Optimizing Data Transfer Nodes using Packet Pacing: A Journey of Discovery

Brian Tierney, ESnet
Nathan Hanford, Dipak Ghosal, UC Davis
INDIS Workshop, 2015
November 16, 2015

ESnet

Observation

- When doing a DTN to DTN transfer, we often come across test results where TCP is dropping packets, but the cause it not obvious
  - No errors
  - No congestion
- After more investigation, the cause is often:
  - Speed mismatch
  - Under-buffered devices
  - Under-powered firewalls
- Examples of this on the next slides

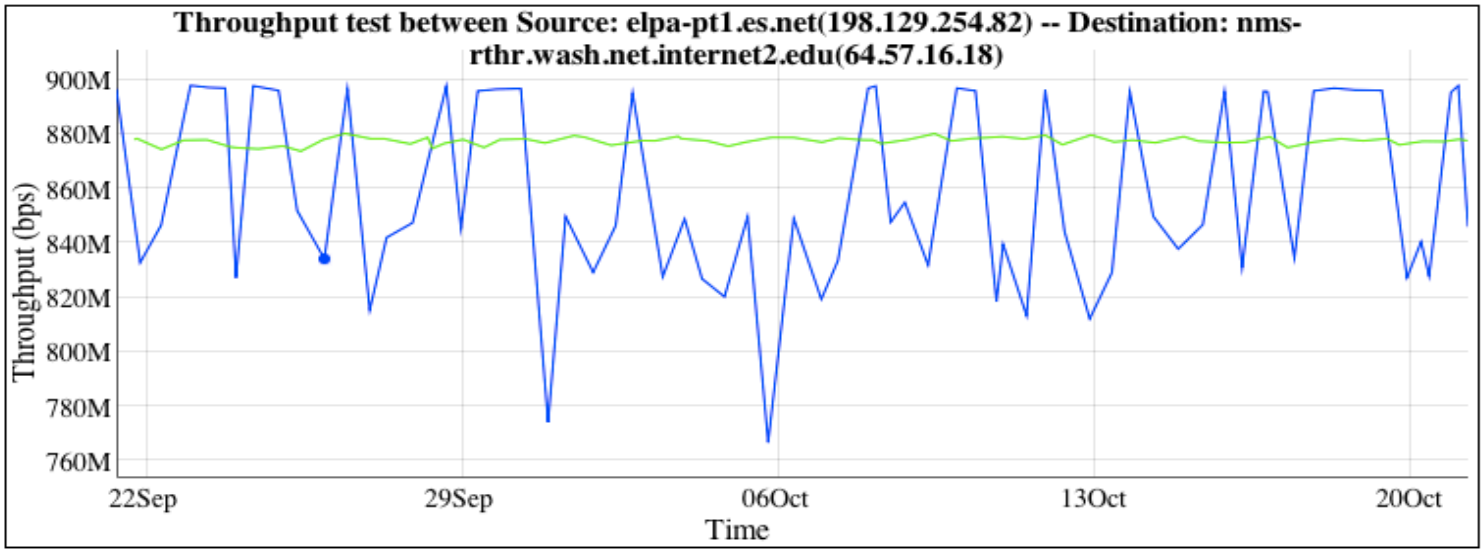
ESnet

Speed Mismatch Issues

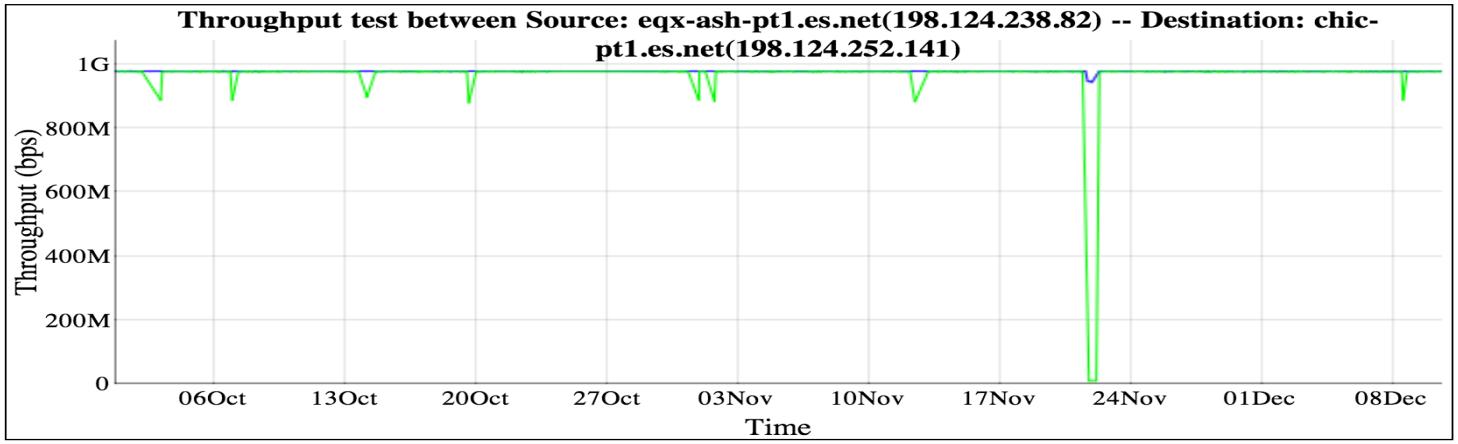
- Sometimes we see problems sending from a faster host to a slower host
  - This can look like a network problem (lots of TCP retransmits)
  - Actually a receive host issue
- This may be true for:
  - 10G to 1G host
  - 10G host to a 2G circuit
  - 40G to 10G host
  - Fast host to slower host

ESnet

Example: 10G Host to a 1G host



Note: with larger buffers 10G to 1G can work just fine too….



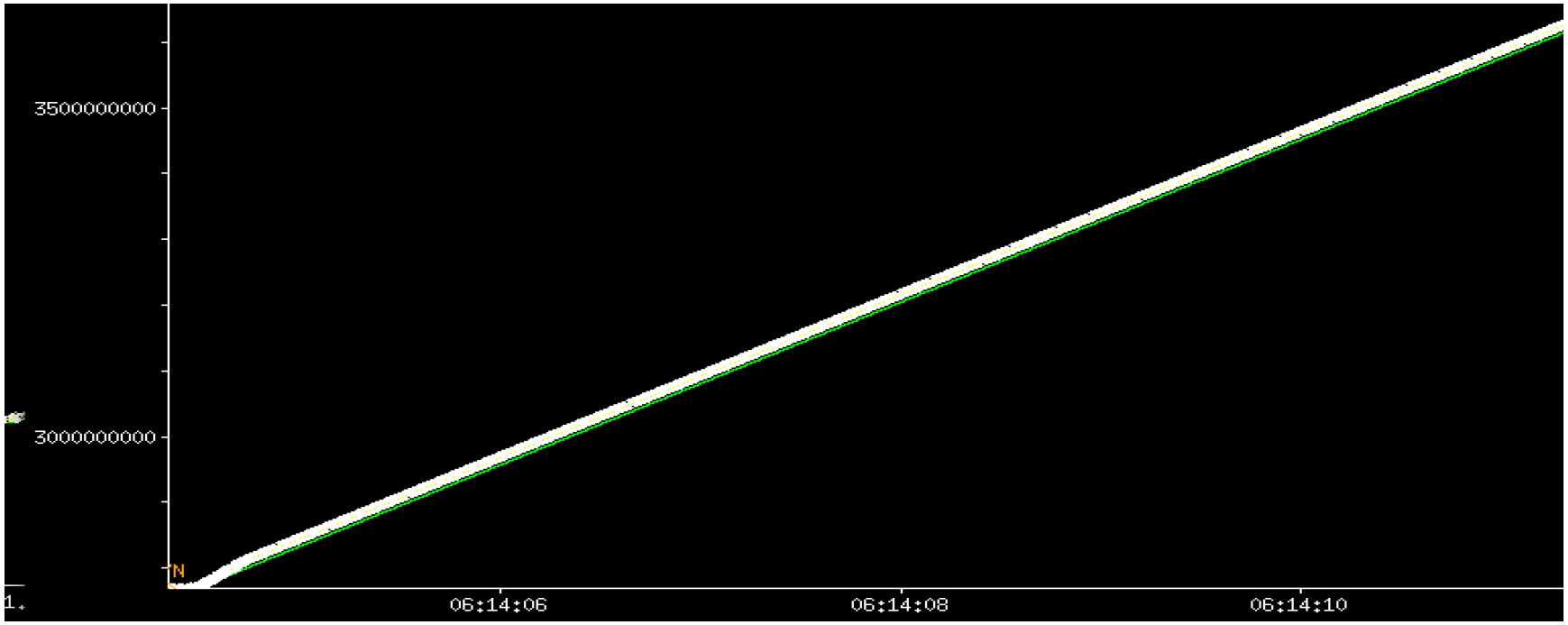
<- 1 month

1 month ->

ESnet

Compare tcpdumps:

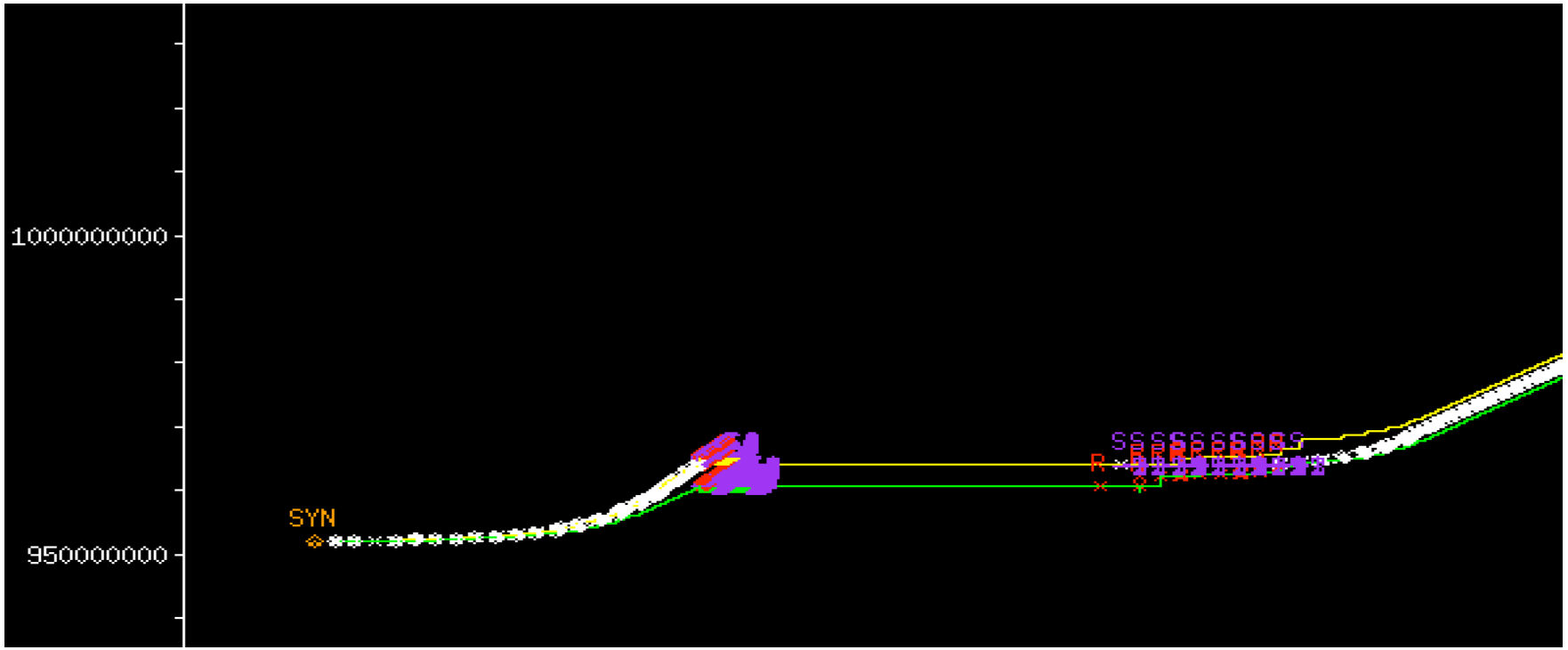
kans-pt1.es.net (10G) to eqx-chi-pt1.es.net (1G)



ESnet

Compare tcpdumps

kans-pt1.es.net (10G) to uct2-net4.uchicago.edu (1G)



ESnet

Sample 40G results: Fast host to Slower Host

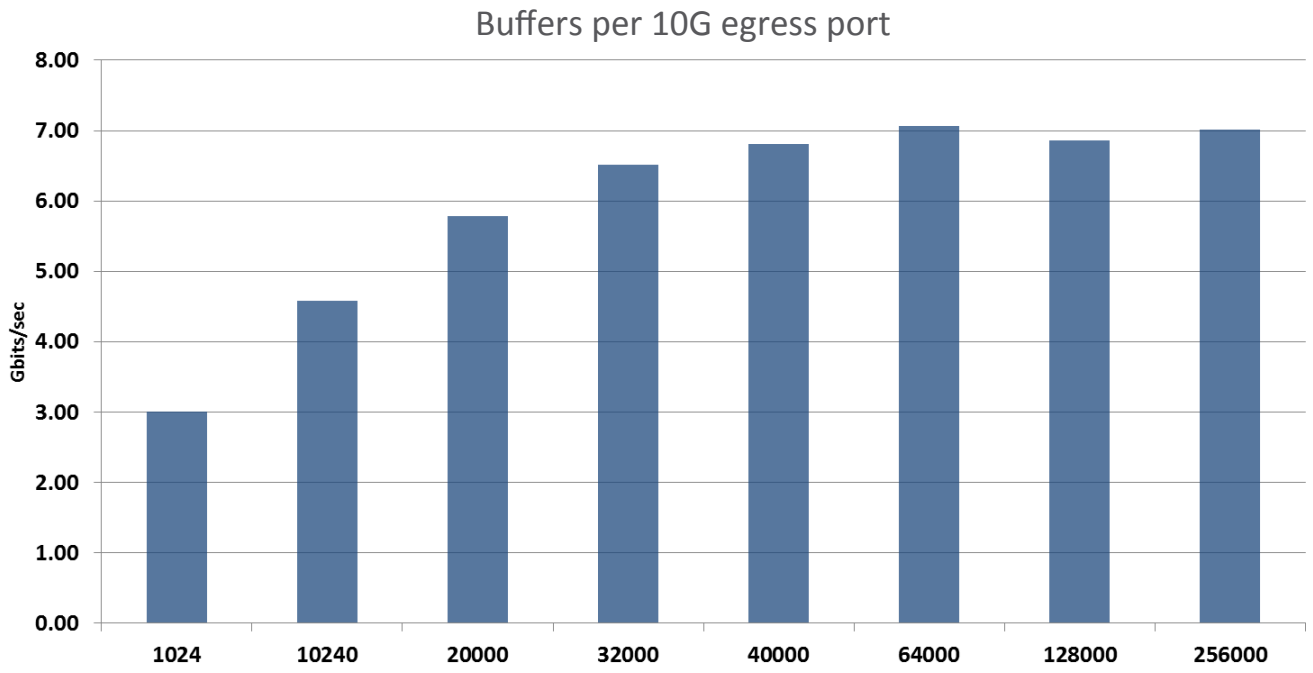
Intel(R) Xeon(R) CPU 2.90GHz to 2.00GHz

```
nuttcp -i1 192.168.2.31
  410.7500 MB /   1.00 sec = 3445.5139 Mbps     0 retrans
  339.5625 MB /   1.00 sec = 2848.4966 Mbps     0 retrans
  354.5625 MB /   1.00 sec = 2974.2888 Mbps   350 retrans
  326.3125 MB /   1.00 sec = 2737.3022 Mbps     0 retrans
  377.7500 MB /   1.00 sec = 3168.8220 Mbps   179 retrans

nuttcp –r -i1 192.168.2.31  (reverse direction)
 2091.0625 MB /   1.00 sec = 17540.8230 Mbps     0 retrans
 2106.7500 MB /   1.00 sec = 17672.0814 Mbps     0 retrans
 2103.6250 MB /   1.00 sec = 17647.0326 Mbps     0 retrans
 2086.7500 MB /   1.00 sec = 17504.7702 Mbps     0 retrans
```

ESnet

Issues due to lack of network device buffering



Slide from Michael Smitasin, LBNL

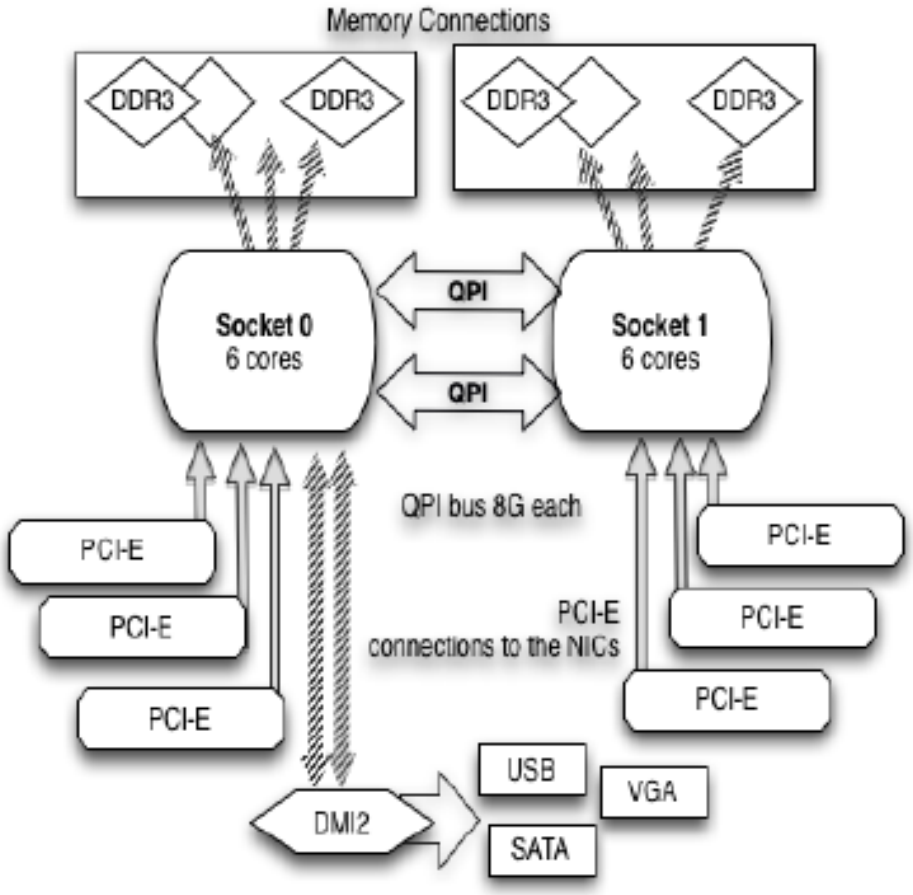
[1] NI-MLX-10Gx8-M

Tunable Buffers with a Brocade MLXe[1]

ESnet

NUMA Issues

- NUMA Architecture means that data from NIC may have to traverse QPI
- This is an issue for 40G/ 100G hosts



NUMA Issues

Sample results: TCP On Intel “Sandy Bridge” Motherboards

30% Improvement using the right core!

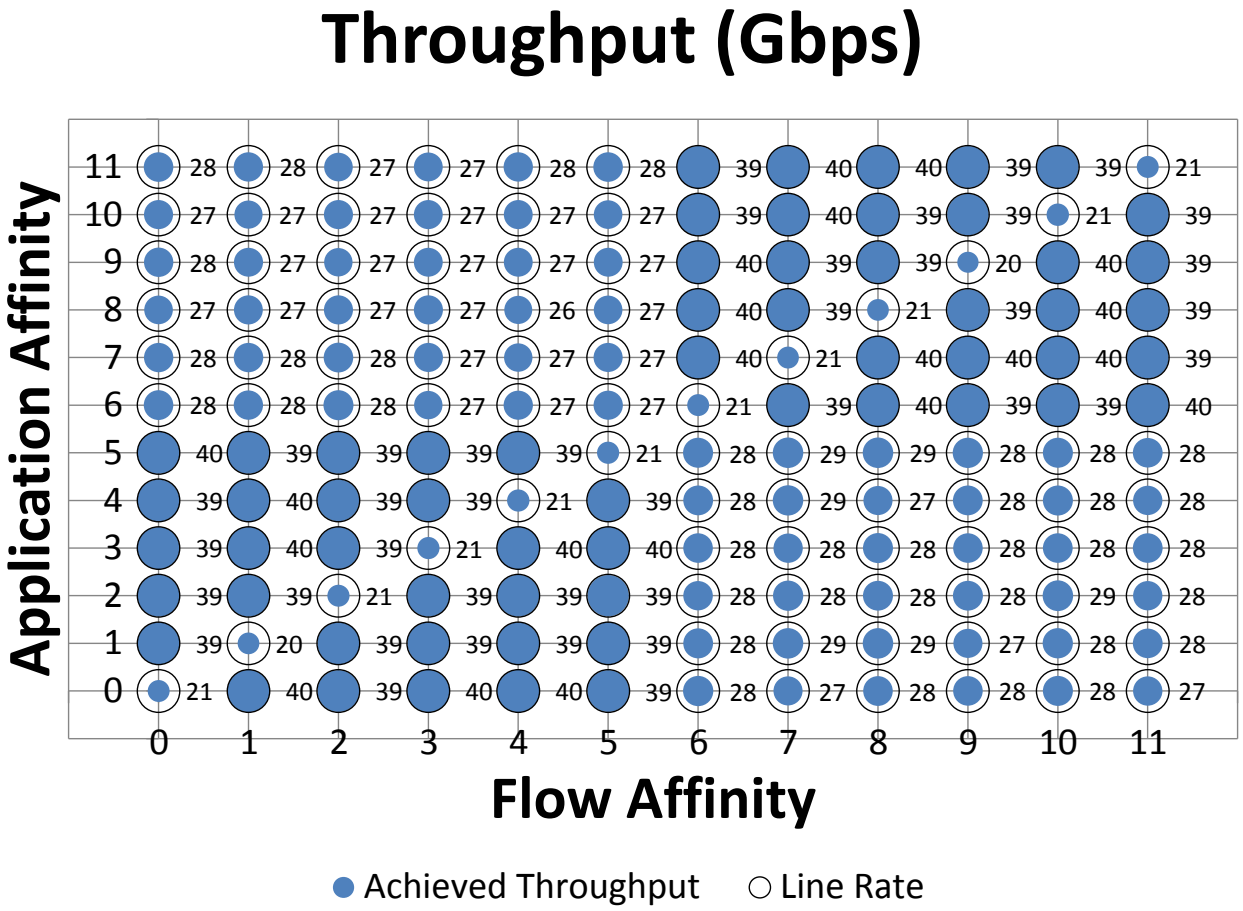
```
nuttcp -i 192.168.2.32
 2435.5625 MB /   1.00 sec = 20429.9371 Mbps     0 retrans
 2445.1875 MB /   1.00 sec = 20511.4323 Mbps     0 retrans
 2443.8750 MB /   1.00 sec = 20501.2424 Mbps     0 retrans
 2447.4375 MB /   1.00 sec = 20531.1276 Mbps     0 retrans
 2449.1250 MB /   1.00 sec = 20544.7085 Mbps     0 retrans

nuttcp -i1 -xc 2/2 192.168.2.32
 3634.8750 MB /   1.00 sec = 30491.2671 Mbps     0 retrans
 3723.8125 MB /   1.00 sec = 31237.6346 Mbps     0 retrans
 3724.7500 MB /   1.00 sec = 31245.5301 Mbps     0 retrans
 3721.7500 MB /   1.00 sec = 31219.8335 Mbps     0 retrans
 3723.7500 MB /   1.00 sec = 31237.6413 Mbps     0 retrans
```

nuttcp: http://lcp.nrl.navy.mil/nuttcp/beta/nuttcp-7.2.1.c

ESnet

NUMA issues: Socket and Core Matter



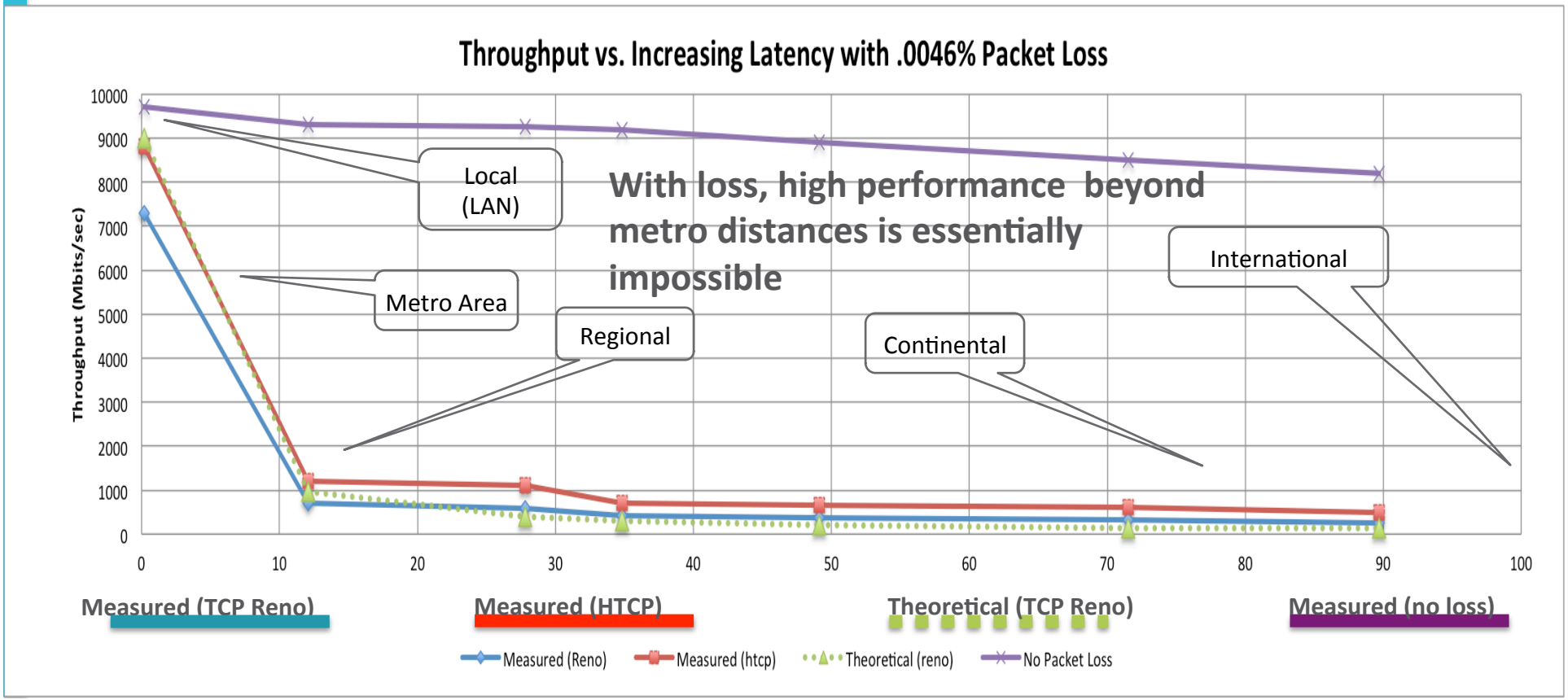
For more details see NDM 2014 paper

ESnet

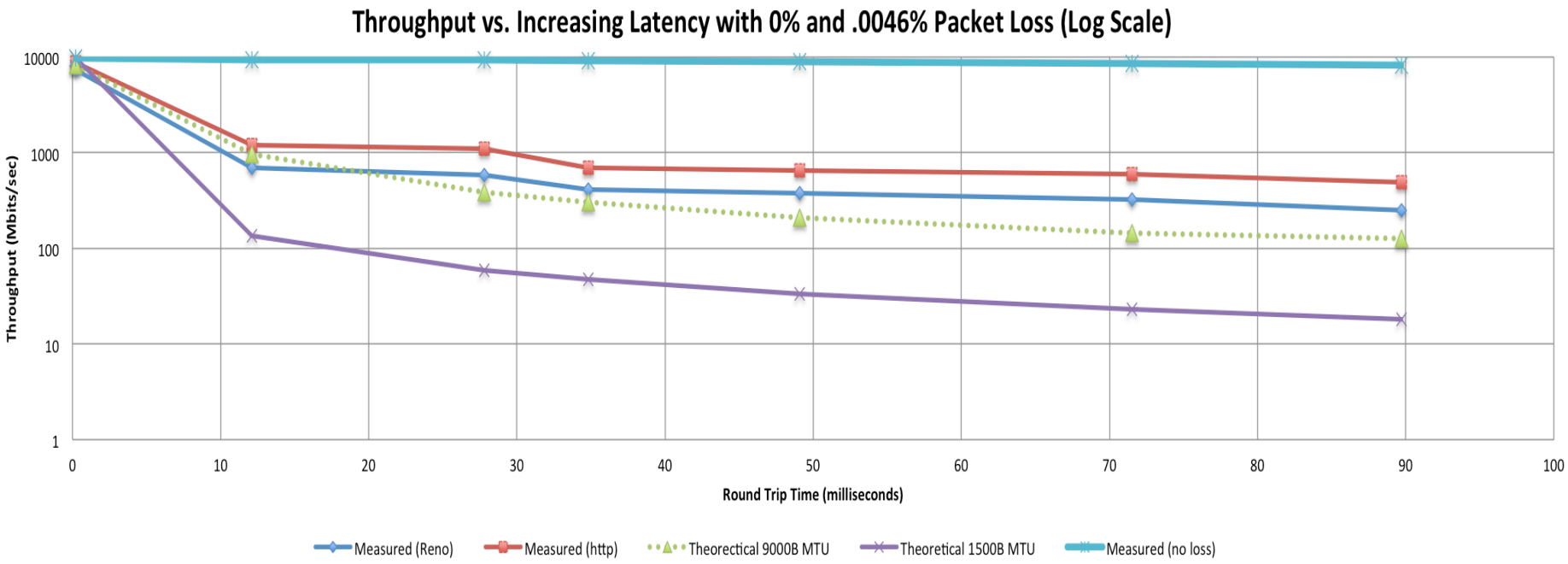
These issues are all worse on long RTT paths due to TCP dynamics

ESnet

A small amount of packet loss makes a huge difference in TCP performance



Impact of Packet Loss on High Latency Paths, Log Scale



ESnet

Q: Can we solve these issues with Packet Pacing?

ESnet

Packet Pacing Techniques

- Hardware-based Pacing
  - E.g.: “Data Reservoir” project from Univ Tokyo
  - First demonstrated advantage of packet pacing in 2004 using “TGNLE-1” (FPGA-based NIC)
- Kernel-based pacing
  - Linux Hierarchical Token Bucket (HTB) queue and traffic rates that are slightly below the bottleneck capacity greatly improve TCP performance.
  - E.g: For a 10Gbps host sending to a 1Gbps circuit on a 36ms RTT path, performance went from 25Mbps to 825Mbps, a dramatic improvement!
- Sample Linux tc example to set up a 900Mbps shaper to a particular subnet.

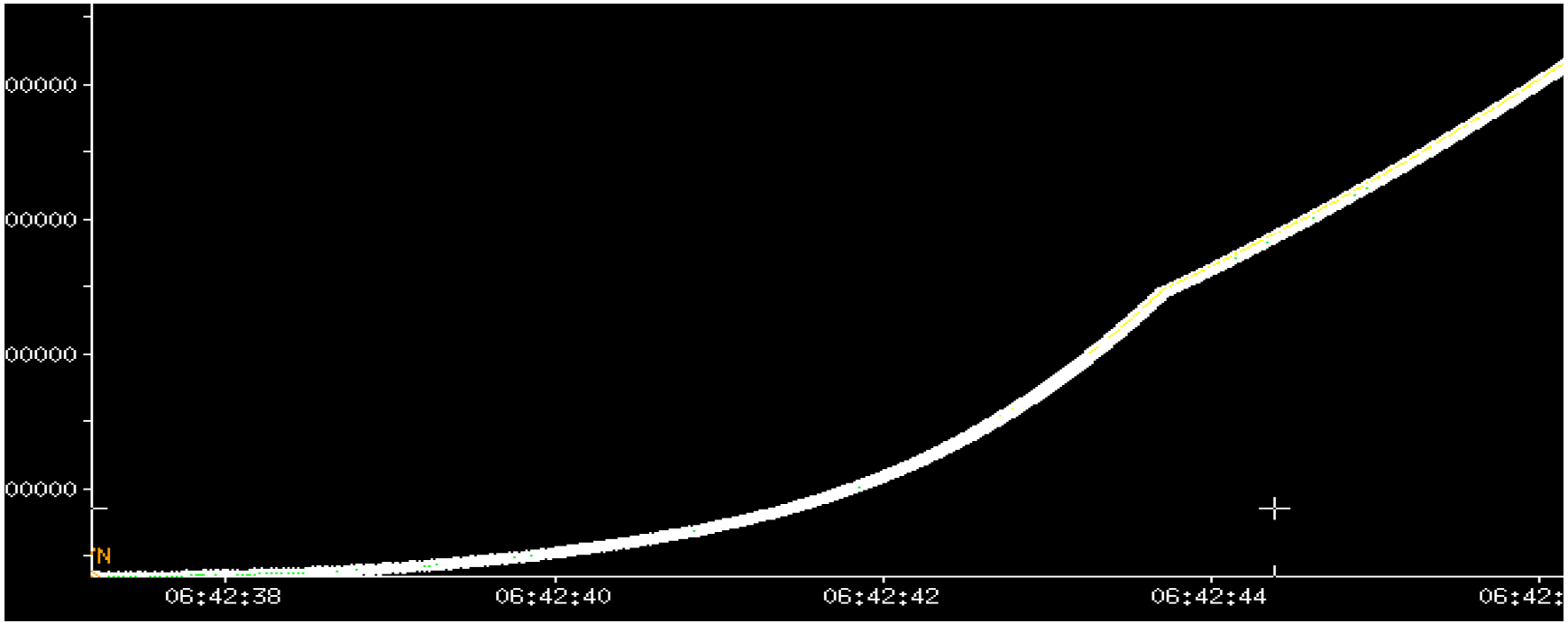
```
#create a Hierarchical Token Bucket
/sbin/tc qdisc add dev eth0 handle 1: root htb
#add a 'class' to our route queue with a rate of 900Mbps
/sbin/tc class add dev eth0 parent 1: classid 1:1 htb rate 900mbit
#create a filter that restricts our tc queue and class to a specific source subnet
/sbin/tc filter add dev eth0 parent 1: protocol ip prio 1 u32 match ip dst X.Y.Z.0/24 flowid 1:1
```

- See: http://fasterdata.es.net/host-tuning/packet-pacing/

ESnet

10G host to 1G host with Pacing

kans-pt1.es.net (10G) to uct2-net4.uchicago.edu (1G) with pacing



ESnet

Packet Pacing Helps at 40G too

40G to 10G tests

No Pacing		
Retransmissions (average)	RTT	Throughput
367	.5 ms	9.4 Gbps
826	47 ms	7.1 Gbps
With Pacing		
Retransmissions (average)	RTT	Throughput
.3	.5 ms	9.8 Gbps
.2	47 ms	9.3 Gbps

ESnet

Q: Can we write a daemon that will automatically figure out what to pace, and how much?

ESnet

DTN Tuning Daemon Goals

- Develop a DTN Tuning daemon that can pace TCP so that fewer packets are dropped
  - Help with fast host to slow host issues
  - Help with under buffered switch issues
  - Help with under powered firewall issues
  - Better allocation of available resources
- Use end-system awareness to put “back pressure” closer to the sender: avoid invoking TCP congestion avoidance

ESnet

Can we build a tool to auto-pace?

- Several tools exist to get detailed information from TCP sockets
  - Lots of information ends up in /proc on Linux
- Create a database of previous TCP sessions on the DTN
- Based on the analysis of internal per-flow TCP parameters, apply packet pacing
- Use ‘tc’ to pace flows to certain endpoints

- (More details are in our paper)

ESnet

TCP instrumentation

We explored the following options:

- ss (socket statistics)
  - http://man7.org/linux/man-pages/man8/ss.8.html
- TCP Probe kernel module
  - http://www.linuxfoundation.org/collaborate/workgroups/networking/tcpprobe
- tstat TCP Statistics tool : http://tstat.polito.it/
- Netlink and the NETLINK_INET_DIAG socket-family
  - http://kristrev.github.io/2013/07/26/passive-monitoring-of-sockets-on-linux/

ESnet

Sample ‘ss’ tool (ss –it) output

- Older kernels (e.g.: CentOS):

```
    htcp wscale:5,13 rto:315 rtt:104.75/22.75 cwnd:5
ssthresh:4 send 522.4Kbps rcv_space:17896
```

- Newer kernels (e.g.: Debian):

```
    htcp wscale:5,13 rto:321 rtt:118.453/20.801 mss:1368
cwnd:3 ssthresh:3 send 277.2Kbps lastsnd:43 lastrcv:
1212906624 lastack:43 pacing_rate 923.9Kbps unacked:6
retrans:1/8127 lost:1 sacked:3 rcv_space:26844
```

ESnet

Why this turned out to be hard

- We don't know the actual link capacity of the receivers beforehand
- Heuristic condition used: max cwnd >> avg cwnd and losses occurred
  - High max cwnd shows sender witnessed a high BDP.
  - Low average cwnd
- Problem: the above case could still come from a variety of sources
  - dirty fiber optic connection
  - Actual congestion
  - Receiver over wireless link/link with changing throughput
- Many "slow" transfers over known fast links were due to disk read/write limitations
- We needed more data, but the data source is changing…

ESnet

New Features in the Linux Kernel

ESnet

TSO sizing and the FQ scheduler to the rescue

- New enhancements to Linux Kernel make a huge difference!

- TSO Sizing
  - On by default starting with 3.12 kernel
- FQ Scheduler
  - tc qdisc add dev $ETH root fq

https://lwn.net/Articles/564978/

ESnet

TCP segmentation offload (TSO) fixes

- TCP segmentation offload (TSO) is a hardware-assisted technique to improve performance of outbound data
  - A NIC that supports TSO can accept a large buffer of data and do segmentation in hardware.
  - This reduces the load on the host CPU, making the transmission process more efficient.
  - This was a good idea 10-15 years ago
- Problem with TSO on fast hosts
  - causes the NIC to dump a large number of packets onto the wire in a short period of time.
  - Packets end up sitting in a buffer somewhere, contributing to bufferbloat and increasing the chances that some of those packets will be dropped
  - Impact worse on high latency paths due to TCP dynamics
- If those packets were transmitted at a more steady pace, the stress on the network is reduced
  - and throughput will increase!

ESnet

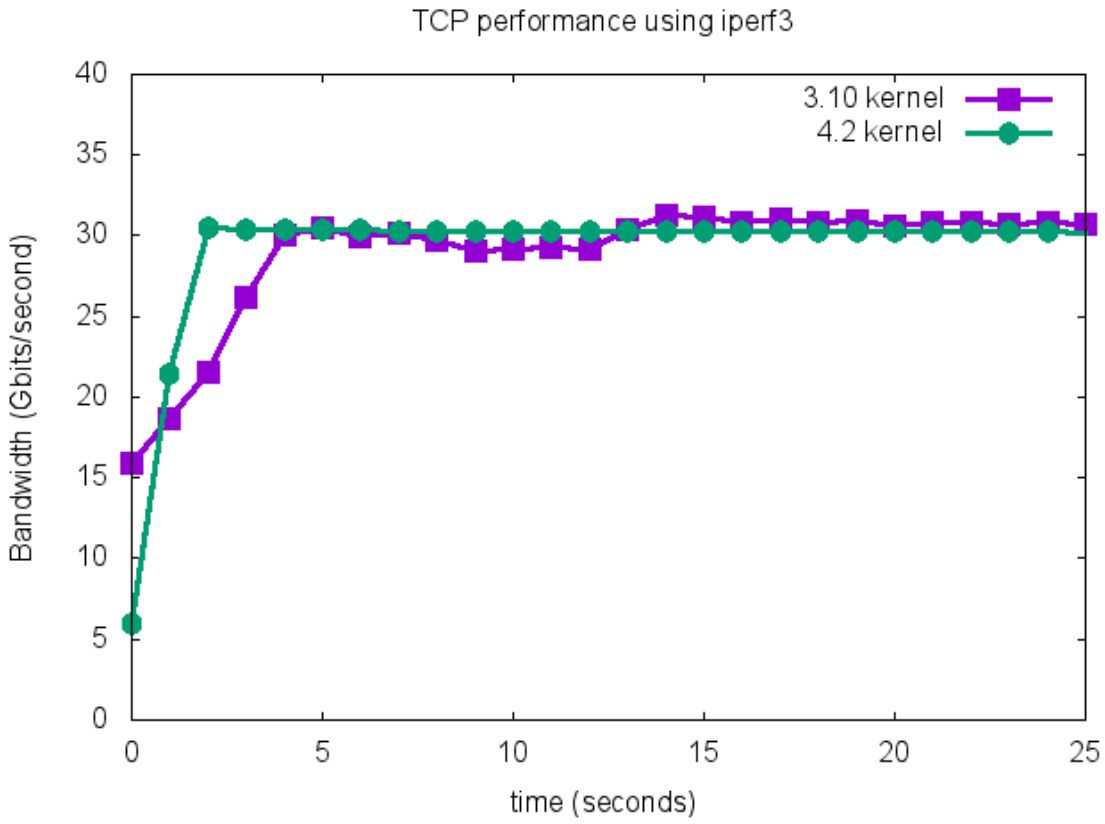
https://lwn.net/Articles/564978/

TSO automatic sizing

- New "TSO automatic sizing" tries to spread out transmissions more evenly
- New idea: make intelligent choices about how much data should be handed to the interface in a single TSO transmission.
  - With the automatic sizing patch, that buffer size is reduced to an amount that will take roughly 1ms to transmit at the current flow rate.
- Result: each transmission will produce a smaller burst of data

ESnet

TCP comparison 3.10 vs 4.2 kernel, 40G hosts

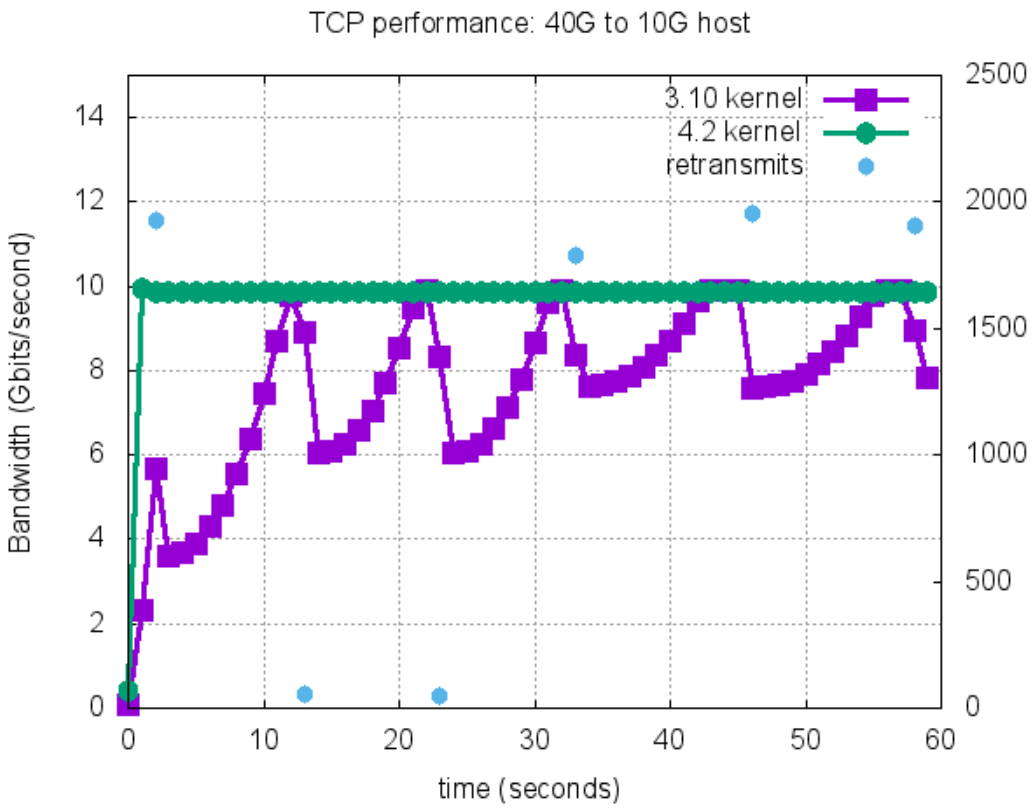


RTT = 90ms

ESnet

TCP comparison 3.10 vs 4.2 kernel 40G to 10G hosts

RTT = 47ms



ESnet

Fair Queuing

- FQ (Fair Queue) is a classless packet scheduler designed to achieve per flow pacing.
- An application can specify a maximum pacing rate using SO_MAX_PACING_RATE setsockopt call.
  - This packet scheduler adds delay between packets to respect rate limitation set by TCP stack.
- Dequeueing happens in a round-robin fashion.
  - A special FIFO queue is reserved for high priority packets (TC_PRIO_CONTROL priority), such packets are always dequeued first.
- TCP pacing is good for flows having idle times, as the congestion window permits TCP stack to queue a possibly large number of packets.
  - This removes 'slow start after idle', which hits large BDP flows in particular

http://man7.org/linux/man-pages/man8/tc-fq.8.html

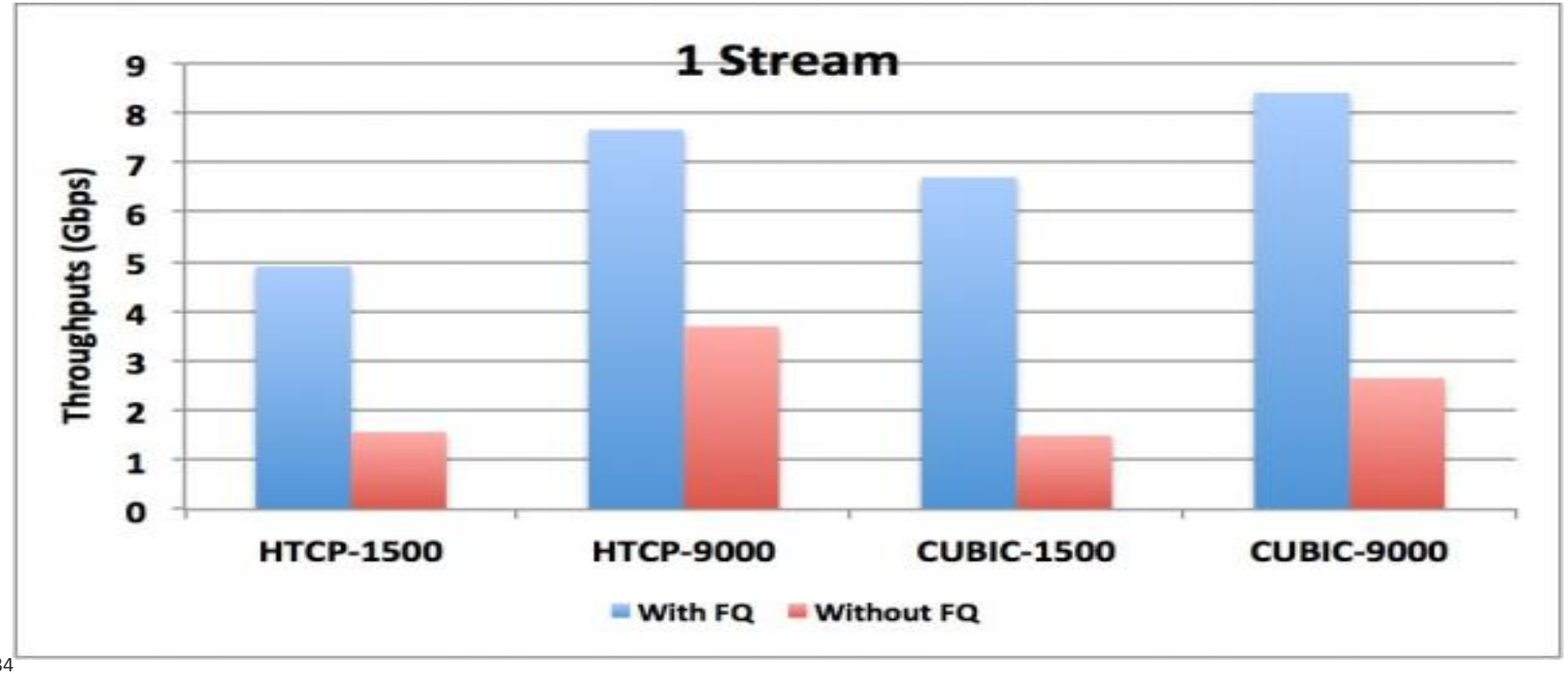
History of TSO autosizing and FQ Updates

- Added to 3.12 kernel in August 2013
  - Written by Eric Dumazet, Google
- Releases:
  - Ubuntu 13.10 (October 2013)
  - Fedora 20 (December 2013)
  - Debian 8 (April 2015)
  - Unfortunately RHEL 7.1 / CentOS 7.1 still using 3.10 kernel
    - Back-port of TCP stack from 3.18 will be in 7.2 (beta available now):
      - https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7-Beta/html/7.2_Release_Notes/networking.html
    - Beta came out in September, releases are usually 6-7 months later
    - Note: you can easily install the most recent "stable" kernel from kernel.org on RHEL6/7 using packages from elrepo.org

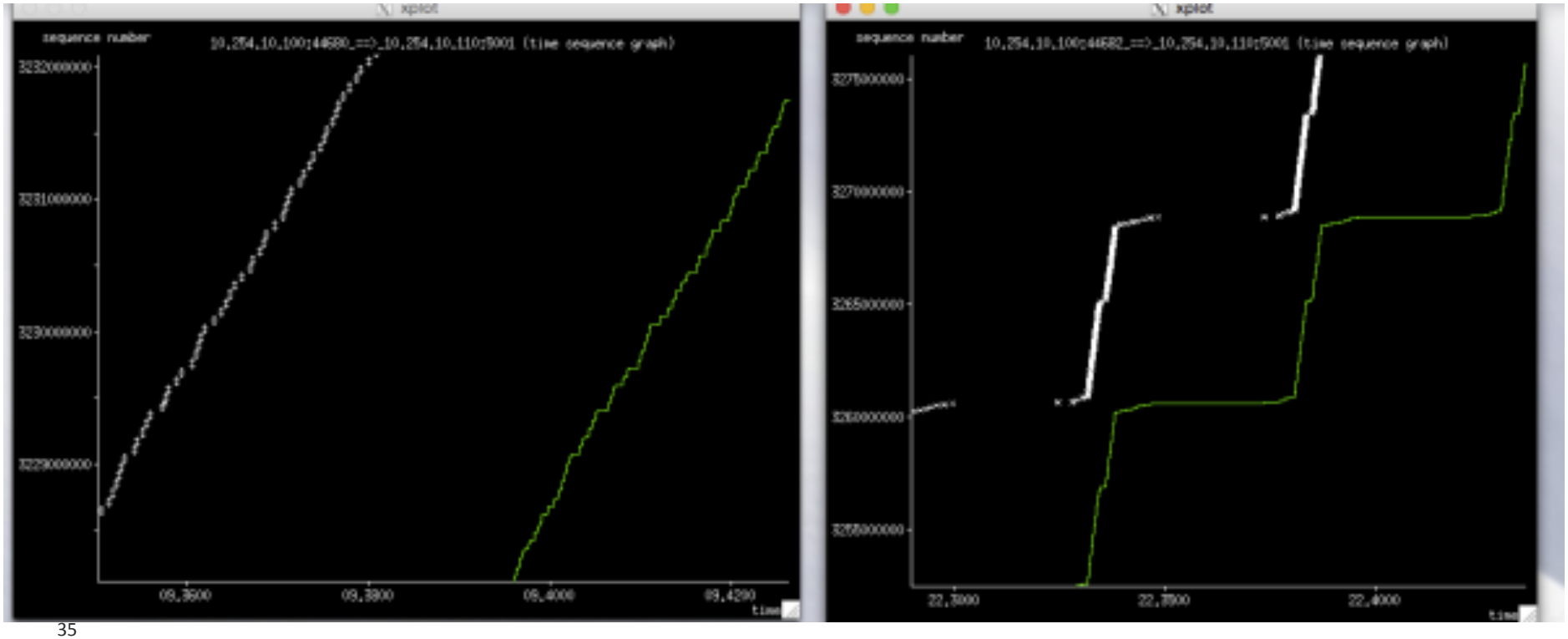
ESnet

Linux Fair Queuing Testing: Wenji Wu, FNAL

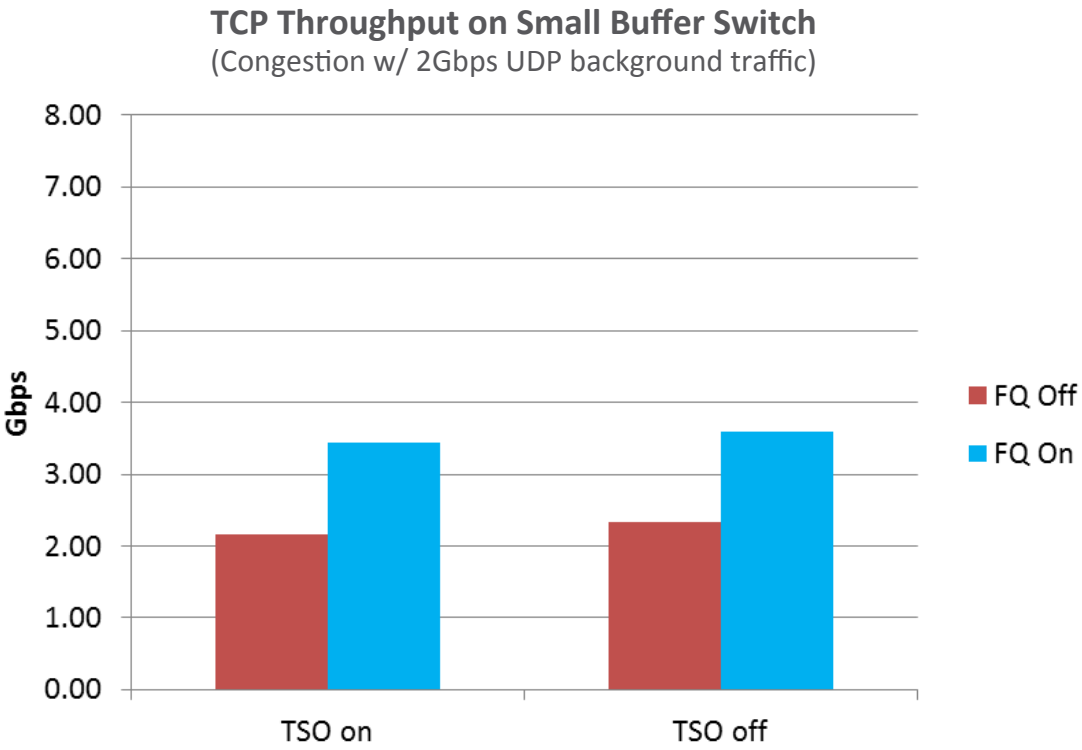
- tc qdisc add dev $ETH root fq



FQ Results: tcptrace/xplot (FQ on left)



Linux Fair Queuing Testing: Michael Smitasin, LBNL



Results from older Arista 7120T with very small buffers

- `tc qdisc add dev EthN root fq`
  - Enable Fair Queuing
- Pacing side effect of Fair Queuing yields ~1.25Gbps increase in throughput @ 10Gbps on our hosts
- TSO differences still negligible on our hosts w/ Intel X520

ESnet

Summary and Conclusions

ESnet

Problem Solved?

- Maybe?
  - Need lots more testing, but so far looks very promising
  - Please try it out and let us know what you find.
- Improvement may be due to other changes in the kernel and NIC drivers as well

ESnet

Lots of useful information on fasterdata.es.net

- http://fasterdata.es.net/host-tuning/40g-tuning/
- http://fasterdata.es.net/host-tuning/packet-pacing/
- http://fasterdata.es.net/host-tuning/linux/fair-queuing-scheduler/
- http://fasterdata.es.net/host-tuning/linux/recent-tcp-enchancements/
- http://fasterdata.es.net/performance-testing/troubleshooting/network-troubleshooting-quick-reference-guide/

ESnet