perfSONAR deployment & Performance Troubleshooting

Network Performance and Monitoring workshop

Lætitia Delvaux, PSNC, laetitia.delvaux@man.poznan.pl

perfSONAR is developed by a partnership of

Deployment plan

How to build a useful measurement mesh?

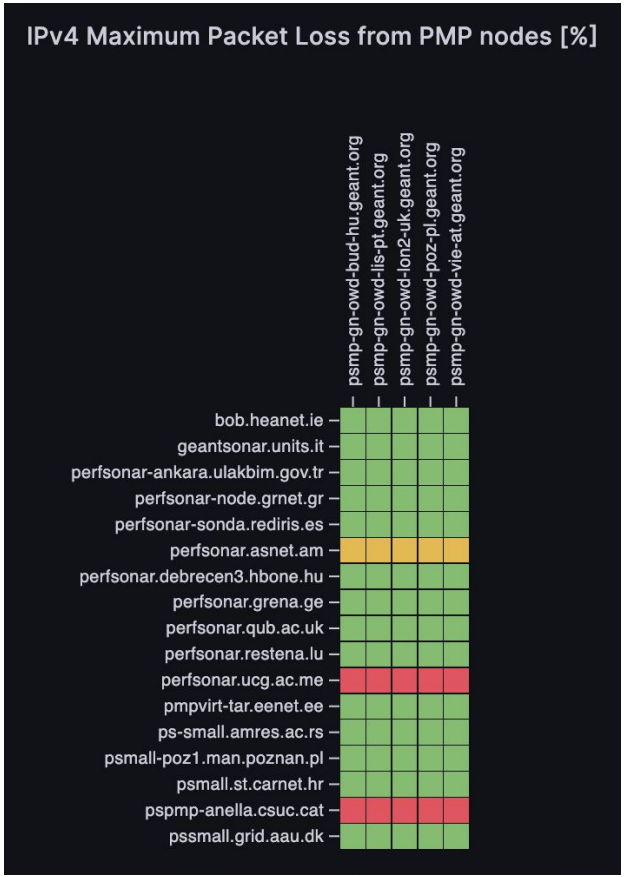
Importance of Regular Testing

- We can’t wait for users to report problems and then fix them (soft failures can go unreported for years!)
- Things just break sometimes
  - Failing optics
  - Somebody messed around in a patch panel and kinked a fiber
  - Hardware goes bad
- Problems that get fixed have a way of coming back
  - System defaults come back after hardware/software upgrades
  - New employees may not know why the previous employee set things up a certain way and back out fixes
- Important to continually collect, archive, and alert on active throughput test results

Dashboard grids

PMP dashboard:
https://pmp-archive.geant.org

© Scott Adams, Inc./Dist. by UFS, Inc.



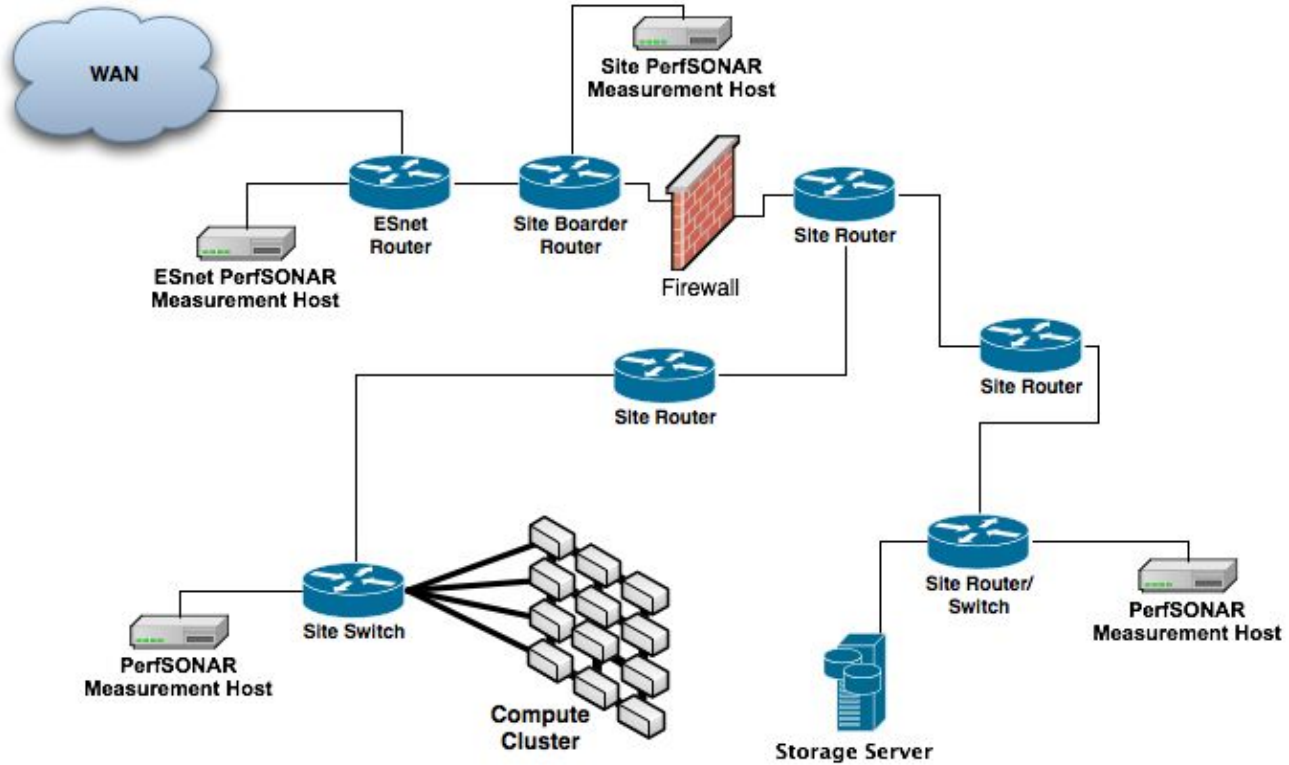
Regular perfSONAR Tests

- We run regular tests to check for three things
  - TCP throughput
  - One way packet loss and delay
  - traceroute
- perfSONAR has mechanisms for managing regular testing between perfSONAR hosts
  - Statistics collection and archiving
  - Graphs
- This infrastructure is deployed now – perfSONAR hosts at facilities can take advantage of it
- At-a-glance health check for data infrastructure

perfSONAR Deployment Locations

- Critical to deploy near key resources such as DTNs
- More perfSONAR hosts allow segments of the path to be tested separately
  - Reduced visibility for devices between perfSONAR hosts
  - Must rely on counters or other means where perfSONAR can't go
- Effective test methodology derived from protocol behavior
  - TCP suffers much more from packet loss as latency increases
  - TCP is more likely to cause loss as latency increases
  - Testing should leverage this in two ways
    - Design tests so that they are likely to fail if there is a problem
    - Mimic the behavior of production traffic as much as possible
  - Note: don't design your tests to succeed
    - The point is not to "be green" even if there are problems
    - The point is to find problems when they come up so that the problems are fixed quickly

Sample Site Deployment



Develop a Test Plan

- What are you going to measure?
  - Achievable bandwidth
    - 2-3 regional destinations
    - 4-8 important collaborators
    - 4-8 times per day to each destination
    - 20 second tests within a region, longer across oceans and continents
  - Loss/Availability/Latency
    - OWAMP: ~10-20 collaborators over diverse paths
  - Interface Utilisation & Errors (via SNMP or streaming telemetry)
  - Write it in a pSConfig file (will be for another workshop…)
- What are you going to do with the results?
  - Grafana Alerts
  - Reports to user community
  - Grids

Performance Troubleshooting

What to do when performance is not good

Where to Start?

- perfSONAR was designed to debug end-to-end problems.
  - Easily deployed where it needs to be
  - Selection of tools that can be invoked (remotely) on demand
  - Ability to set up regular tests, and view a historical record
- Before starting any investigation, create a plan (perhaps share this with your helpdesk/operations team too …)

Where to Start?

- As we get into this, we will be using a variation of '**divide and conquer**', but we need to be careful:
  - Segment by segment analysis is not helpful – this is not a 'many parts make a whole' problem, TCP doesn't work that way
  - We want to divide our path more intelligently, and build it/take it apart slowly. Testing every single segment from ingress to egress is a false way to diagnose an end-to-end problem

Developing a Plan

- Suggested Flow Chart:
  - Problem Report
  - Triage
  - Resource Identification
  - 1st Pass Fact Finding
  - Performance Testing – Step 1 (Regular Testing Setup)
  - Performance Testing – Step 2 (OWAMP)
  - Performance Testing – Step 3 (Throughput)
  - Performance Testing – Step 4 (Optional – NDT or other tools)
  - Evaluation of Results
  - Hypothesis & Environmental Changes

Describing the Steps

- Problem Report
  - Typically this comes in the form of a ticket, or from a user directly
  - Try to get as much info as you can – and remember this may goes beyond it being a ‘network’ issue (e.g. remember that hosts and applications matter too)
    - Problem seen from where to where?
    - Observed when?
    - Host(s) used?
    - Applications used?
    - Is it repeatable?
    - Prior problems for this set of circumstances?

Describing the Steps

- Triage
  - An offshoot of the previous step – weigh the items from the report and look for the low-hanging fruit. In particular:
    - Make sure the host(s) are tuned for the job they are trying to do, and that the application is working correctly.

Describing the Steps

- Resource Identification:
- **perfSONAR at each 'end' is a must**. This can be a toolkit or a testpoint, it could be a bundle (e.g. the 'tools' bundle)
- If you don't have it, spend the 30 minutes to get a host running (placement at the border, or where the user's resources are to start). Get it on the other end too.
- **Location of perfSONAR resources 'in the middle'** comes next:
  - Do a quick traceroute to see the networks involved.
  - Use the **perfSONAR LS dashboard** (https://stats.perfsonar.net) to locate servers on those networks. We don't need full coverage at this point, but it's good to find things.
  - Try to get LAN maps for the 'ends' if you can as well. Knowing exactly what sits between the user and the border will help you later.

Describing the Steps

- 1st Pass Fact Finding
  - With a sheet of paper (or electronic paper), start making notes:
    - The traceroute and reverse traceroute between the resources. What does these look like? Are they the same, or different?
    - If you have the LAN maps, do you see any abnormalities? Mark locations where there are bandwidth step-downs (e.g. multiple 1Gs into a 10G, etc.).
    - If you are using CACTI/Cricket/MRTG/etc., look at the utilization, drops, and errors from the interfaces that make up the network path. Is congestion a factor? Are there patterns of errors/drops consistent with use, or do they look constant?
    - If you have access to regional/national data for the above two items – try to do the same as well. You do not need to have all the answers at this stage
  - As in the experimental method, make a hypothesis

Describing the Steps

- Performance Testing – Step 1 (Regular Testing Setup)
  - If you are using a perfSONAR Toolkit – establish some regular tests:
    - To the remote end having problems
    - To the regional that remote end connects to
    - To the backbone location the regional may use (e.g. Internet2 POP in some city)
    - To your regional
    - To your network demarcation (if you are testing from behind a firewall)
  - OWAMP tests are *very* important, we want to know packet loss
  - Throughput tests are also important, we want to see throughput. Note that for 'close' things (your network demarcation and your regional), throughput may be 'good', even in the case of packet loss.
    - Confirm that the host's CPU is not the bottleneck. Consider using 'mpstat' or the output of iperf3 or nuttcp
  - It takes a couple of days to get good data, but don't let that stop you from continuing with the steps

Describing the Steps

- Performance Testing – Step 2 (OWAMP)
  - **`owping`** is the tool we are going to use:
    - `$ owping $HOST`
  - Twiddle some of the options:
    - -c: number of packets sent, e.g. –c 1000 = 1000 packets in the session
    - -i: interval between packets, e.g. –i .01 = send 100 packets per second
    - -s: size of the packets sent, e.g. –s100 = 100 byte (pre header) size of packets
  - Note that twiddling the options does increase the length of time of the test, and possibly the bandwidth used. For example:
    - `$ owping -c 1000 -I .01 -s 100` will send 1000 packets, of 100 bytes in size, where 100 packets are sent every second. The entire test will take 10 seconds total.
  - What we want to do is tackle situations where the devices in the path may be congested on a micro scale. If we run a long enough test, we could see packet loss or out of order behavior.
  - Test against the opposite end, and to things in the middle. Keep good notes.

Describing the Steps

- Performance Testing – Step 3 (Throughput/iperf3)
  - As in the previous case, we are going to test to the other end, and to things in the middle with:
    ```
    $ pscheduler task --tool iperf3 throughput --source $SENDING_HOST
    --dest $RECEIVE_HOST --duration PT30S --interval PT2S
    ```
  - Suggestions on use:
    - Use iperf3 as the tool (should be default)
    - Tests should be between 20-30 seconds, and use an interval of 1-2. This will let you see TCP behavior as it rises and falls (and gives you sufficient time to ramp up speed and window size)
    - Avoid parallel streams to start – see what a delicate single stream will do.
    - Run both directions (exchange --source and --dest) for each target. We want to see patterns, and make sure they are consistent.
  - Make sure we are recording things – if possible draw a map between the networks and notate the links with these speeds

Describing the Steps

- Performance Testing – Step 3 (Throughput/iperf3) – cont.
  - **Our end goal is to pull out the 'longest, cleanest' path we can.** What does that mean?
    - TCP dynamics tell us that throughput will be "high", even with packet loss, when the latency is "small". This is why testing exclusively to something on your border, or at your regional network is not useful from a throughput standpoint
    - If the original report is correct, and performance is 'bad' to the other end, we want to slowly back away away from that end until we can find where things look 'good'.
      - Maybe this is to the regional network, maybe it's to the backbone.
      - Maybe it's to neither, and the problem is really on our network – this is why we need OWAMP to help with packet loss, we can't rely on throughput alone.

Describing the Steps

- Performance Testing – Step 3 (Throughput/iperf3) – cont.
  - What about UDP? It can detect a class of problems (e.g. failing equipment) that TCP can't
    - UDP testing is a double edged sword:
      - By default, perfSONAR nodes limit the rate of UDP to 50Mbps (fear of packet canon exposure). To use this for higher bandwidth testing, you need to modify your limits files
      - UDP testing requires a lot of CPU and memory to work well. If your host doesn't have the base hardware to support this, the results will appear limited, even if the network is fine. See this write up from ESnet for more information: https://fasterdata.es.net/network-tuning/udp-tuning/

Describing the Steps

- If you are comfortable allowing UDP to be used, and know how to play with the /etc/pscheduler/limits file to change the bandwidth limit, it can be useful for determining some things that TCP testing, and OWAMP, wouldn't find:
    - The limit at which packets are dropped without a back-off mechanism (e.g. this helps determine buffer limitations, or points of congestion)
    - If adjusting size, burst rate, or inserting a rate limit have any impact

- The takeaway: TCP tools will work off the bat, UDP is harder to work with unless you know the people involved on the path, and can negotiate for resources (removing filters).

Describing the Steps

- Evaluation of Results
  - It's important not to get ahead of yourself – do all the tests first to get a full picture based on the different metrics
    - Don't perform any false diagnostics until everything is known
  - Go back to the regular testing after a day or two as well
    - Do the results that are graphed look similar to what the live testing revealed?
    - If not, maybe another round of live testing is in order
  - Update/formalize the map of what you see

Describing the Steps

- Hypothesis & Environmental Changes
  - The data may start to show patterns:
    - Perhaps good throughput is found up to one point, and then it degrades after that
    - Perhaps throughput to *everything* is poor, except for a resource that is 1-5 msec away
    - Perhaps packet loss is seen in a very specific location
  - Using the map, mark off areas that are ‘clean’ versus those that warrant more attention
    - A section that shows higher throughput and zero packet loss doesn’t need additional scrutiny (be careful of TCP dynamics in this case)
    - A section that shows a drop in throughput (either or both directions) and hints of packet loss should be looked at

Describing the Steps

- Hypothesis & Environmental Changes – cont.
  - Possible environmental things to examine:
    - Using CACTI/MRTG/Cricket (etc.), what does the utilization profile look like, and do you see a pattern of errors/discards? Remember, if you don't see it at Layer 3, you might want to look at layer 2 or 1 …
    - What does the tuning of the buffers for the devices look like? Can the buffers be tuned at all? See the ESnet resources for assistance: http://fasterdata.es.net/network-tuning/router-switch-buffer-size-issues/
    - Is there high 'fan in' or 'fan out'? This occurs when many smaller connections reach a larger or equal sized connection.
  - After changes, it's important to retest to verify things.
    - Try to test after each small change – don't fall into the trap of fixing many things, and then retesting (fixing a network should be like fixing a car, it is not a random walk of possibilities).

HIDDEN SLIDE

- After the theory, here is a practical example.
- This is a very involved problem that Jason/Eli debugged.
- If you don't know the story – ask them to tell it to you. If you don't think you can explain it, cut out these slides.

Putting Theory Into Practice

- Let's look at a real life example using some of the steps above.
- Brevity is being employed, but please ask questions if you are not making the connections.

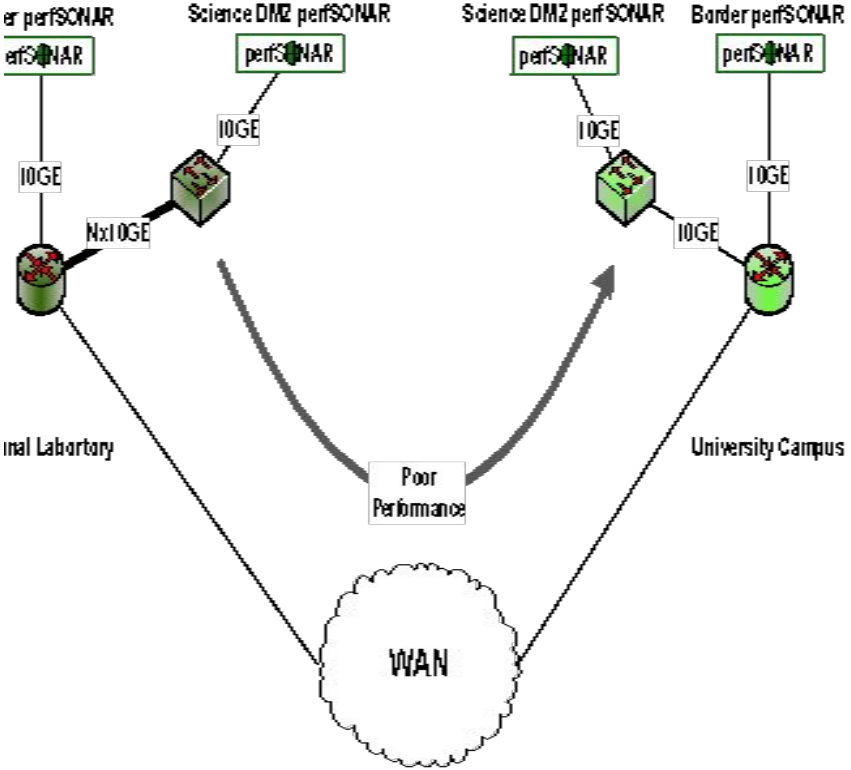
WAN Test Methodology – Problem Isolation

- We said it before, but it bears repeating: segment-to-segment testing is not helpful
  - TCP dynamics will be different, and in this case all the pieces do not equal the whole
    - E.g. high throughput on a 1ms path with high packet loss vs. the same segment in a longer 20ms path
  - Problem links can test clean over short distances
  - An exception to this is hops that go thru a firewall

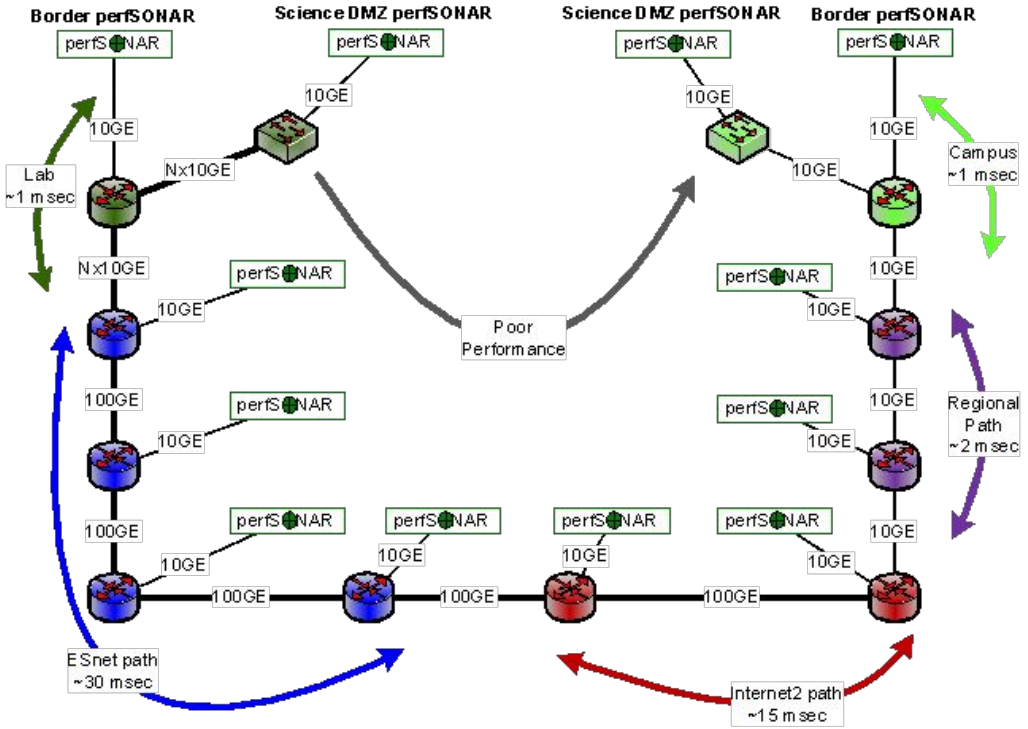
WAN Test Methodology – Problem Isolation

- Run long-distance tests
  - ***Run the longest clean test you can***, then look for the ***shortest dirty test*** that includes the path of the clean test
- In order for this to work, the testers need to be already deployed when you start troubleshooting
  - ESnet has at least one perfSONAR host at each hub location.
    - Many (most?) R&E providers in the world have deployed at least 1
  - If your provider does not have perfSONAR deployed ask them why, and then ask when they will have it done

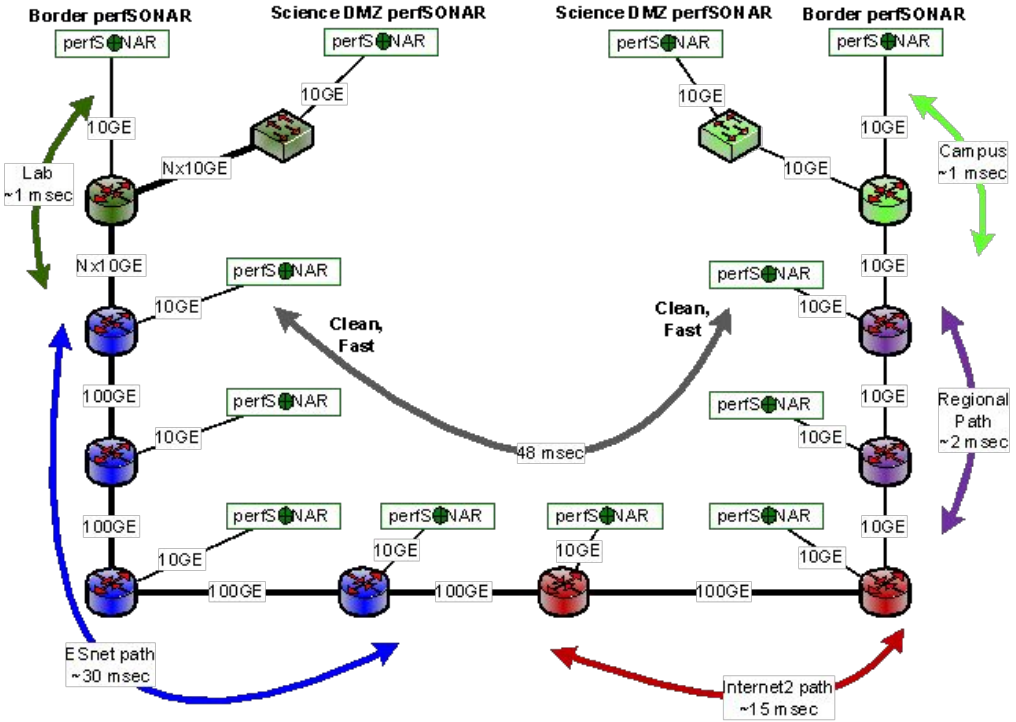
Network Performance Troubleshooting Example



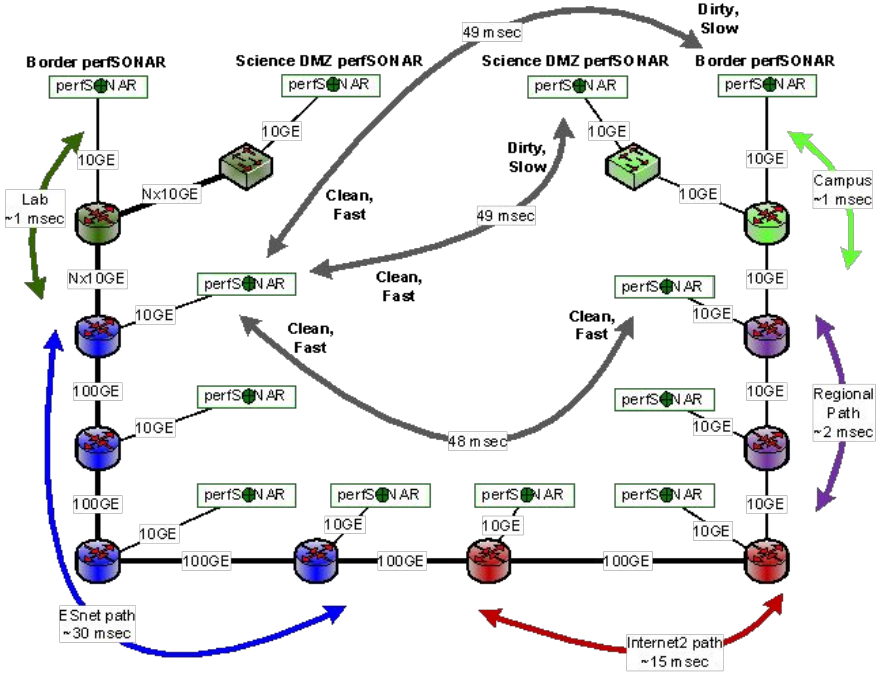
Wide Area Testing – Full Context



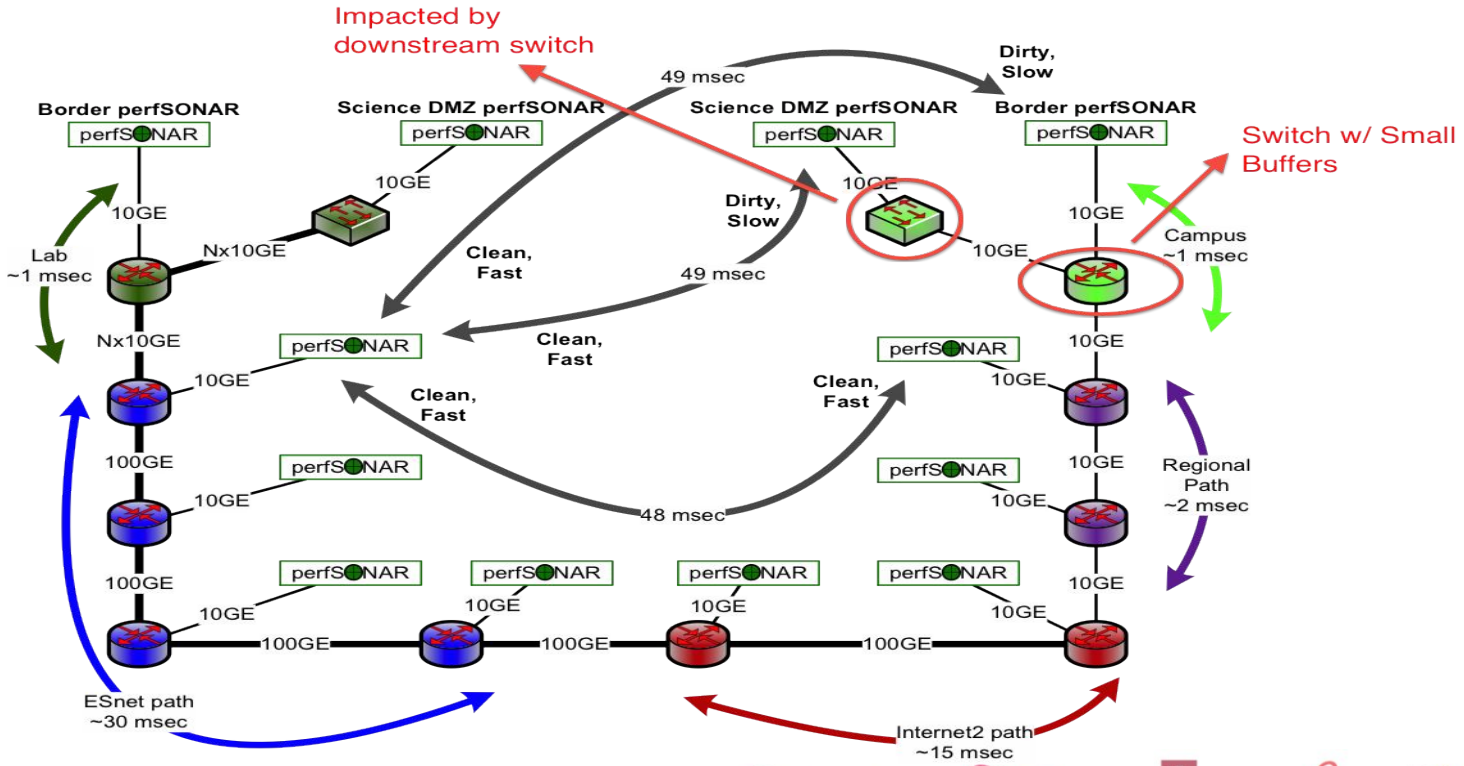
Wide Area Testing – Long Clean Test



Wide Area Testing – Poorly Performing Tests Illustrate Likely Problem Areas



Likely Problem Area



Lessons From This Example

- This testing can be done quickly if perfSONAR is already deployed
- Huge productivity
  - Reasonable hypothesis developed quickly
  - Probable administrative domain identified
  - Testing time can be short – an hour or so at most
- Without perfSONAR cases like this are very challenging
- Time to resolution measured in months
- In order to be useful for data-intensive science, the network must be fixable quickly, because it ***will*** break

HIDDEN SLIDE

- The next section is really living documentation on how to use TCP dump/plot/trace tools.
- Please run the tools yourself to get comfortable with this stuff. If you are not comfortable, don't attempt.

Advanced Topics

- 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

Advanced Topics

- 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

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

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)
  - -s 96 (always good to include, otherwise you may drop packets)

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

- A note about high-performance packet capture
  - For performance debugging, you usually only need the headers
  - We use the snapshot length to reduce the portion of the packet we write to disk
  - -s 96 (or even –s 84) can be helpful (remember, perfSONAR hosts don’t usually have DTN disk subsystems!)
  - Make sure tcpdump says “0 packets dropped by kernel” when you are done
  - If for some reason your host isn’t using PF_RING (this is a linux-ism) then get a more up to date libpcap (look for this if it’s just impossible to write packet headers to disk without loss)

TCPDump

- Invocation:
  - `sudo tcpdump -s 96 -i eth0 192.168.0.1`
    - or
  - `sudo tcpdump -s 96 -i eth0 host 192.168.0.1`
    - Capture traffic from specific host (incoming and outgoing) on target interface.
  - `sudo tcpdump -s 96 -i eth0 net 192.168.0`
    - Capture traffic from /24 subnet (incoming and outgoing) on target interface.

TCPDump

- Invocation (cont.):
  - `sudo tcpdump -s 96 -i eth0 port 5001`
    - Capture traffic from specific port (incoming and outgoing) on target interface.
  - `sudo tcpdump -s 96 -i eth0 portrange 100-200`
    - Capture traffic from specific ports (incoming and outgoing) on target interface.

TCPDump

- Invocation (cont.):
  - `sudo tcpdump -s 96 -i eth0 src 192.168.0.1`
    - Capture traffic where host is the src, on specific interface
  - `sudo tcpdump -s 96 -i eth0 dst net 192.168.0`
    - Capture traffic where /24 subnet is the dst, on specific interface

TCPDump

- Invocation (cont.):
  - `sudo tcpdump -s 96 -i eth0 src or dst port 5001`
    - Capture traffic where port 5001 is the destination, on specific interface
  - `sudo tcpdump -s 96 -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

TCPDump

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

TCPDump

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.)

TCPDump

- 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: http://www.tcptrace.org/manual/index.html
  - `$ 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)

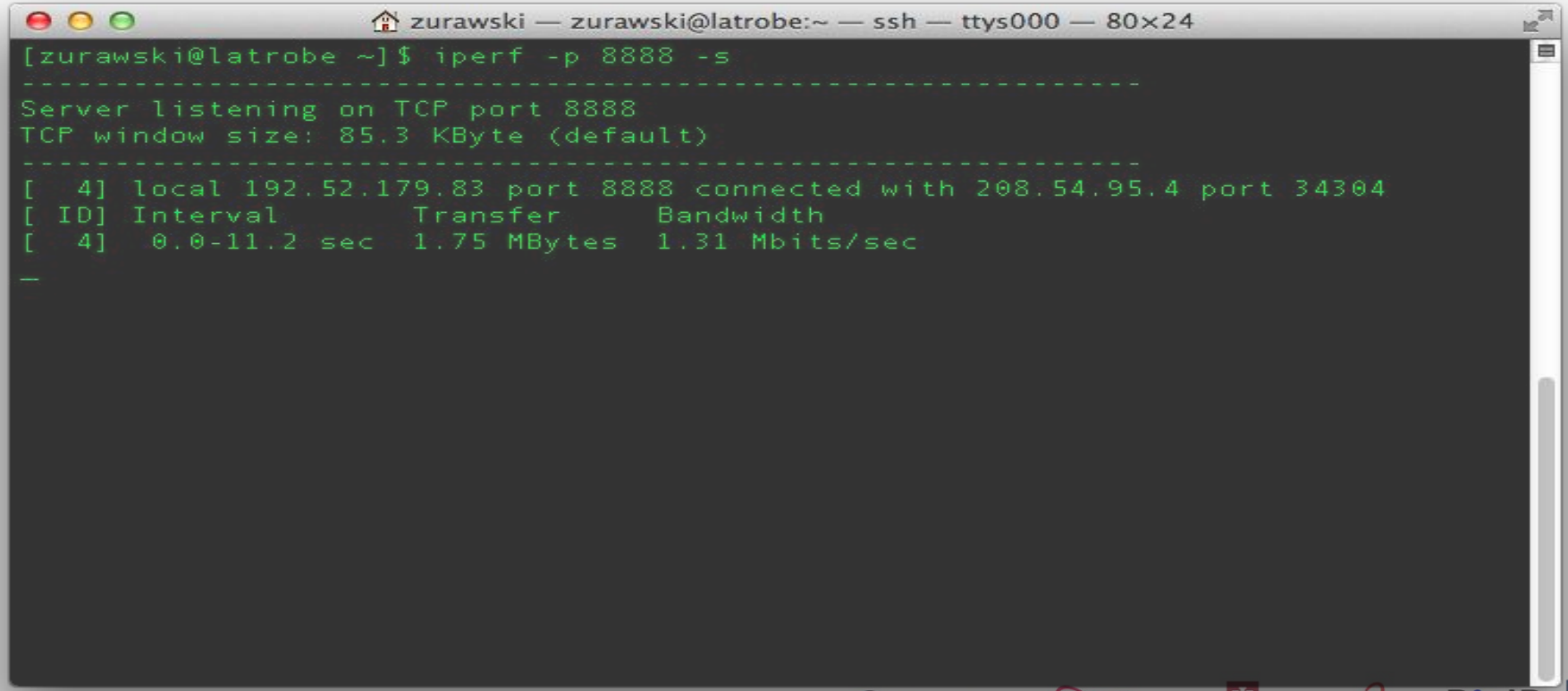
XPlot

- 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’

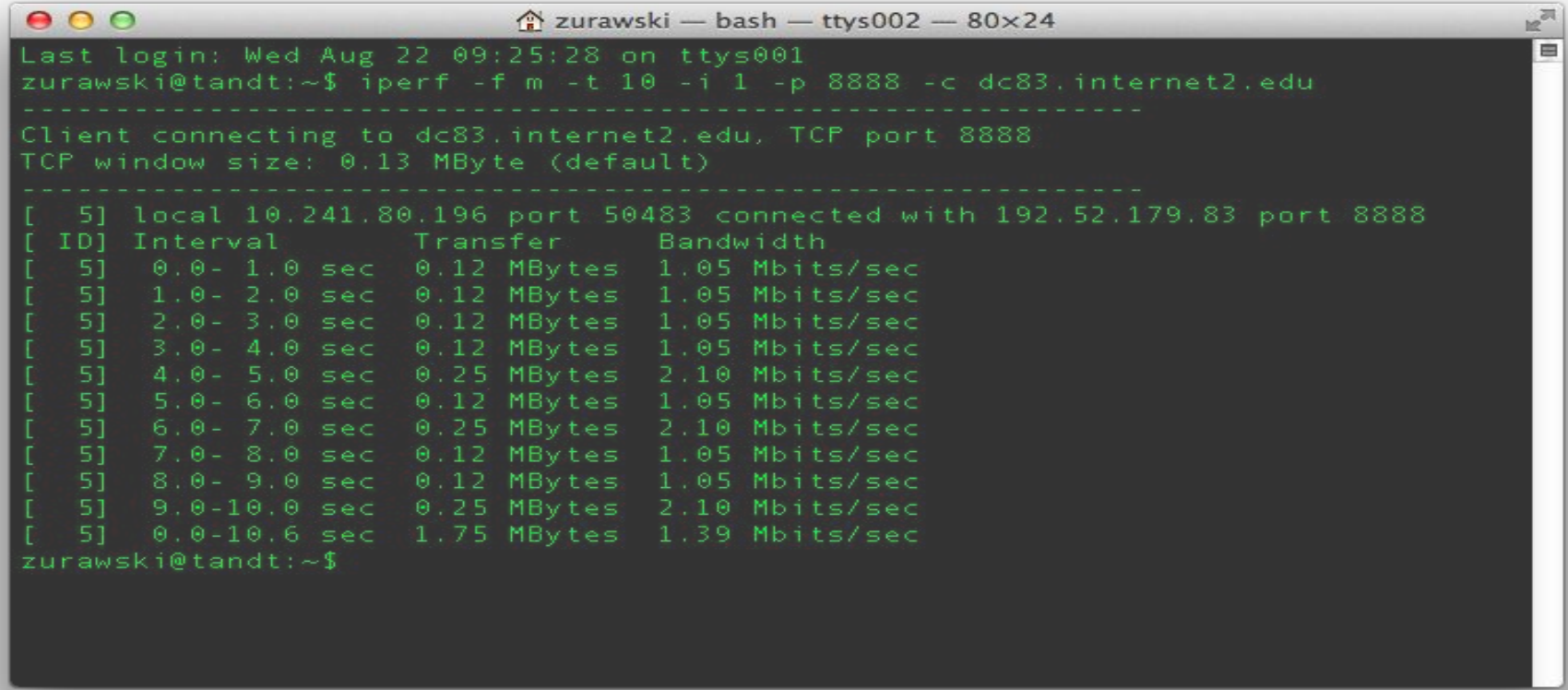
Example Use Case

- 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)

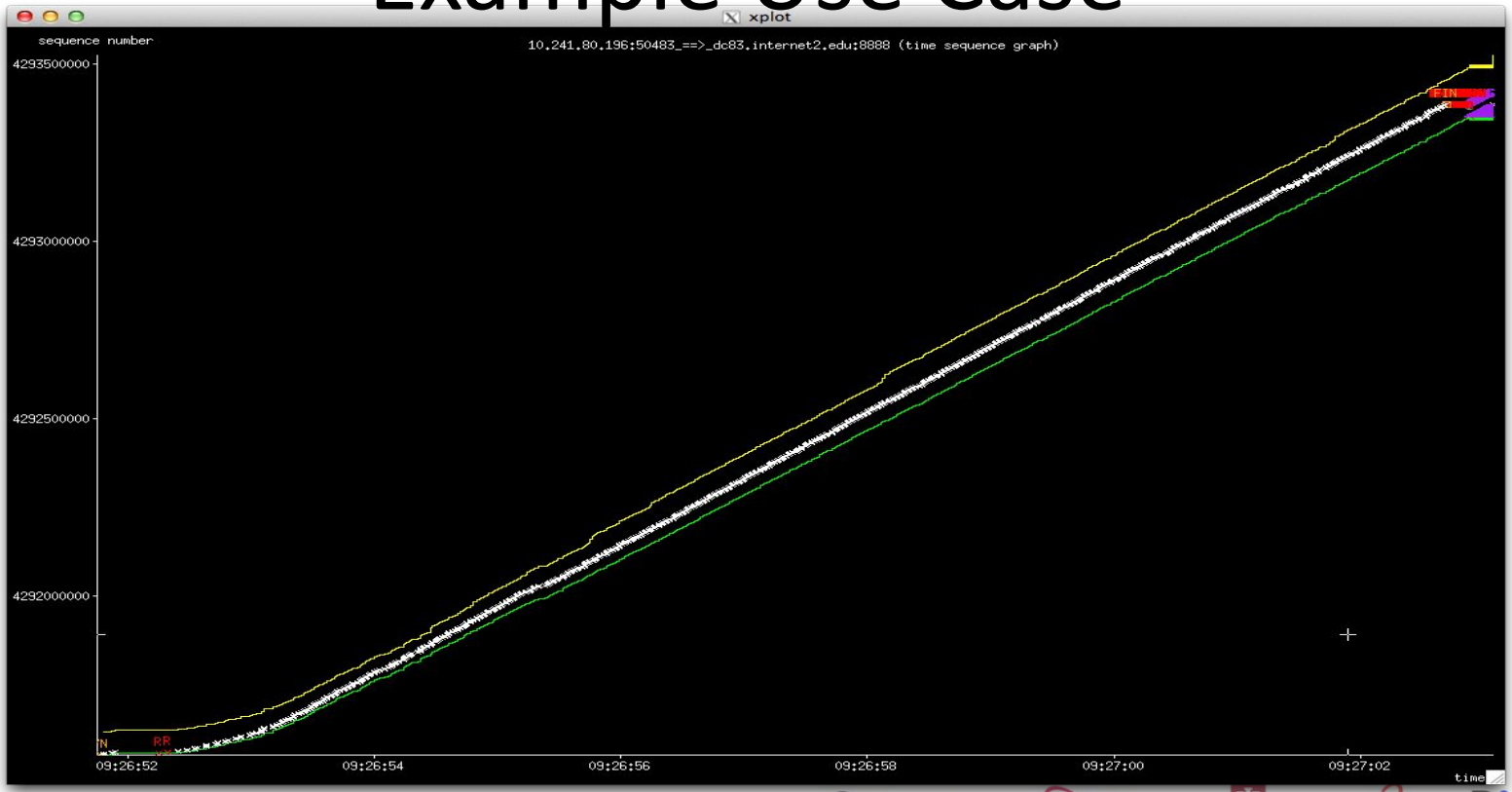
Example Use Case



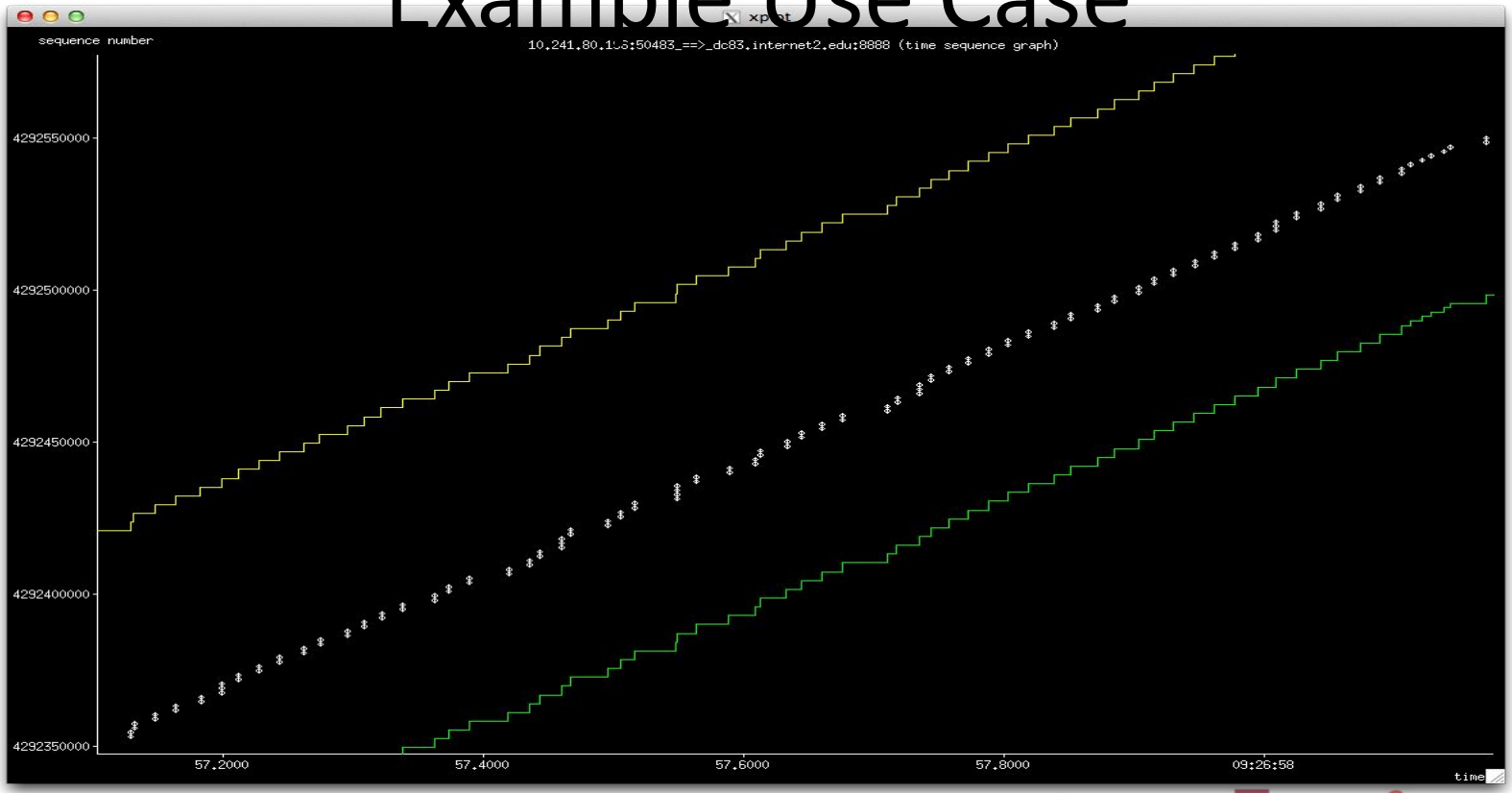
Example Use Case



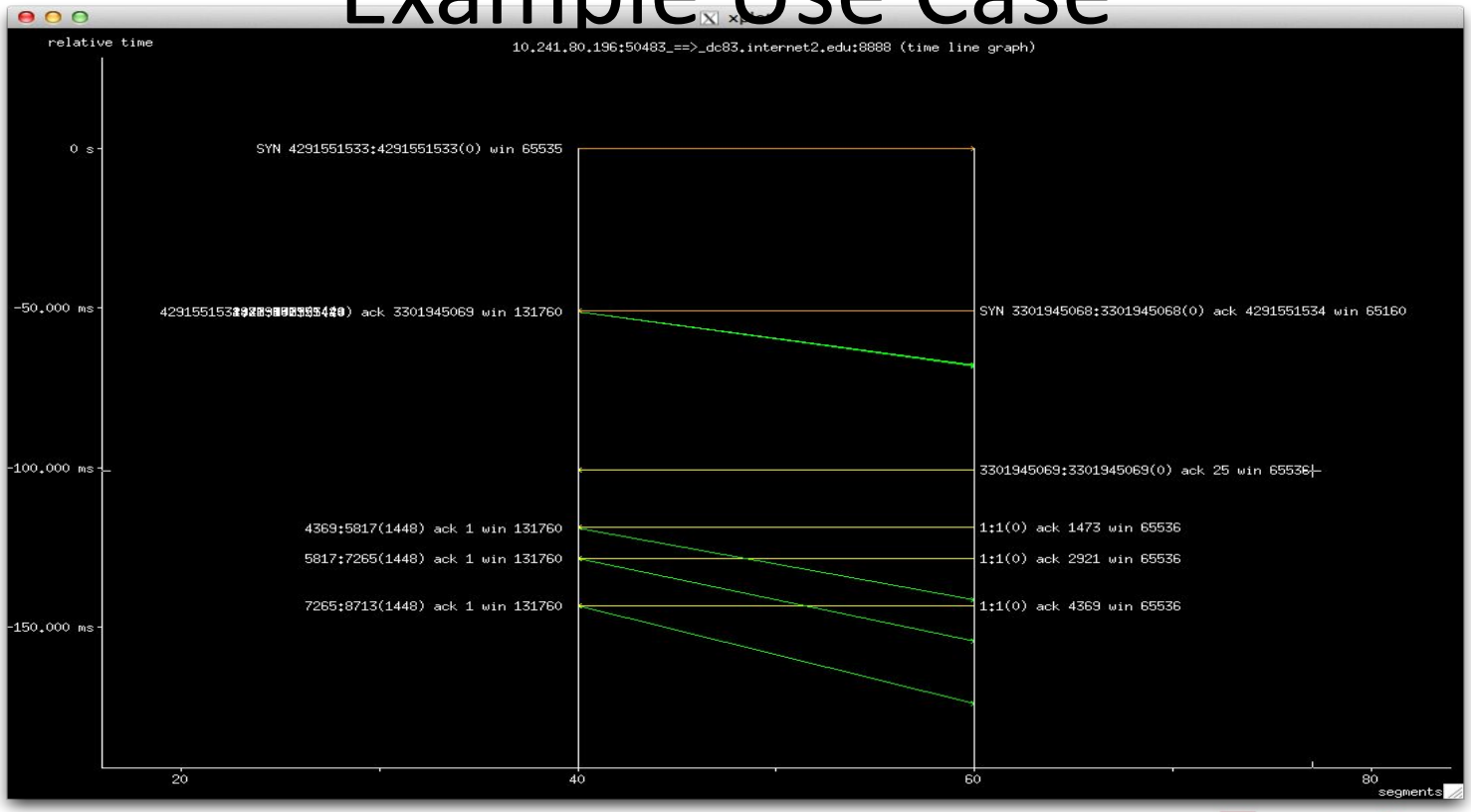
Example Use Case



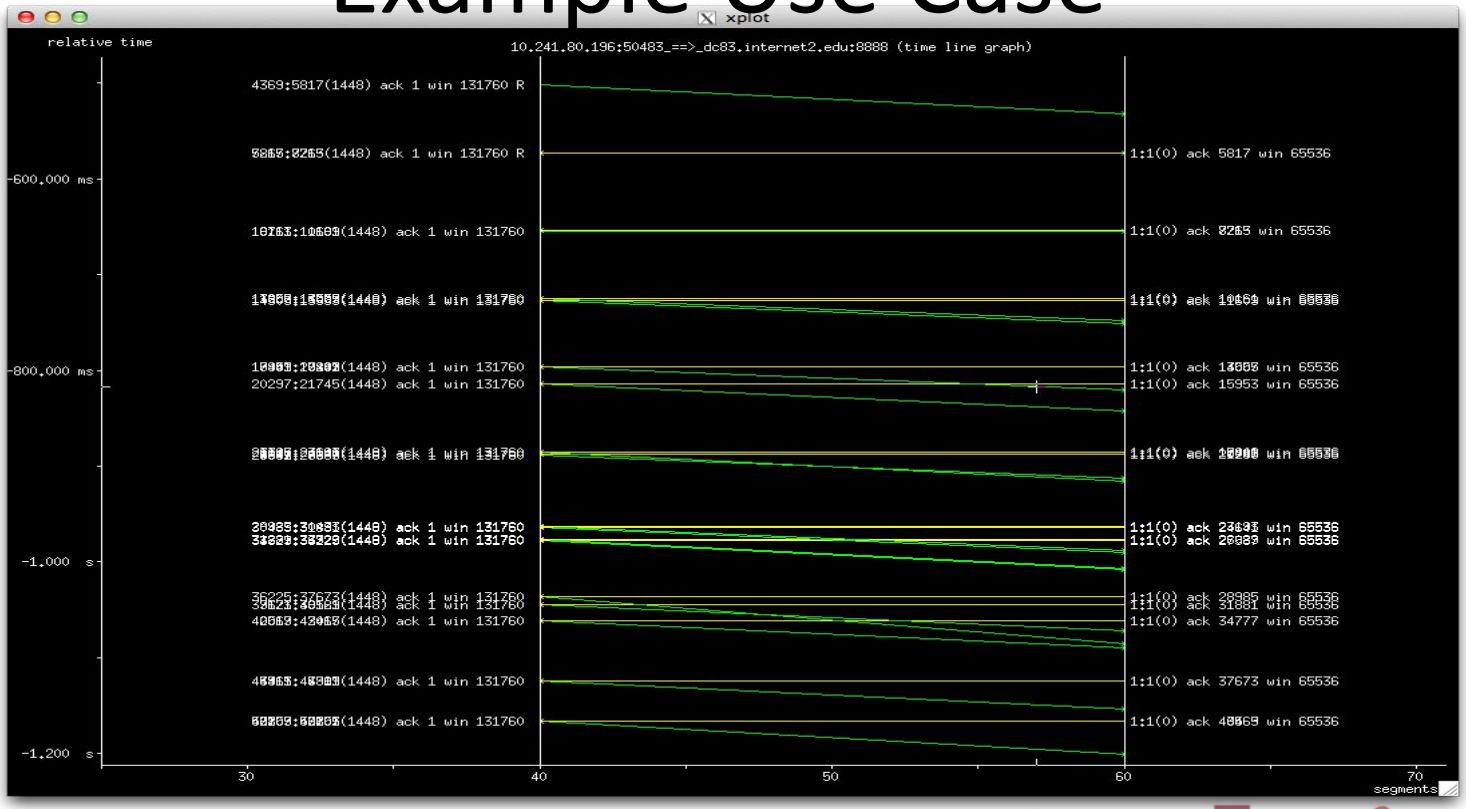
Example Use Case



Example Use Case



Example Use Case



Experimentation

- 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 Side (Outbound)

- Run “nuttcp” server:
  - `$ nuttcp -S -p 10200 --nofork`

Client Side (Outbound)

- Start “tcpdump” on interface (note – isolate traffic to server’s IP Address/Port as needed):
  - `$ sudo tcpdump -s 96 -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`

Client Side (Outbound)

- 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:`

Analysis (Outbound)

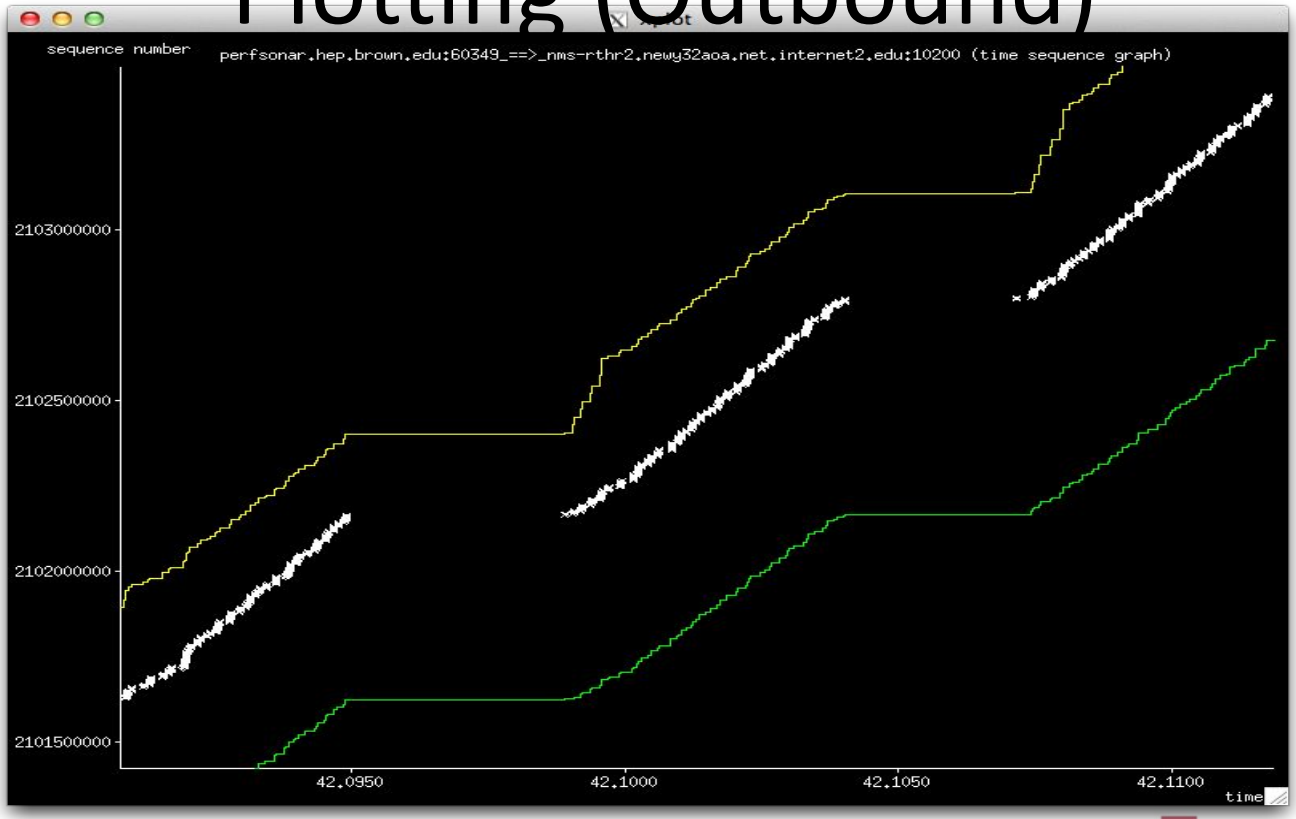
Perform “tcptrace” graph generation:

- `$ 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 (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`
  - `...`
  - `     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`

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:`

Analysis (Inbound)

Perform “tcptrace” graph generation:

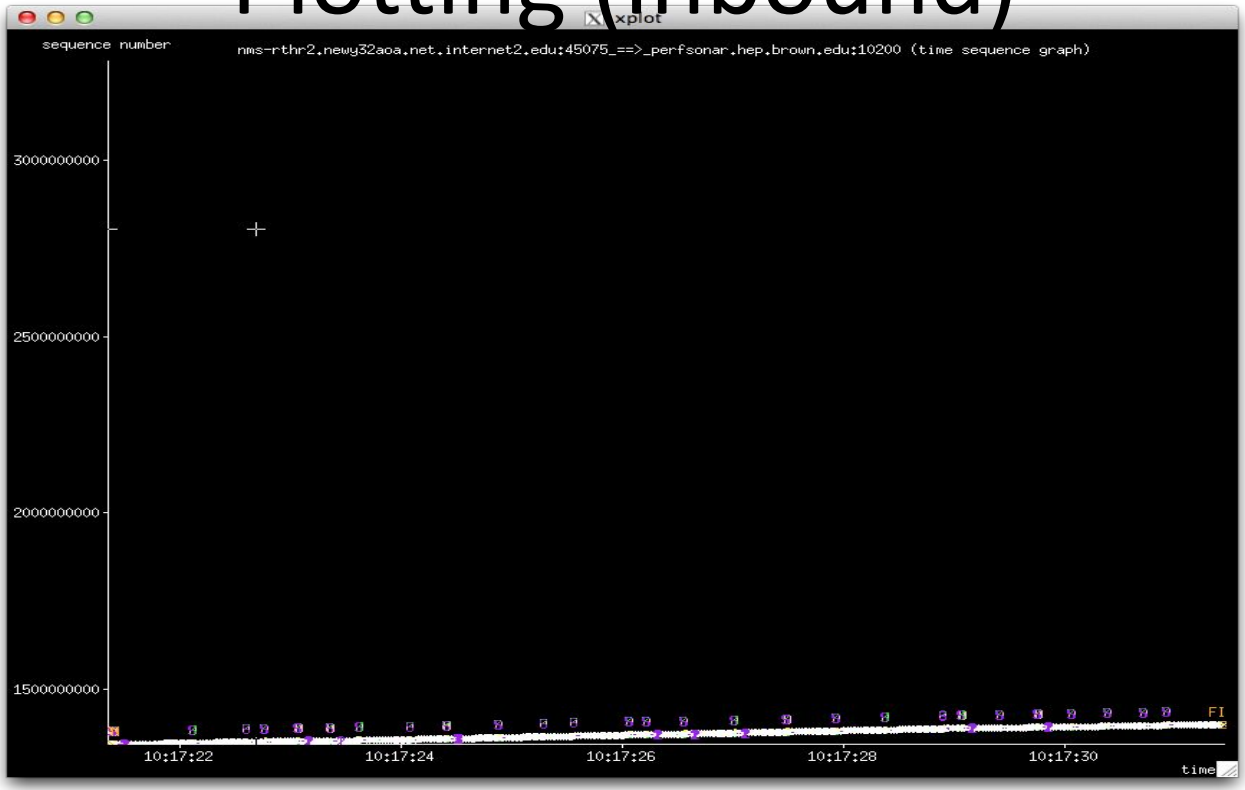
- `tcptrace -G nuttcp2.dmp`
- `1 arg remaining, starting with 'nuttcp2.dmp'`
- `Ostermann's tcptrace -- version 6.6.7 -- Thu Nov  4, 2004`

- `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)`

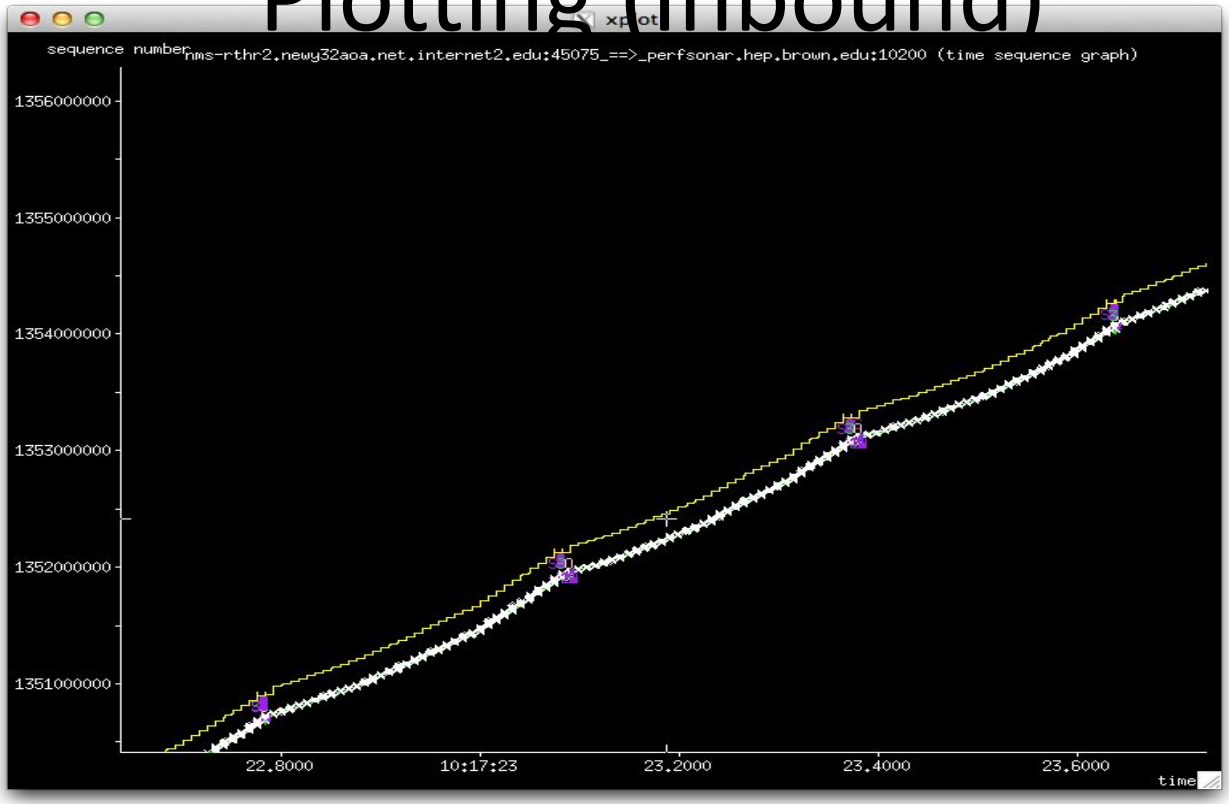
Launch “xplot”:

- `xplot FILE.xpl &`

Plotting (Inbound)

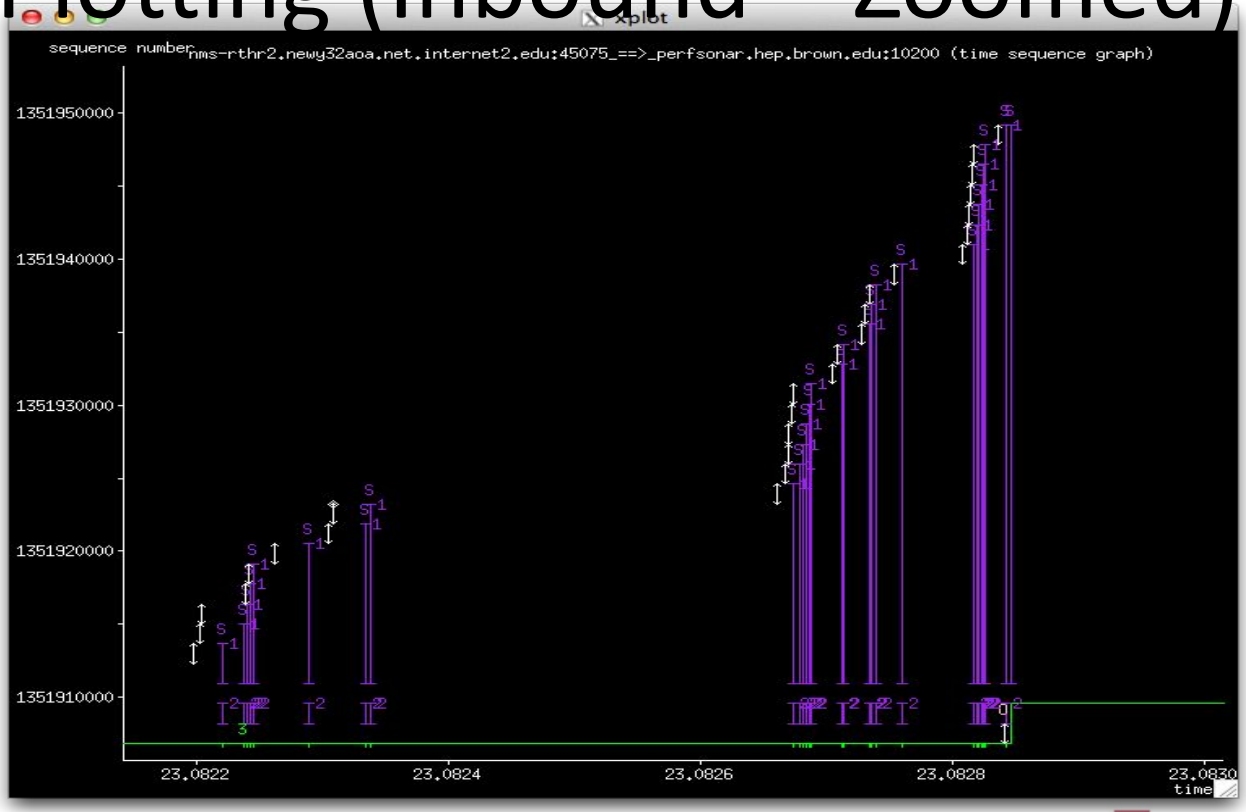


Plotting (Inbound)

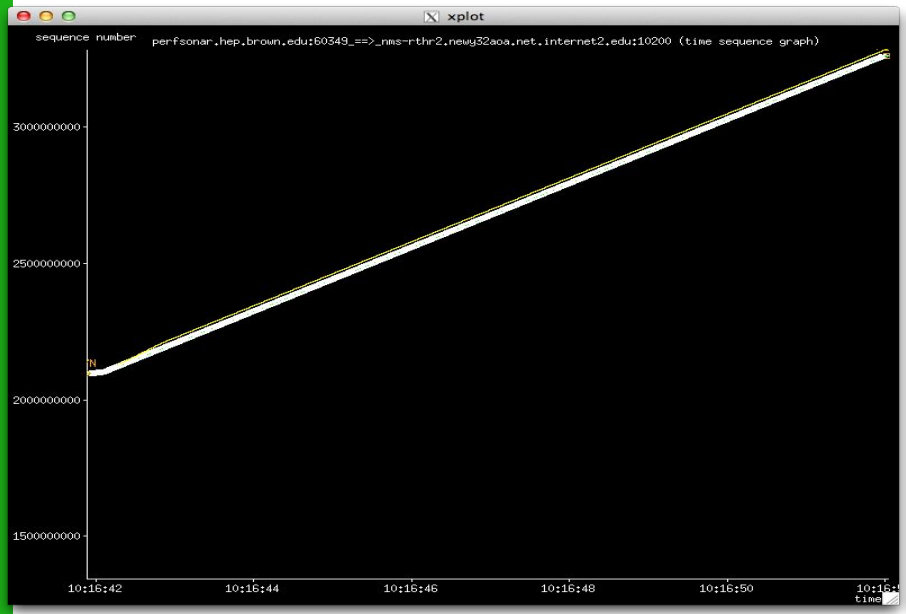


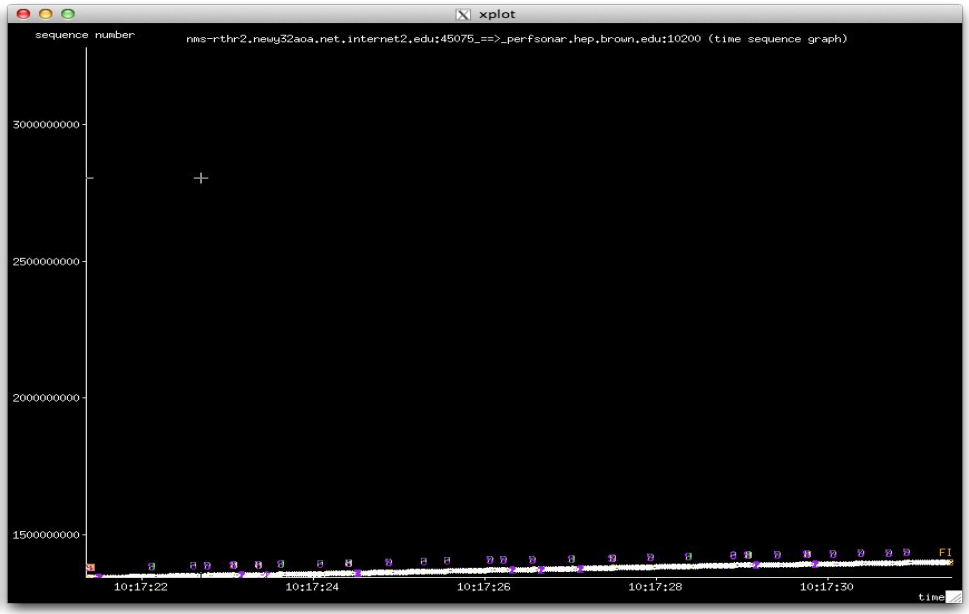
Plotting (Inbound)

Plotting (Inbound – Zoomed)



Slope = Throughput





Conclusions

- 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
  - Out of Order Packets (OOP)
  - Loss
  - Retransmissions/Timeouts

perfSONAR deployment
&
Network Debugging Strategies

Network Performance and Monitoring workshop

Lætitia Delvaux, PSNC, laetitia.delvaux@man.poznan.pl

perfSONAR is developed by a partnership of