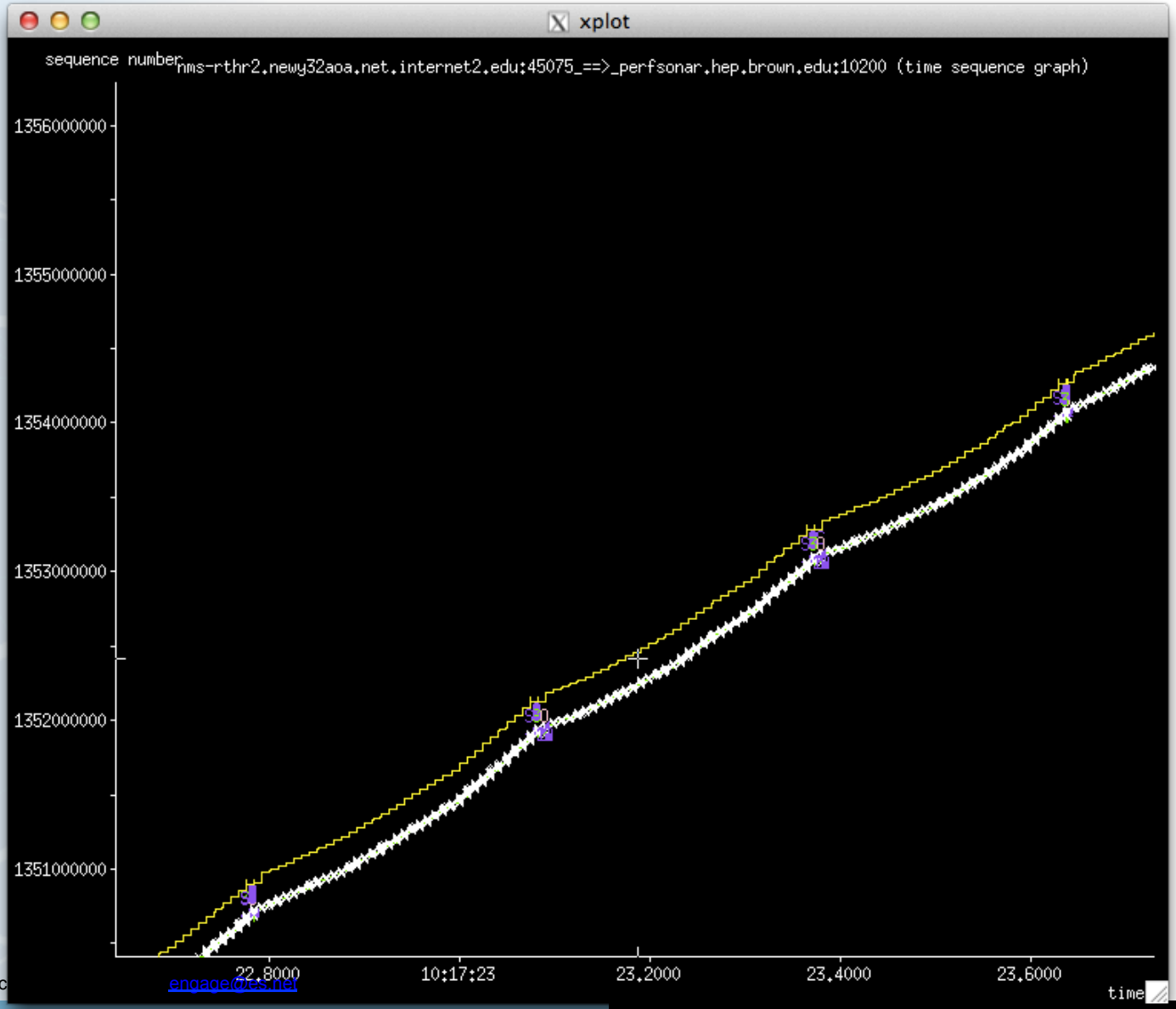
- `tcptrace -G nuttcp2.dmp`
- `1 arg remaining, starting with 'nuttcp2.dmp'`
- `Ostermann's tcptrace -- version 6.6.7 -- Thu Nov  4, 2004`
- `62681 packets seen, 62681 TCP packets traced`
- `elapsed wallclock time: 0:00:02.285531, 27425 pkts/sec analyzed`
- `trace file elapsed time: 0:00:10.188876`
- `TCP connection info:`
- `1: perfsonar.hep.brown.edu:50560 - nms-rthr2.newy32aoa.net.internet2.edu:5000 (a2b)  8>  7< (complete)`
- `2: nms-rthr2.newy32aoa.net.internet2.edu:45075 - perfsonar.hep.brown.edu:10200 (c2d) 41205> 21461< (complete)`

perfSONAR powered

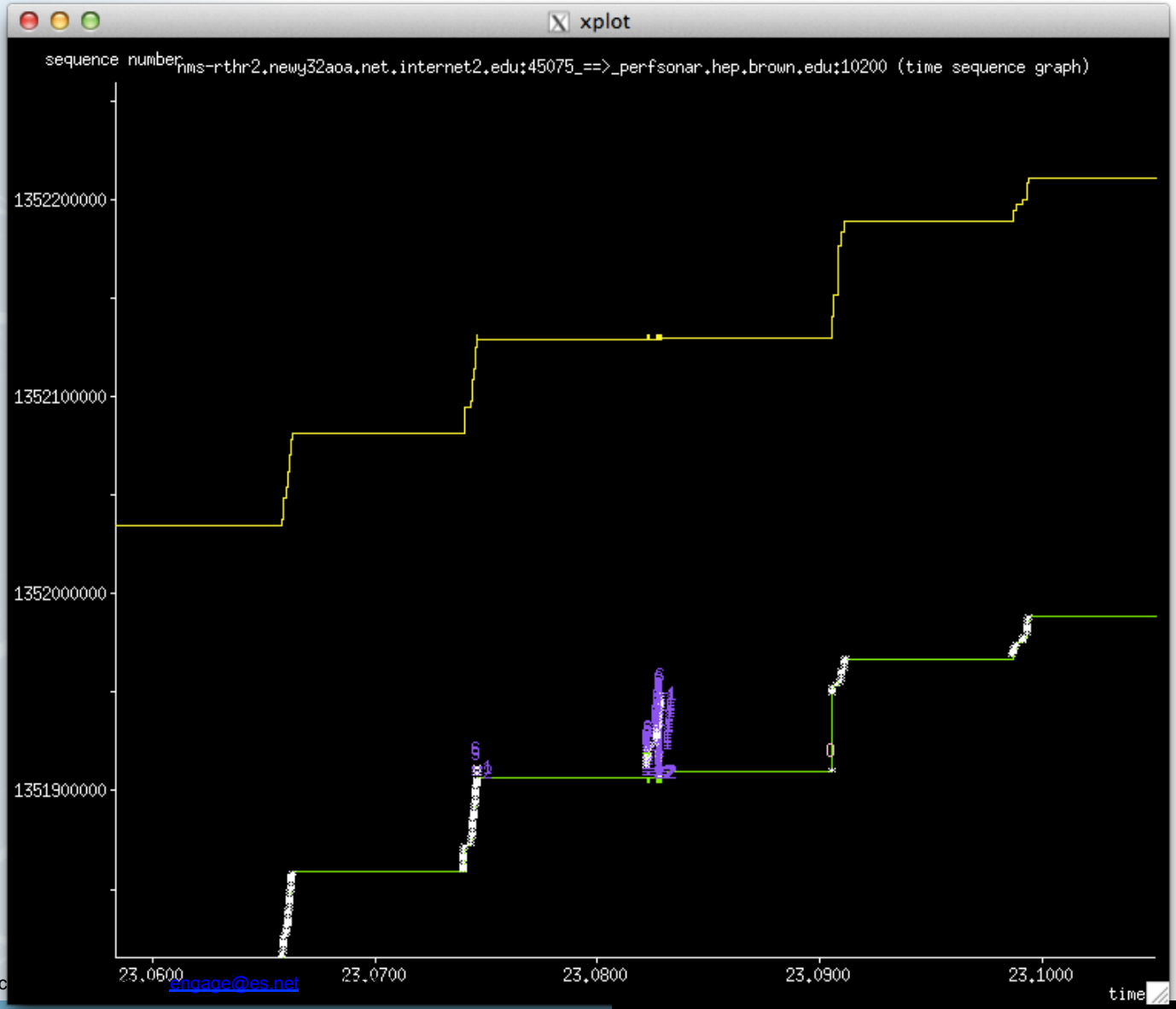
# Plotting (Inbound)



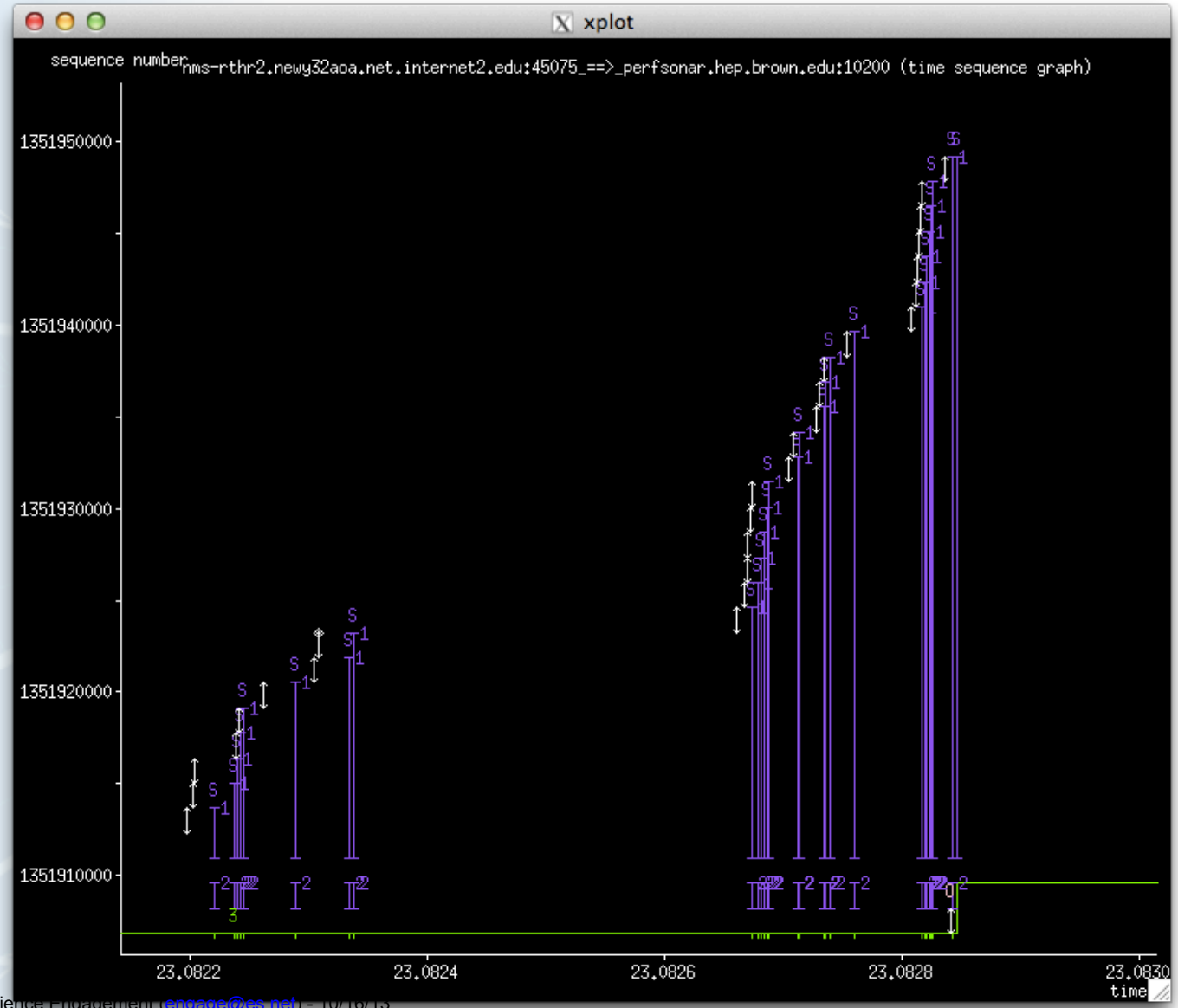
# Plotting Zoomed (Inbound)



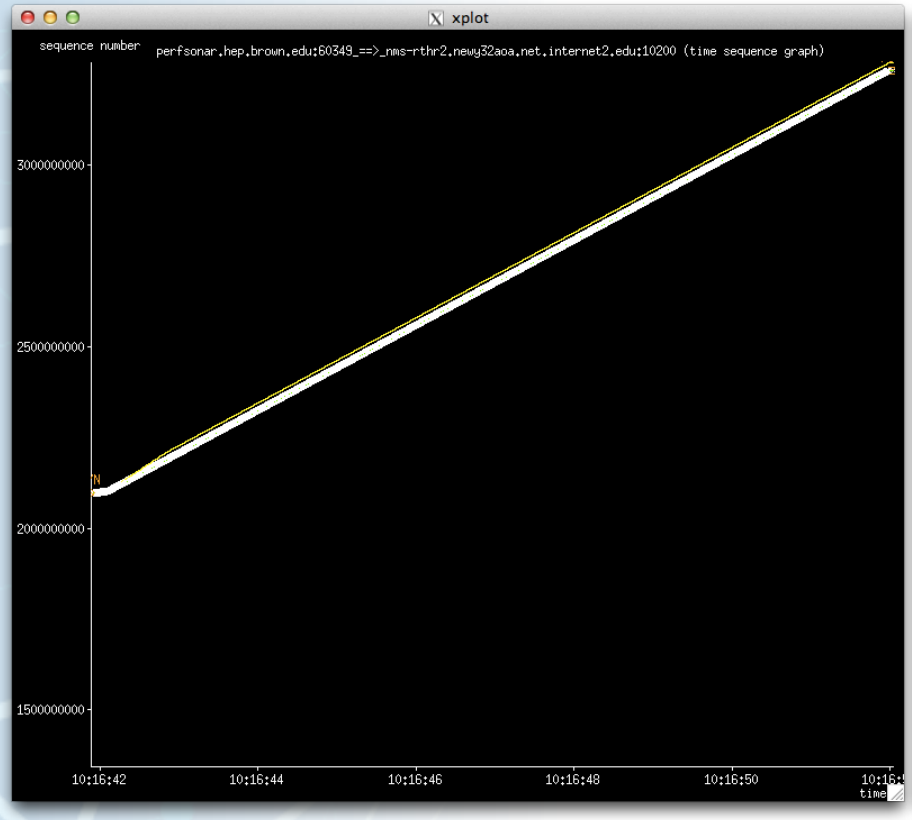
# Plotting Zoomed (Inbound)

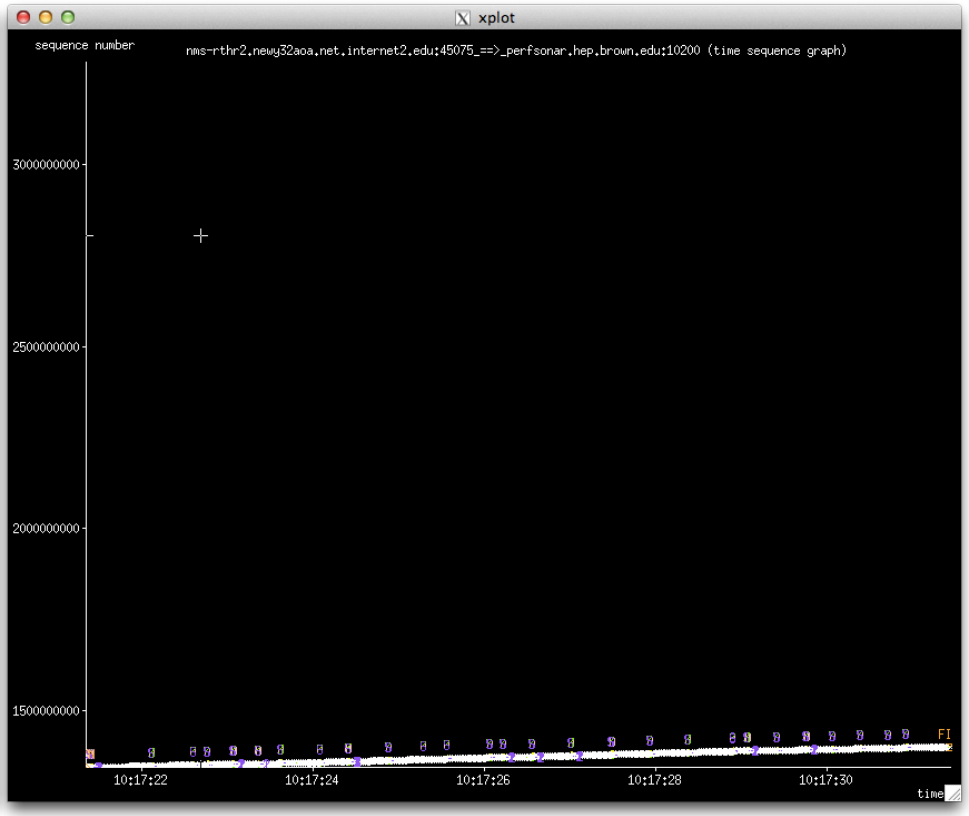


# Plotting Zoomed to OOP/SAK (Inbound)



# Side by Side (Slope = Throughput)



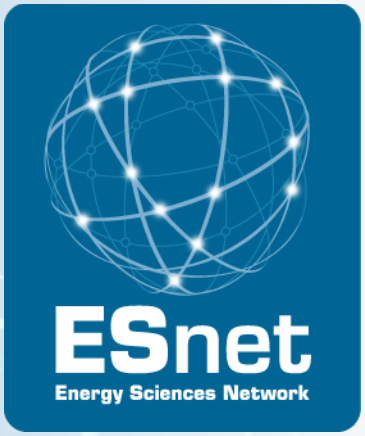



# Conclusion

Lower level tools can help diagnose what the higher level tools are seeing

Low throughput is the result of all network problems when using TCP

- OOP
- Loss
- Retransmissions/Timeouts




# Using TCPDump, TCPTrace, & XPlot to Debug Network Problems

Jason Zurawski - zurawski@es.net

ESnet Science Engagement – engage@es.net

http://fasterdata.es.net



perfSONAR powered