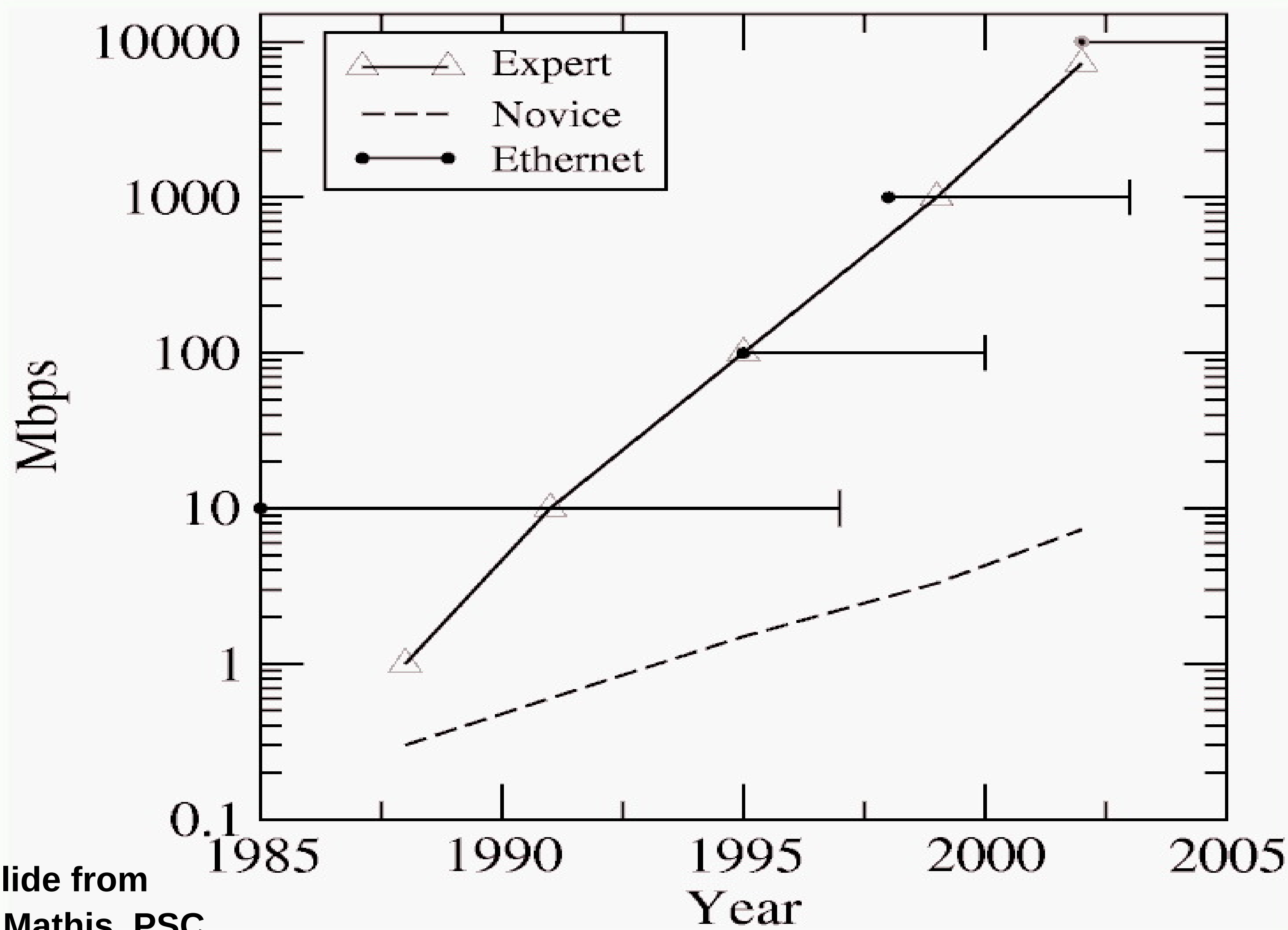
TCP Tuning Techniques for High-Speed Wide-Area Networks

Brian L. Tierney

Distributed Systems Department
Lawrence Berkeley National Laboratory

Wizard Gap



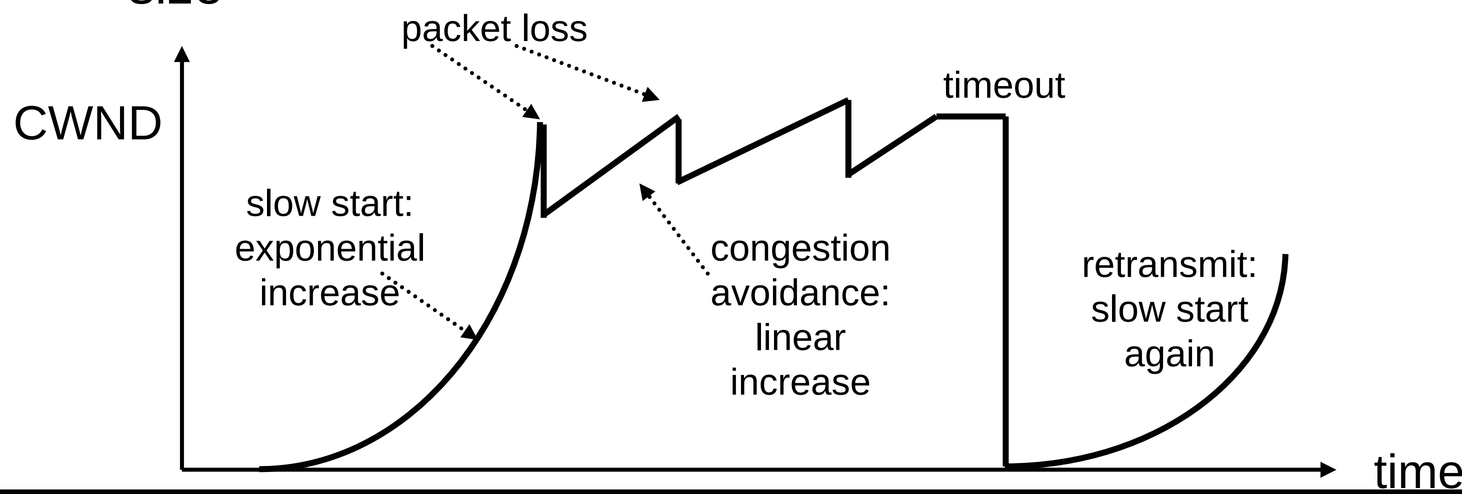
Slide from
Matt Mathis, PSC

Today's Talk

- This talk will cover:
  - Information needed to be a "wizard"
  - Current work being done so you don't have to be a wizard
- Outline
  - TCP Overview
  - TCP Tuning Techniques (focus on Linux)
  - TCP Issues
  - Network Monitoring Tools
  - Current TCP Research

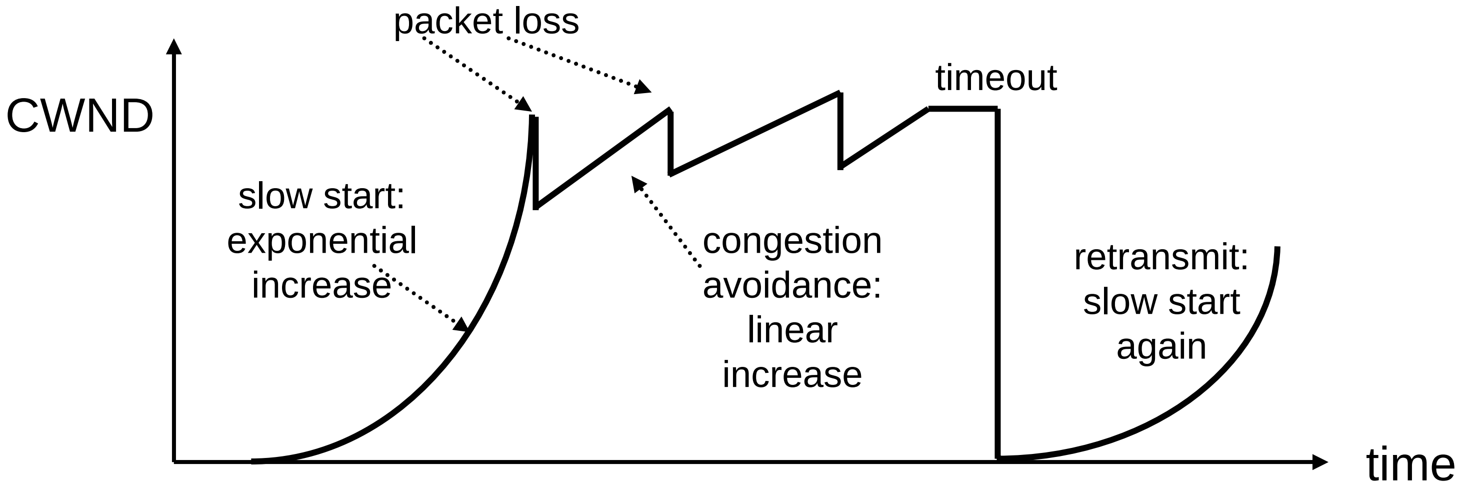
How TCP works: A very short overview

- Congestion window (CWND) = the number of packets the sender is allowed to send
  - The larger the window size, the higher the throughput
    - Throughput = Window size / Round-trip Time
- TCP Slow start
  - exponentially increase the congestion window size until a packet is lost
    - this gets a rough estimate of the optimal congestion window size



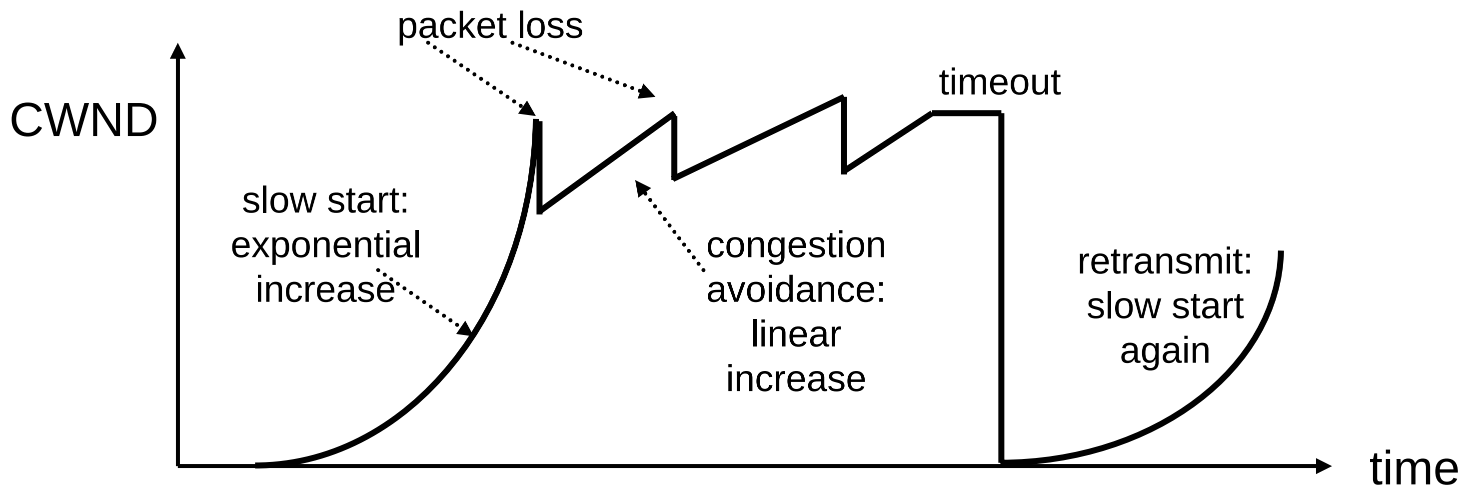
TCP Overview

- Congestion avoidance
  - additive increase: starting from the rough estimate, linearly increase the congestion window size to probe for additional available bandwidth
  - multiplicative decrease: cut congestion window size aggressively if a timeout occurs



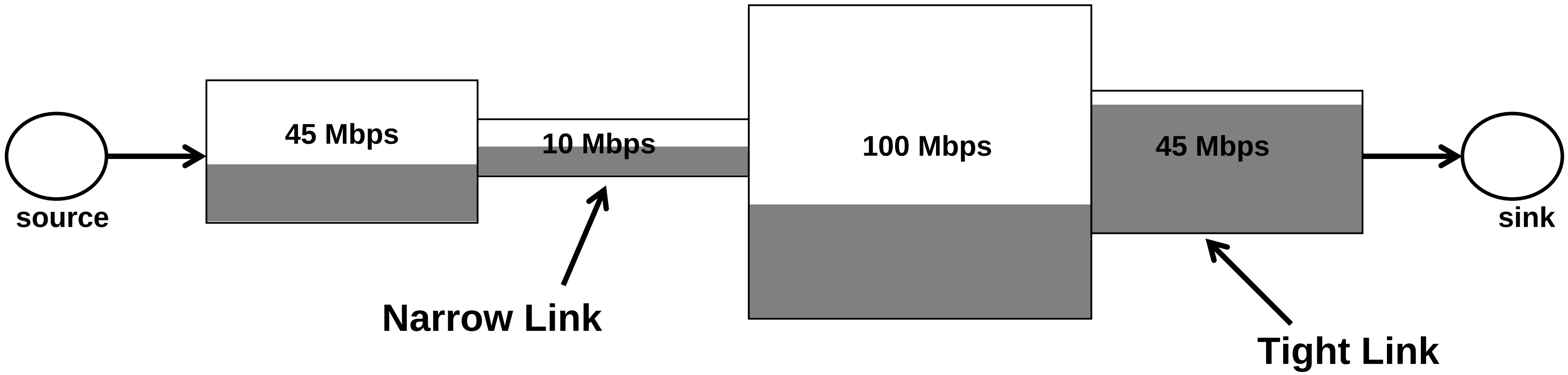
TCP Overview

- Fast Retransmit: retransmit after 3 duplicate acks (got 3 additional packets without getting the one you are waiting for)
  - this prevents expensive timeouts
  - no need to go into “slow start” again
- At steady state, CWND oscillates around the optimal window size
- With a retransmission timeout, slow start is triggered again



Terminology

- The term “Network Throughput” is vague and should be avoided
  - Capacity: link speed
    - Narrow Link: link with the lowest capacity along a path
    - Capacity of the end-to-end path = capacity of the narrow link
  - Utilized bandwidth: current traffic load
  - Available bandwidth: capacity – utilized bandwidth
    - Tight Link: link with the least available bandwidth in a path
  - Achievable bandwidth: includes protocol and host issues



More Terminology

- RTT: Round-trip time
- Bandwidth*Delay Product = BDP
  - The number of bytes in flight to fill the entire path
  - Example: 100 Mbps path; ping shows a 75 ms RTT
    - BDP = 100 * 0.075 / 2 = 3.75 Mbits (470 KB)
- LFN: Long Fat Networks
  - A network with a large BDP

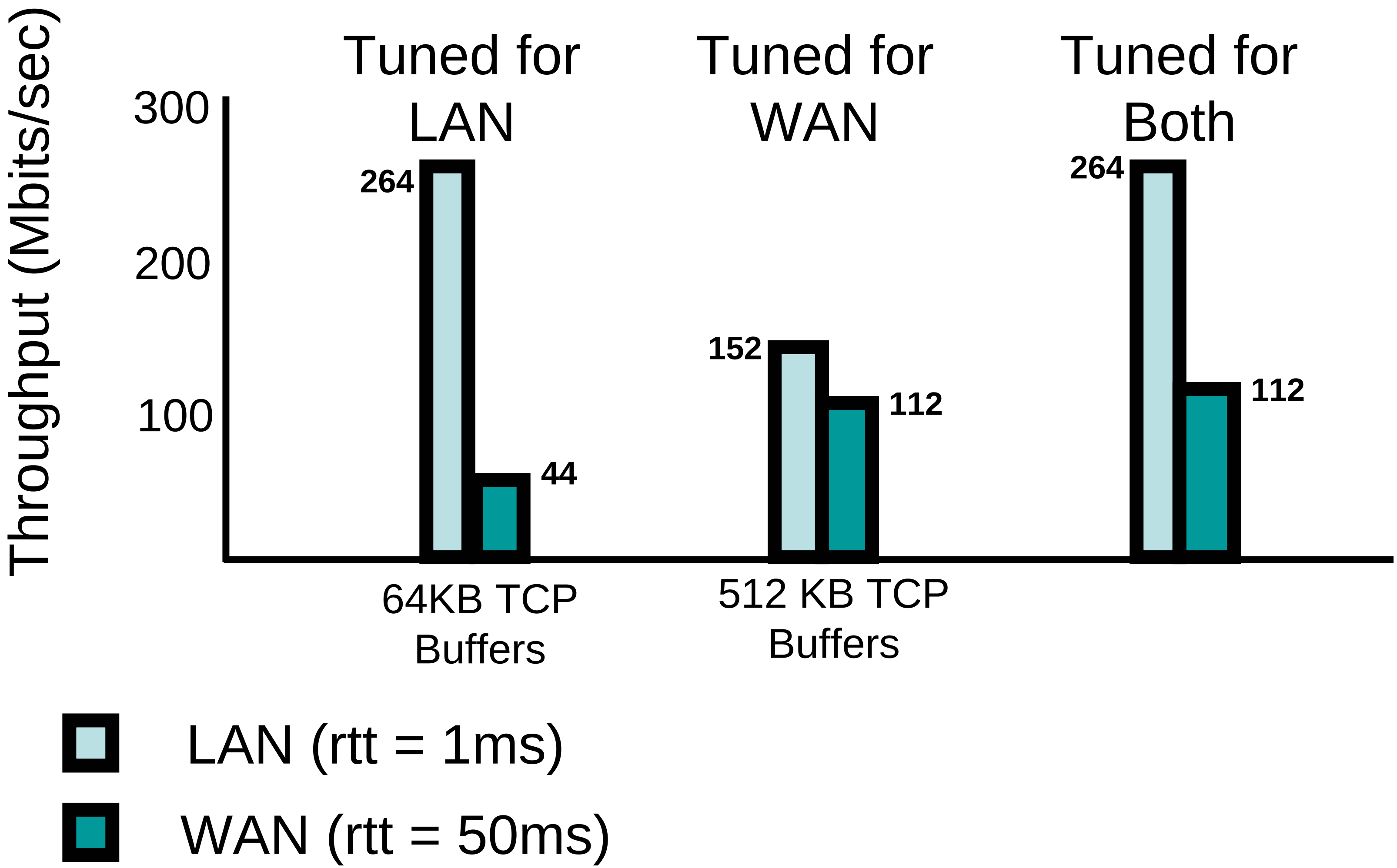
TCP Performance Tuning Issues

- Getting good TCP performance over high-latency high-bandwidth networks is not easy!
- You must keep the pipe full, and the size of the pipe is directly related to the network latency
  - Example: from LBNL (Berkeley, CA) to ANL (near Chicago, IL), the *narrow link* is 1000 Mbits/sec, and the one-way latency is 25ms
    - Bandwidth = 539 Mbits/sec (67 MBytes/sec) (OC12 = 622 Mbps - ATM and IP headers)
  - Need (1000 / 8) * .025 sec = 3.125 MBytes of data “in flight” to fill the pipe

Setting the TCP buffer sizes

- It is critical to use the optimal TCP send and receive socket buffer sizes for the link you are using.
  - Recommended size = 2 x Bandwidth Delay Product (BDP)
  - if too small, the TCP window will never fully open up
  - if too large, the sender can overrun the receiver, and the TCP window will shut down
- Default TCP buffer sizes are way too small for this type of network
  - default TCP send/receive buffers are typically 64 KB
  - with default TCP buffers, you can only get a small % of the available bandwidth!

Importance of TCP Tuning



TCP Buffer Tuning: System

- Need to adjust system max TCP buffer
  - Example: in Linux (2.4 and 2.6) add the entries below to the file /etc/sysctl.conf, and then run "sysctl -p"

```
# increase TCP max buffer size
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
# increase Linux autotuning TCP buffer limits
# min, default, and max number of bytes to use
net.ipv4.tcp_rmem = 4096 87380 16777216
net.ipv4.tcp_wmem = 4096 65536 16777216
```

- Similar changes needed for other Unix OS's
- For more info, see: http://dsd.lbl.gov/TCP-Tuning/

Networks For Non-Networkers

TCP Buffer Tuning: Application

- Must adjust buffer size in your applications:

```
int skt, int sndsize = 2 * 1024 * 1024;
 err = setsockopt(skt, SOL_SOCKET, SO_SNDBUF,
              (char *)&sndsize,(int)sizeof(sndsize));
```

and/or

```
 err = setsockopt(skt, SOL_SOCKET, SO_RCVBUF,
              (char *)&sndsize,(int)sizeof(sndsize));
```

- It's a good idea to check the following:

```
err = getsockopt(skt, SOL_SOCKET, SO_RCVBUF,
   (char *) &sockbufsize,  &size);
If (size != sndsize)
  printf(stderr, "Warning: requested TCP buffer of %d, but only got %d
  \n", sndsize, size);
```

Determining the Buffer Size

- The optimal buffer size is twice the bandwidth*delay product of the link:

  buffer size = 2 * bandwidth * delay

- The ping program can be used to get the delay
  - e.g.: `>ping -s 1500 lxplus.cern.ch`

```
1500 bytes from lxplus012.cern.ch: icmp_seq=0. time=175. ms
1500 bytes from lxplus012.cern.ch: icmp_seq=1. time=176. ms
1500 bytes from lxplus012.cern.ch: icmp_seq=2. time=175. ms
```

- *pipechar* or *pathrate* can be used to get the bandwidth of the slowest hop in your path. (see next slides)
- Since ping gives the round trip time (RTT), this formula can be used instead of the previous one:

  buffer size = bandwidth * RTT

Buffer Size Example

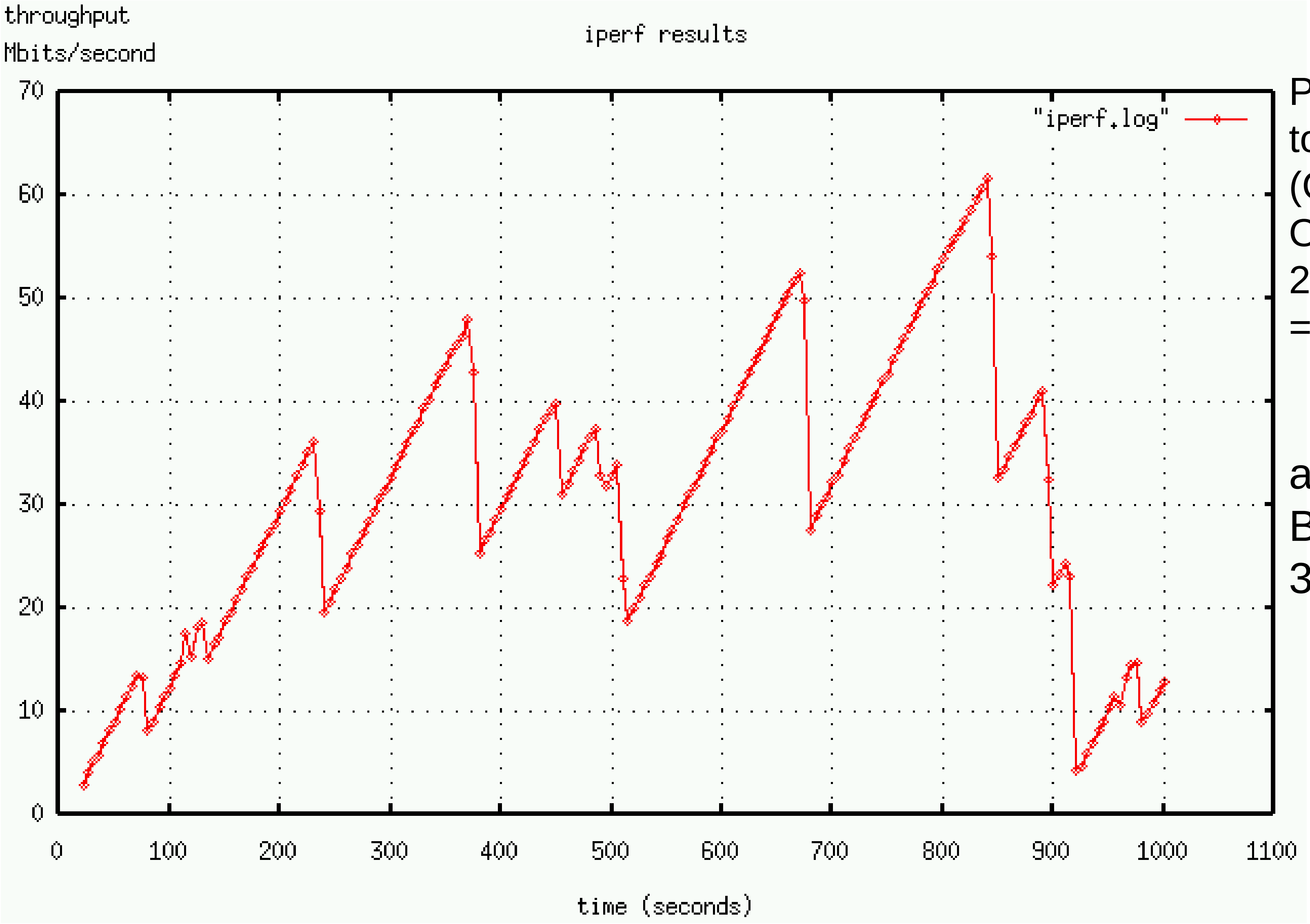
- ping time = 50 ms
- Narrow link = 500 Mbps (62 Mbytes/sec)
  - e.g.: the end-to-end network consists of all 1000 BT ethernet and OC-12 (622 Mbps)
- TCP buffers should be:
  - .05 sec * 62 = 3.1 Mbytes

Sample Buffer Sizes

- UK to...
  - UK (RTT = 5 ms, narrow link = 1000 Mbps) : 625 KB
  - Europe: (25 ms, narrow link = 500 Mbps): 1.56 MB
  - US: (150 ms, narrow link = 500 Mbps): 9.4 MB
  - Japan: (RTT = 260, narrow link = 150 Mbps): 4.9 MB
- Note: default buffer size is usually only 64 KB, and default maximum buffer size for is only 256KB
  - Linux Autotuning default max = 128 KB;
- 10-150 times too small!

Networks For Non-Networkers

More Problems: TCP congestion control

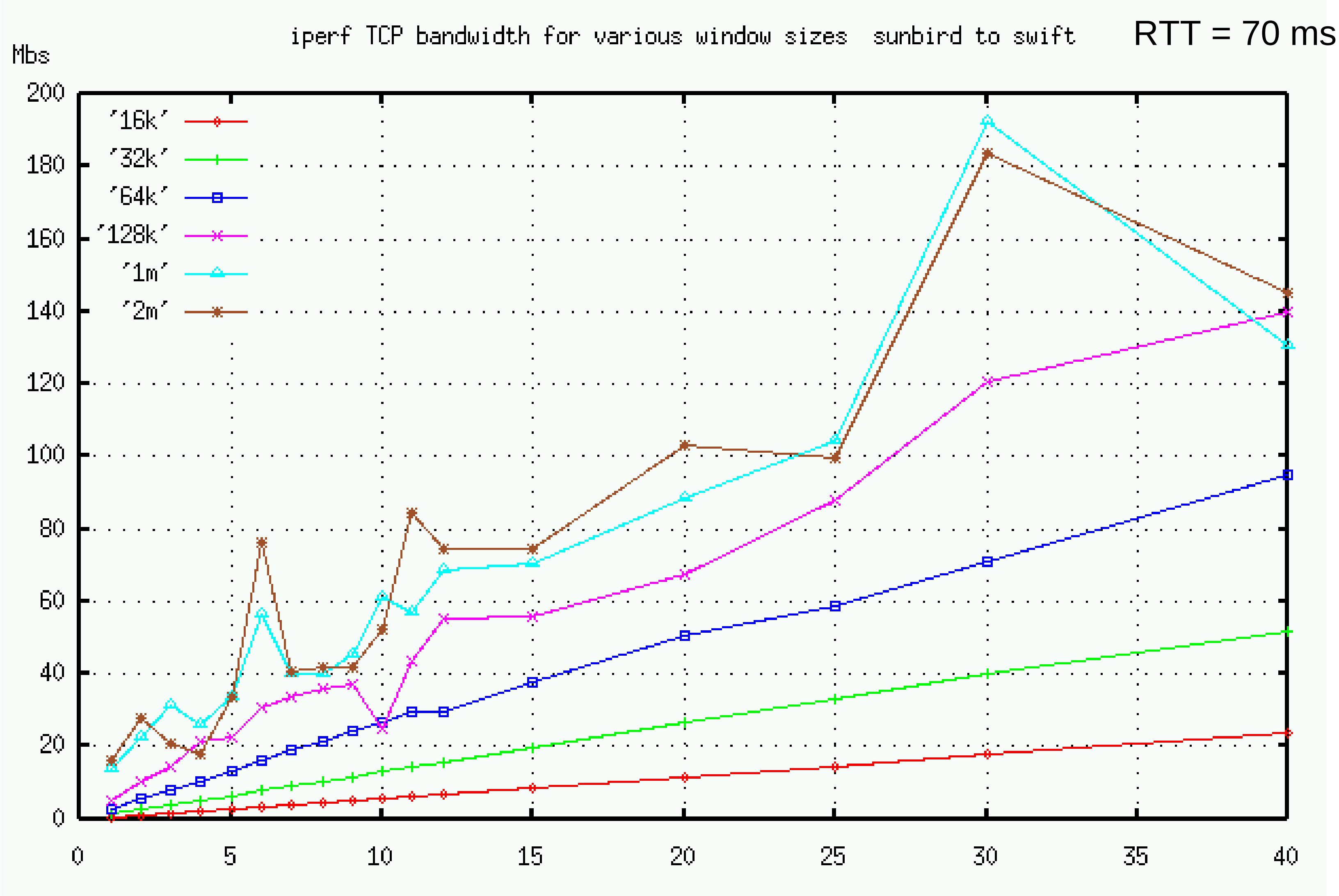


Path = LBL to CERN (Geneva) OC-3, (in 2000), RTT = 150 ms

average BW = 30 Mbps

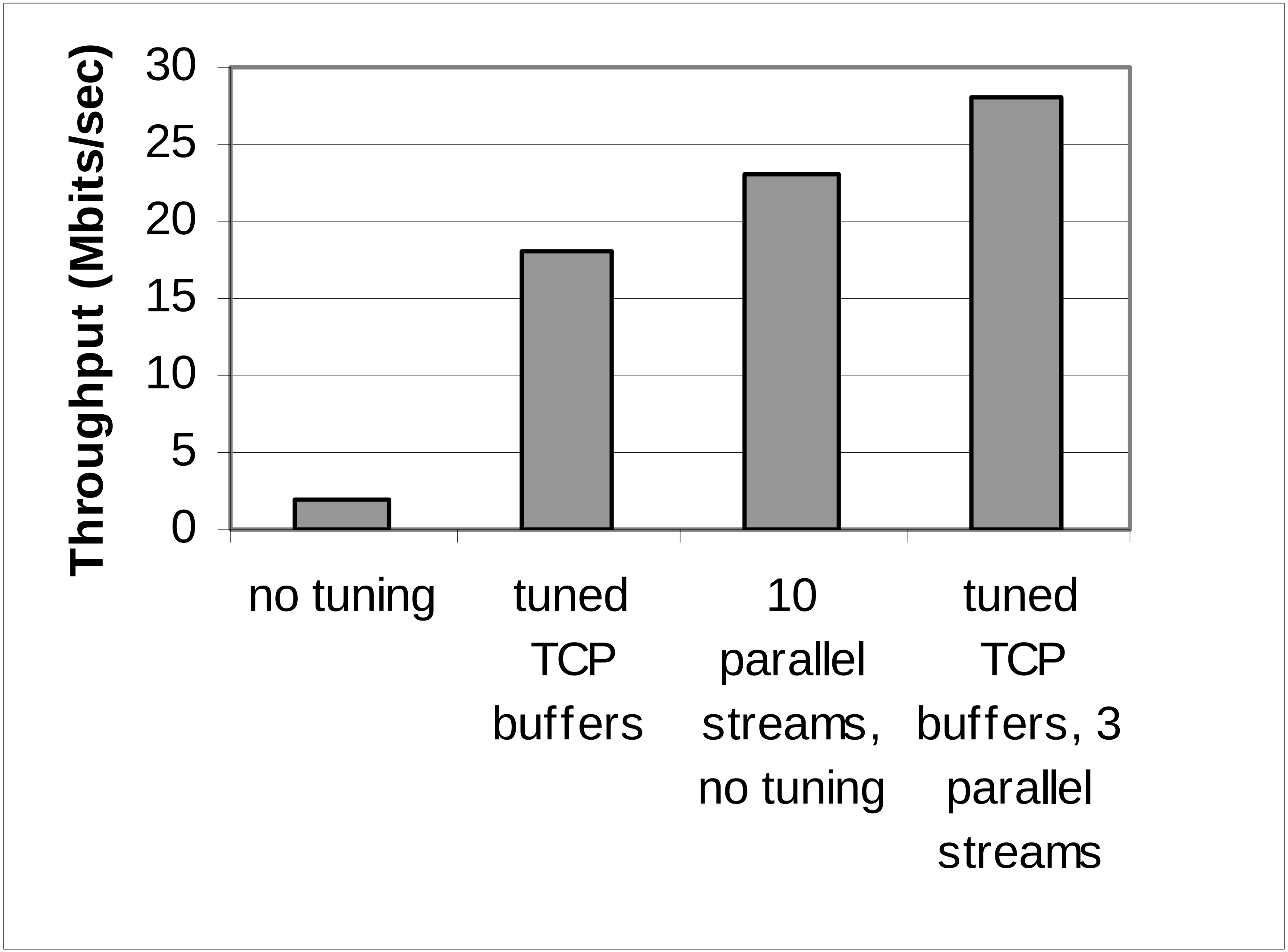
Work-around: Use Parallel Streams

RTT = 70 ms



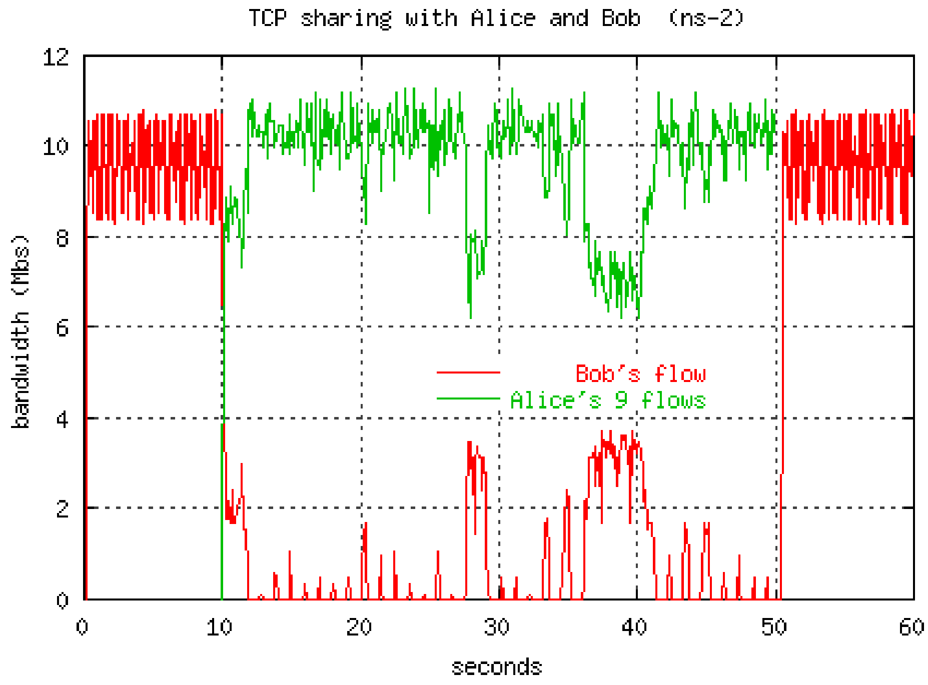
graph from Tom Dunigan, ORNL

Tuned Buffers vs. Parallel Steams



Parallel Streams Issues

- Potentially unfair
- Places more load on the end hosts
- But they are necessary when you don't have root access, and can't convince the sysadmin to increase the max TCP buffers



graph from Tom Dunigan, ORNL

Network Monitoring Tools

traceroute

>traceroute pcgiga.cern.ch

```
traceroute to pcgiga.cern.ch (192.91.245.29), 30 hops max, 40 byte packets
 1  ir100gw-r2.lbl.gov (131.243.2.1)  0.49 ms   0.26 ms   0.23 ms
 2  er100gw.lbl.gov (131.243.128.5)  0.68 ms   0.54 ms   0.54 ms
 3  198.129.224.5 (198.129.224.5)  1.00 ms  *d9*  1.29 ms
 4  lbl2-ge-lbnl.es.net (198.129.224.2)  0.47 ms   0.59 ms   0.53 ms
 5  snv-lbl-oc48.es.net (134.55.209.5)  57.88 ms   56.62 ms   61.33 ms
 6  chi-s-snv.es.net (134.55.205.102)  50.57 ms   49.96 ms   49.84 ms
 7  ar1-chicago-esnet.cern.ch (198.124.216.73)  50.74 ms  51.15 ms  50.96 ms
 8  cernh9-pos100.cern.ch (192.65.184.34)  175.63 ms   176.05 ms   176.05 ms
 9  cernh4.cern.ch (192.65.185.4)  175.92 ms   175.72 ms   176.09 ms
10 pcgiga.cern.ch (192.91.245.29)  175.58 ms   175.44 ms   175.96 ms
```

Can often learn about the network from the router names:

ge = Gigabit Ethernet

oc48 = 2.4 Gbps (oc3 = 155 Mbps, oc12=622 Mbps)

Iperf

- iperf : very nice tool for measuring end-to-end TCP/UDP performance
  - http://dast.nlanr.net/Projects/Iperf/
  - Can be quite intrusive to the network
- Example:
  - Server: iperf -s -w 2M
  - Client: iperf -c hostname -i 2 -t 20 -l 128K -w 2M

```
Client connecting to hostname
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 2.0 sec  66.0 MBytes   275 Mbits/sec
[  3]  2.0- 4.0 sec   107 MBytes   451 Mbits/sec
[  3]  4.0- 6.0 sec   106 MBytes   446 Mbits/sec
[  3]  6.0- 8.0 sec   107 MBytes   443 Mbits/sec
[  3]  8.0-10.0 sec   106 MBytes   447 Mbits/sec
[  3] 10.0-12.0 sec   106 MBytes   446 Mbits/sec
[  3] 12.0-14.0 sec   107 MBytes   450 Mbits/sec
[  3] 14.0-16.0 sec   106 MBytes   445 Mbits/sec
[  3] 16.0-24.3 sec  58.8 MBytes  59.1 Mbits/sec
[  3]  0.0-24.6 sec   871 MBytes   297 Mbits/sec
```

pathrate / pathload

- Nice tools from Georgia Tech:
  - pathrate: measures the capacity of the narrow link
  - pathload: measures the available bandwidth
- Both work pretty well.
  - pathrate can take a long time (up to 20 minutes)
  - These tools attempt to be non-intrusive
- Open Source; available from:
  - http://www.pathrate.org/

pipechar

- Tool to measure hop-by-hop available bandwidth, capacity, and congestion
- Takes 1-2 minutes to measure an 8 hop path
- But not always accurate
  - Results affected by host speed
    - Hard to measure links faster than host interface
  - Results after a slow hop typically not accurate, for example, if the first hop is a wireless link, and all other hops are 100 BT or faster, then results are not accurate
- client-side only tool: puts very little load on the network (about 100 Kbits/sec)
- Available from: http://dsd.lbl.gov/NCS/
  - part of the *netest* package

pipechar output

```
dpsslx04.lbl.gov(59)>pipechar firebird.ccs.ornl.gov
PipeChar statistics: 82.61% reliable
From localhost:    827.586 Mbps GigE (1020.4638 Mbps)
1: ir100gw-r2.lbl.gov          (131.243.2.1 )
|    1038.492 Mbps  GigE   <11.2000 % BW used>
2: er100gw.lbl.gov             (131.243.128.5)
|    1039.246 Mbps  GigE   <11.2000 % BW used>
3: lbl2-ge-lbnl.es.net         (198.129.224.2)
|    285.646 Mbps  congested bottleneck <71.2000%  BW used>
4: snv-lbl-oc48.es.net         (134.55.209.5)
|    9935.817 Mbps  OC192  <94.0002 % BW used>
5: orn-s-snv.es.net            (134.55.205.121)
|    341.998 Mbps congested bottleneck <65.2175 % BW used>
6: ornl-orn.es.net             (134.55.208.62)
|    298.089 Mbps congested bottleneck <70.0007 % BW used>
7: orgwy-ext.ornl.gov          (192.31.96.225)
|    339.623 Mbps congested bottleneck <65.5502 % BW used>
8: ornlgwy-ext.ens.ornl.gov    (198.124.42.162)
|    232.005 Mbps congested bottleneck <76.6233 % BW used>
9: ccsrtr.ccs.ornl.gov         (160.91.0.66 )
|    268.651 Mbps  GigE (1023.4655 Mbps)
10: firebird.ccs.ornl.gov      (160.91.192.165)
```

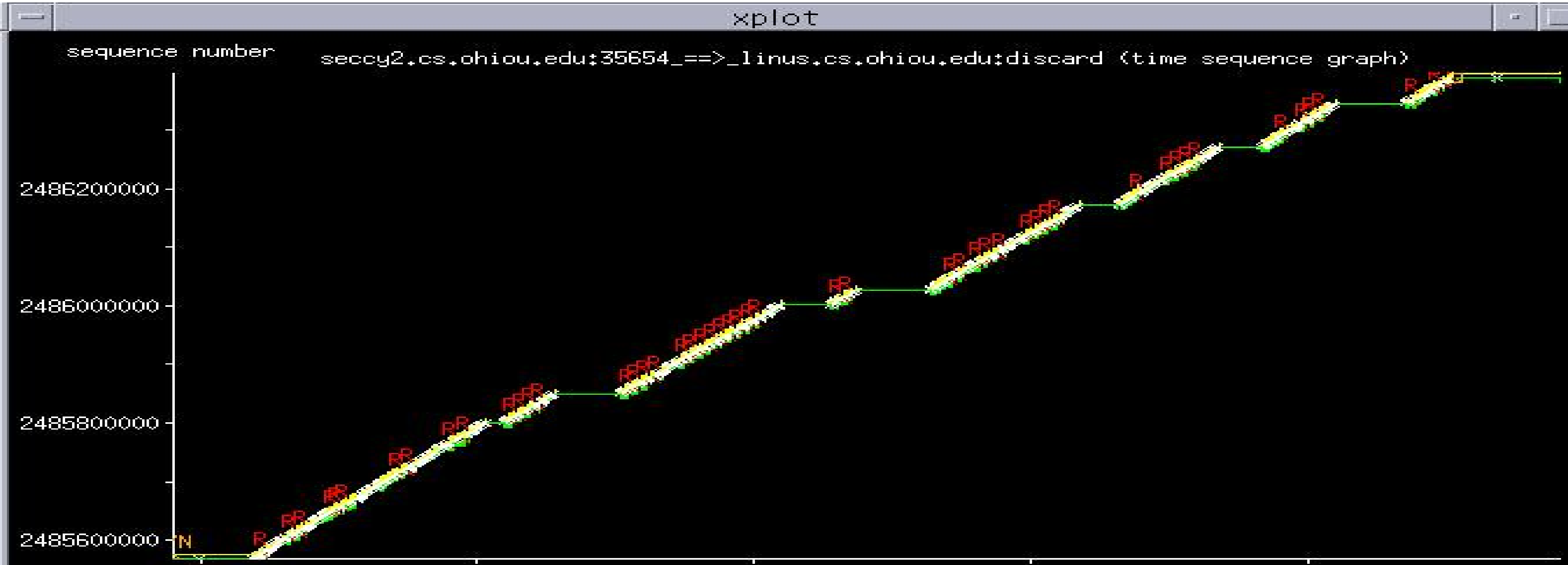
tcpdump / tcptrace

- tcpdump: dump all TCP header information for a specified source/destination
  - ftp://ftp.ee.lbl.gov/
- tcptrace: format tcpdump output for analysis using xplot
  - http://www.tcptrace.org/
  - NLANR TCP Testrig : Nice wrapper for tcpdump and tcptrace tools
    - http://www.ncne.nlanr.net/TCP/testrig/
- Sample use:

```
tcpdump -s 100 -w /tmp/tcpdump.out host hostname
tcptrace -Sl /tmp/tcpdump.out
xplot /tmp/a2b_tsg.xpl
```

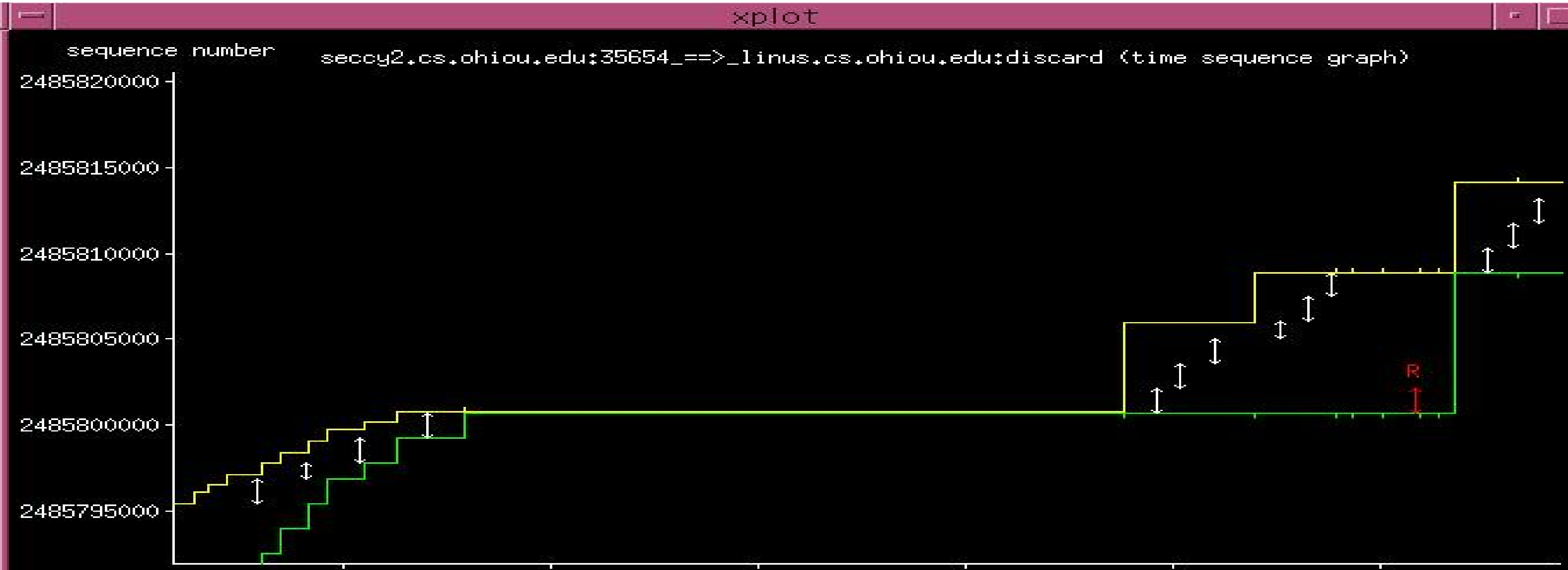
tcptrace and xplot

- X axis is time
- Y axis is sequence number
- the slope of this curve gives the throughput over time.
- xplot tool make it easy to zoom in



Zoomed In View

- **Green Line**: ACK values received from the receiver
- **Yellow Line** tracks the receive window advertised from the receiver
- **Green Ticks** track the duplicate ACKs received.
- **Yellow Ticks** track the window advertisements that were the same as the last advertisement.
- **White Arrows** represent segments sent.
- **Red Arrows (R)** represent retransmitted segments



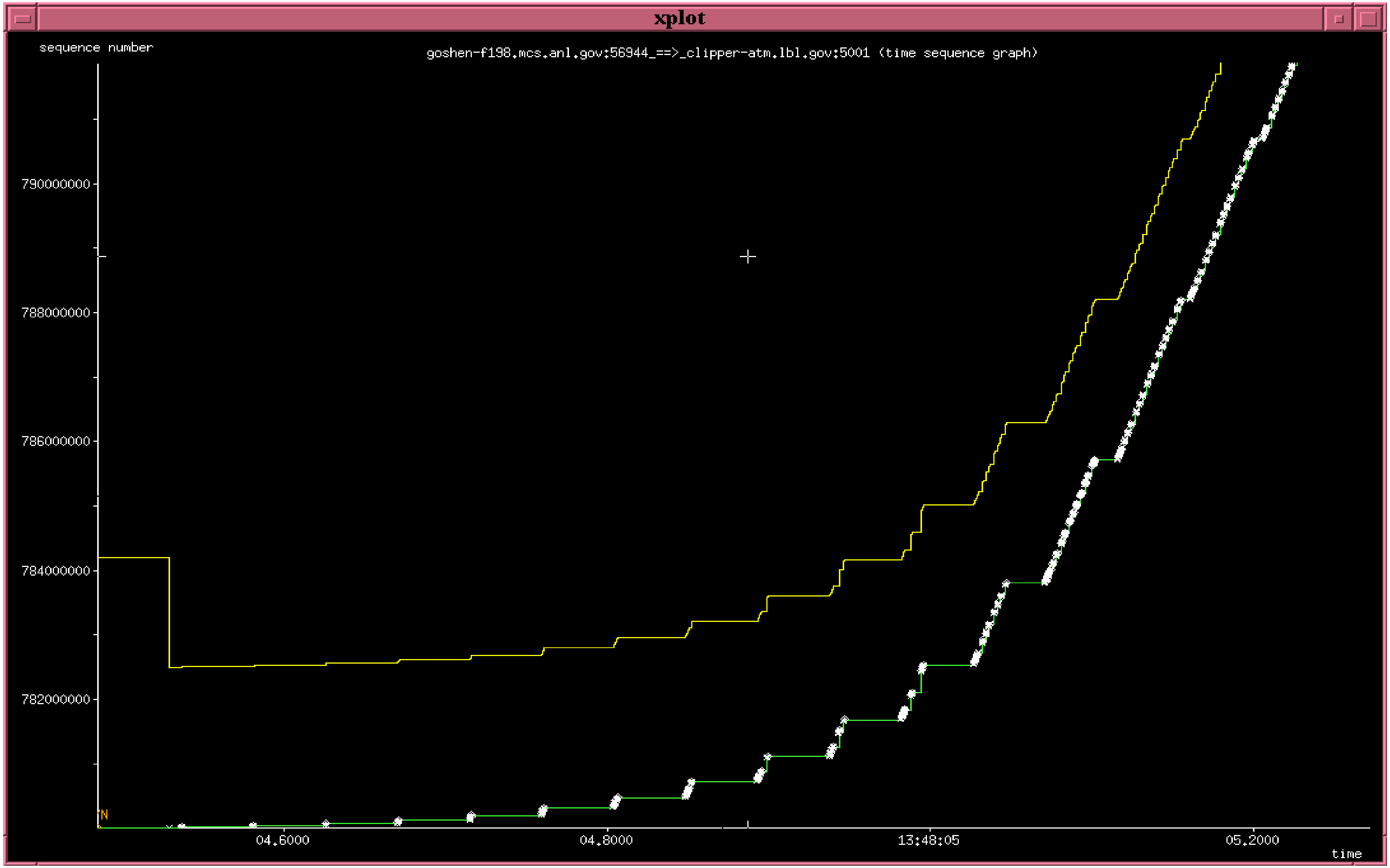
Other Tools

- NLANR Tools Repository:
  - http://www.ncne.nlanr.net/software/tools/
- SLAC Network MonitoringTools List:
  - http://www.slac.stanford.edu/xorg/nmtf/nmtf-tools.html

Other TCP Issues

- Things to be aware of:
  - TCP slow-start
    - On a path with a 50 ms RTT, it takes 12 RTT's to ramp up to full window size, so need to send about 10 MB of data before the TCP congestion window will fully open up.
  - host issues
    - Memory copy speed
    - I/O Bus speed
    - Disk speed

TCP Slow Start



Duplex Mismatch Issues

- A common source of trouble with Ethernet networks is that the host is set to full duplex, but the Ethernet switch is set to half-duplex, or visa versa.
- Most newer hardware will auto-negotiate this, but with some older hardware, auto-negotiation sometimes fails
  - result is a working but very slow network (typically only 1-2 Mbps)
  - best for both to be in full duplex if possible, but some older 100BT equipment only supports half-duplex
- NDT is a good tool for finding duplex issues:
  - http://e2epi.internet2.edu/ndt/

Jumbo Frames

- Standard Ethernet packet is 1500 bytes (aka: MTU)
- Some gigabit Ethernet hardware supports “jumbo frames” (jumbo packet) up to 9 KBytes
  - This helps performance by reducing the number of host interrupts
  - Some jumbo frame implementations do not interoperate
  - Most routers allow at most 4K MTUs
- First Ethernet was 3 Mbps (1972)
- First 10 Gbit/sec Ethernet hardware: 2001
  - Ethernet speeds have increased 3000x since the 1500 byte frame was defined
  - Computers now have to work 3000x harder to keep the network full

Linux Autotuning

- Sender-side TCP buffer autotuning introduced in Linux 2.4
  - TCP send buffer starts at 64 KB
  - As the data transfer takes place, the buffer size is continuously re-adjusted up max autotune size (default = 128K)
- Need to increase defaults: (in /etc/sysctl.conf)

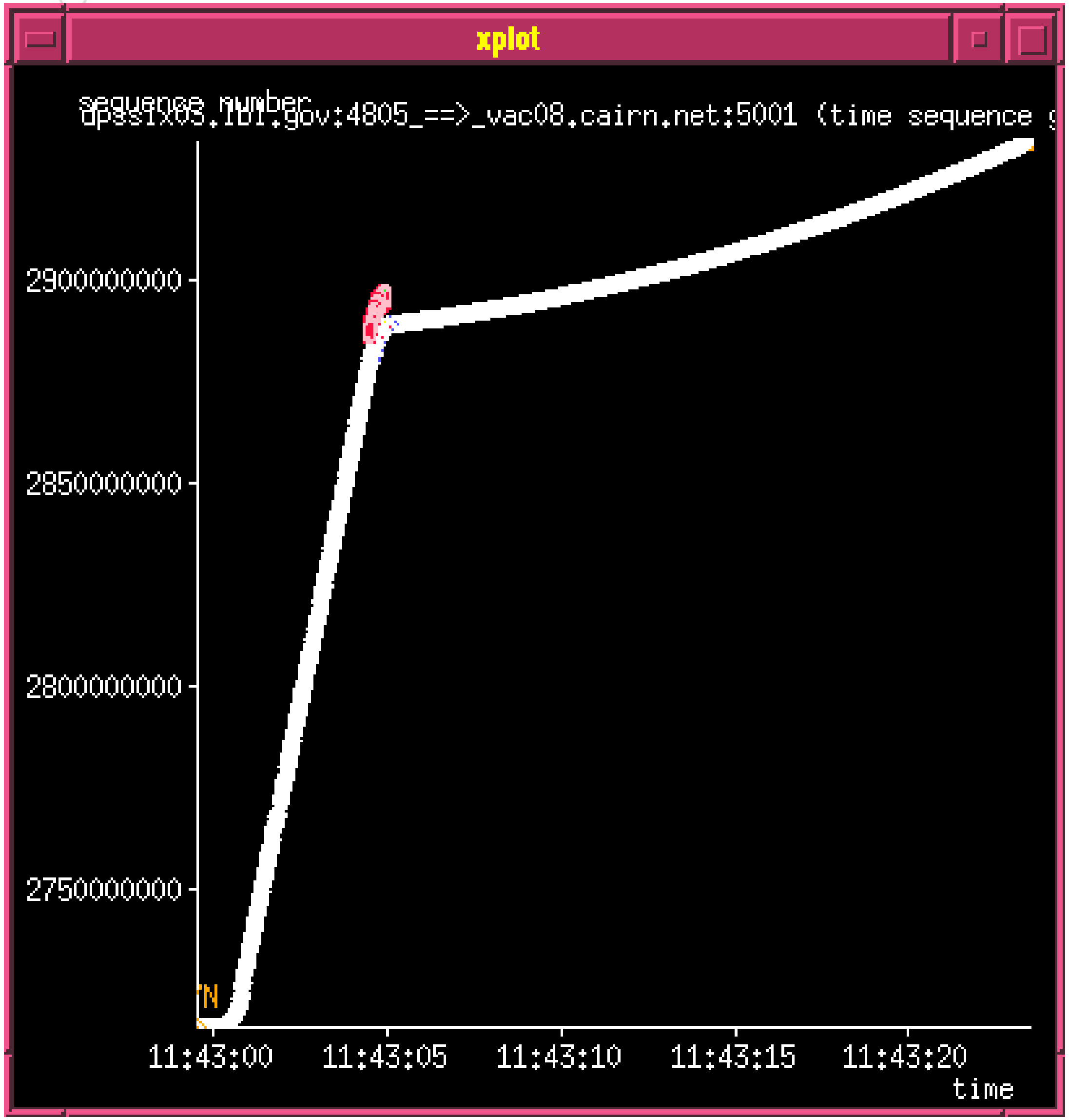
```
# increase TCP max buffer size
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
# increase Linux autotuning TCP buffer limits
# min, default, and max number of bytes to use
net.ipv4.tcp_rmem = 4096 87380 16777216
net.ipv4.tcp_wmem = 4096 65536 16777216
```

- Receive buffers need to be bigger than largest send buffer used
  - Use setsockopt() call

Linux 2.4 Issues

- *ssthresh* caching
  - ssthresh (Slow Start Threshold): size of CWND to use when switching from exponential increase to linear increase
  - The value for ssthresh for a given path is cached in the routing table.
  - If there is a retransmission on a connection to a given host, then all connections to that host for the next 10 minutes will use a reduced ssthresh.
  - Or, if the previous connect to that host is particularly good, then you might stay in slow start longer, so it depends on the path
  - The only way to disable this behavior is to do the following before all new connections (you must be root):
    - sysctl -w net.ipv4.route.flush=1
  - The web100 kernel patch adds a mechanism to permanently disable this behavior:
    - sysctl -w net.ipv4.web100_no_metrics_save = 1

ssthresh caching



- The value of CWND where this loss happened will get cached

Linux 2.4 Issues (cont.)

- SACK implementation problem
  - For very large BDP paths where the TCP window is > 20 MB, you are likely to hit the Linux SACK implementation problem.
  - If Linux has too many packets in flight when it gets a SACK event, it takes too long to located the SACKed packet,
    - you get a TCP timeout and CWND goes back to 1 packet.
  - Restricting the TCP buffer size to about 12 MB seems to avoid this problem, but limits your throughput.
  - Another solution is to disable SACK.

```
sysctl -w net.ipv4.tcp_sack = 0
```

  - This is still a problem in 2.6, but they are working on a solution
- Transmit queue overflow
  - If the interface transmit queue overflows, the Linux TCP stack treats this as a retransmission.
  - Increasing txqueuelen can help:

```
ifconfig eth0 txqueuelen 1000
```

Recent/Current TCP Work

TCP Response Function

- Well known fact that TCP does not scale to high-speed networks
- Average TCP congestion window = $1.2/\sqrt{p}$ segments
  - p = packet loss rate
- What this means:
  - For a TCP connection with 1500-byte packets and a 100 ms round-trip time, filling a 10 Gbps pipe would require a congestion window of 83,333 packets, and a packet drop rate of at most one drop every 5,000,000,000 packets.
  - requires at **most** one packet loss every 6000s, or 1h:40m to keep the pipe full

Proposed TCP Modifications

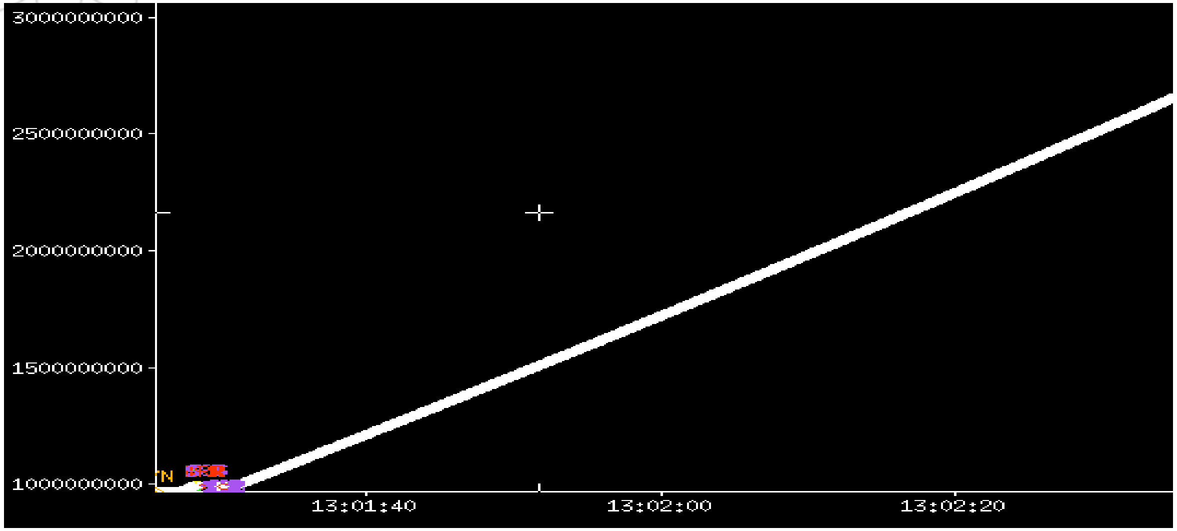
- High Speed TCP: Sally Floyd
  - http://www.icir.org/floyd/hstcp.html
- BIC/CUBIC:
  - http://www.csc.ncsu.edu/faculty/rhee/export/bitcp/
- LTCP (Layered TCP)
  - http://students.cs.tamu.edu/sumitha/research.html
- HTCP: (Hamilton TCP)
  - http://www.hamilton.ie/net//htcp/
- Scalable TCP
  - http://www-lce.eng.cam.ac.uk/~ctk21/scalable/

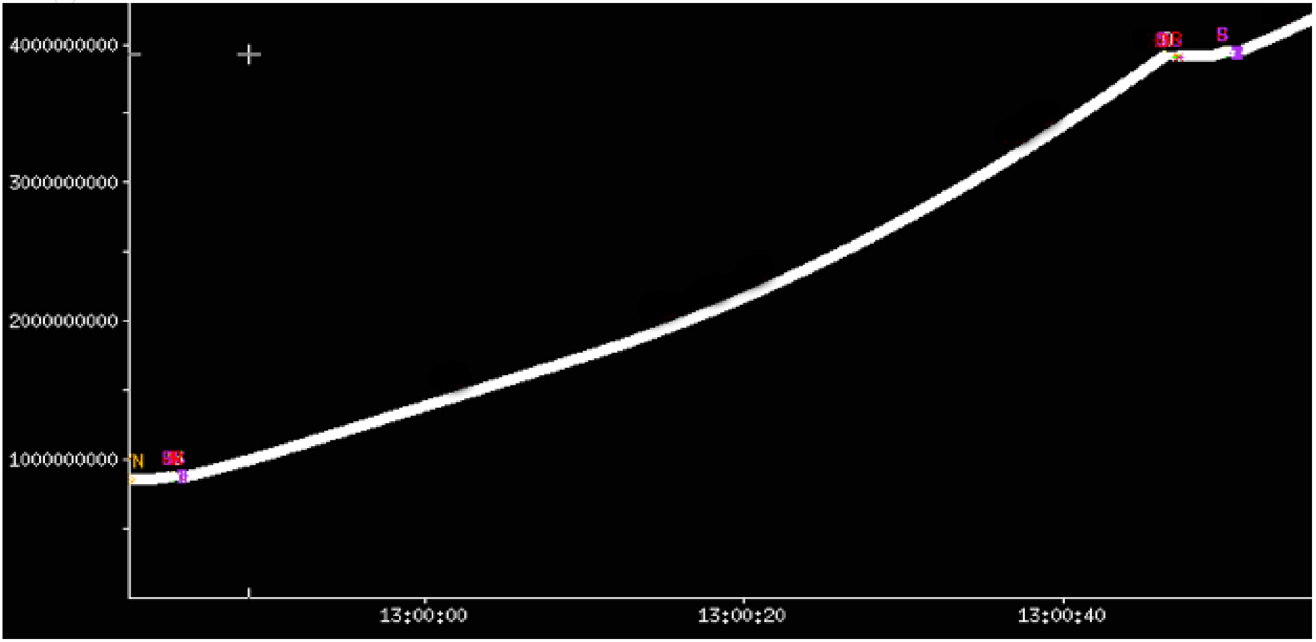
Proposed TCP Modifications (cont.)

- XCP:
  - XCP rapidly converges on the optimal congestion window using a completely new router paradigm.
    - This makes it very difficult to deploy and test
  - http://www.ana.lcs.mit.edu/dina/XCP/
- FAST TCP:
  - http://netlab.caltech.edu/FAST/

- Each if these alternatives give roughly similar throughput
  - Vary mainly in “stability” and “friendliness” with other protocols
- Each of these require sender-side only modifications to standard TCP

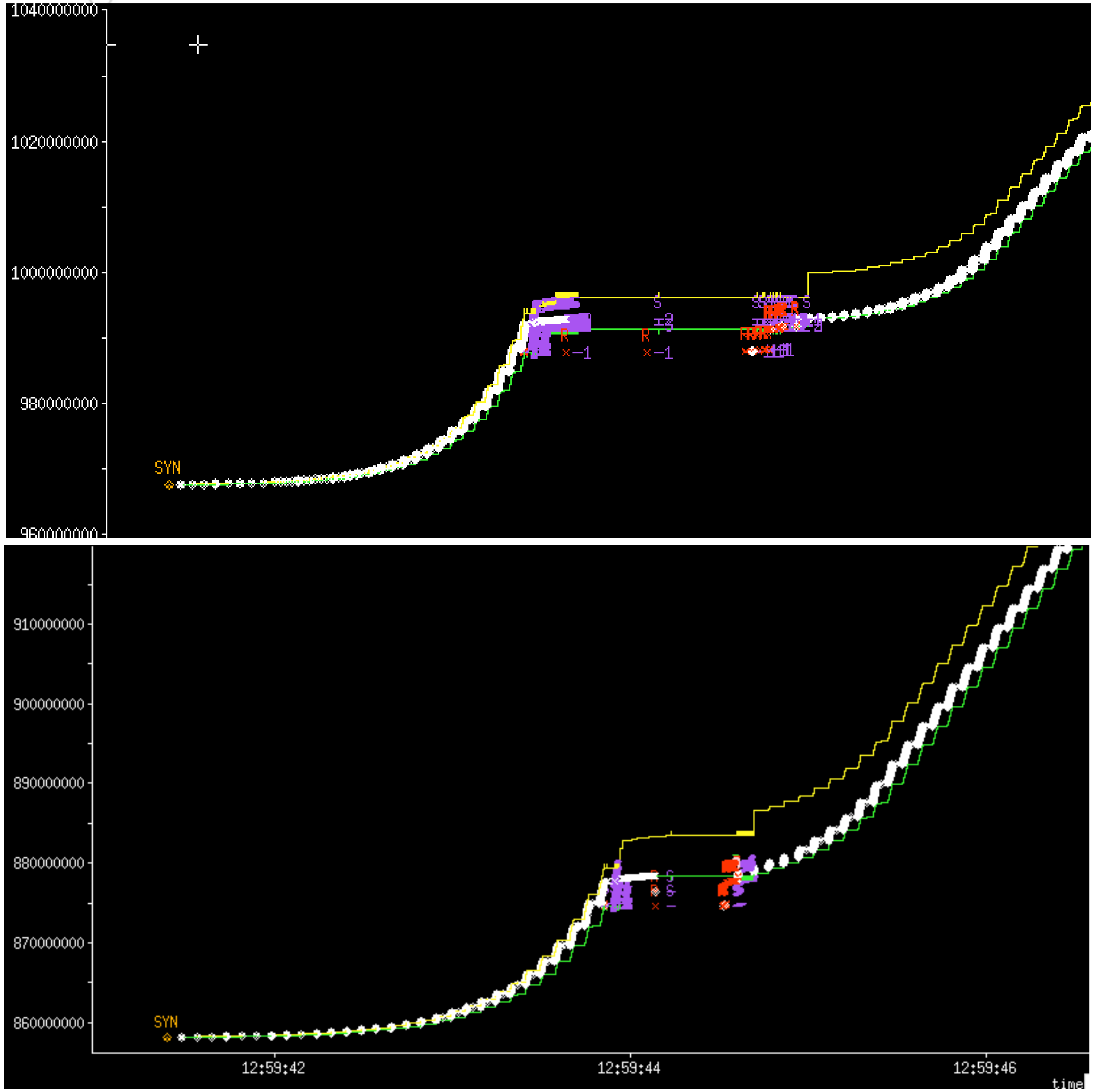
TCP: Reno vs. BIC





- TCP-Reno (Linux 2.4)
- BIC-TCP (Linux 2.6)

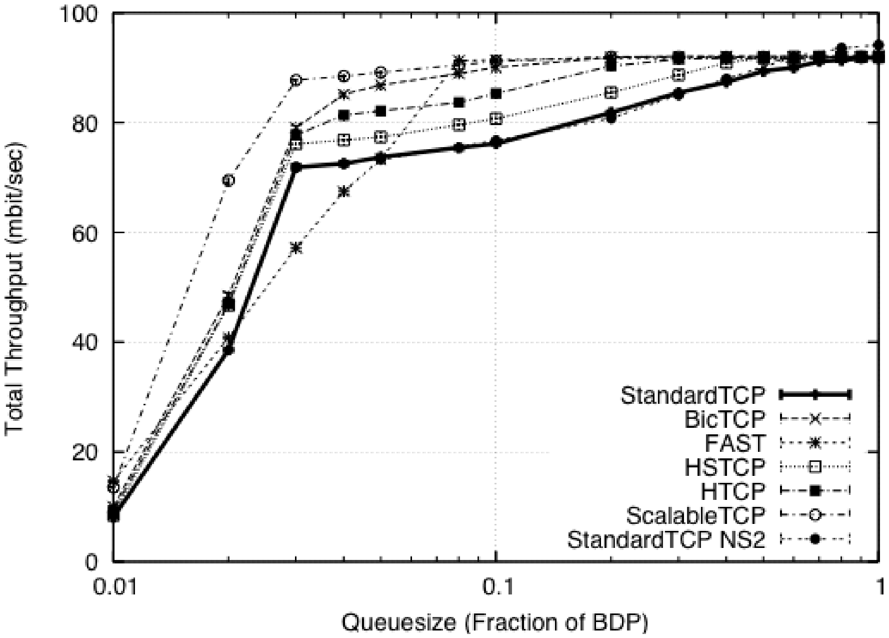
TCP: Reno vs. BIC



- **BIC-TCP recovers from loss more aggressively than TCP-Reno**

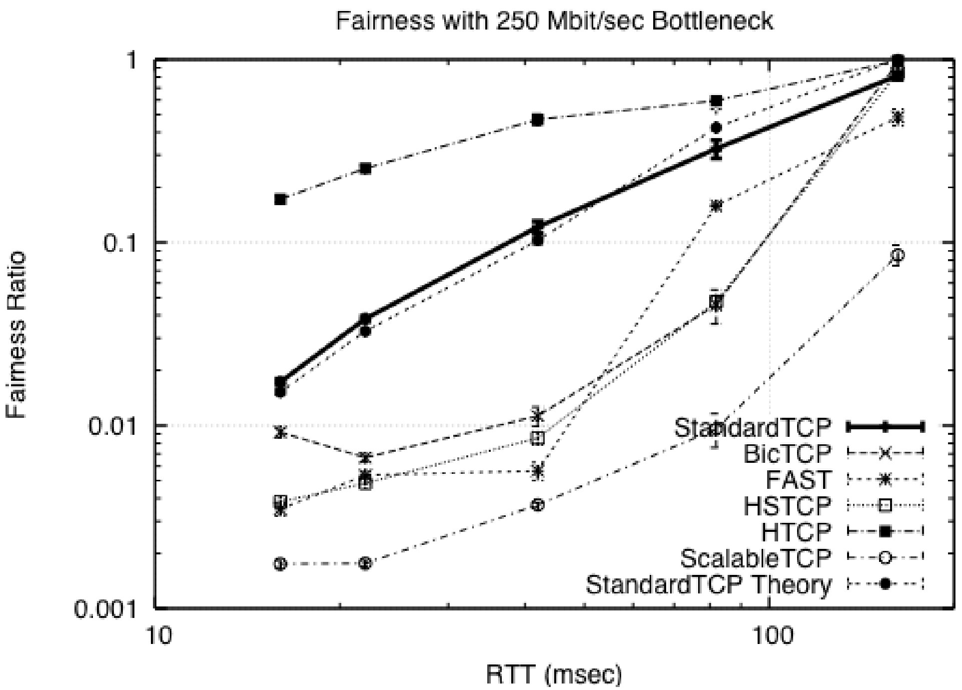
Sample Results

From Doug Leith, Hamilton Institute, http://www.hamilton.ie/net/eval/



Link Utilization

Fairness Between Flows



New Linux 2.6 changes

- Added receive buffer autotuning: adjust receive window based on RTT
  - `sysctl net.ipv4.tcp_moderate_rcvbuf`
  - Still need to increase max value: `net.ipv4.tcp_rmem`
- Starting in Linux 2.6.7 (and back-ported to 2.4.27), BIC TCP is part of the kernel, *and enabled by default*.
- Bug found that caused performance problems under some circumstances, fixed in 2.6.11.
- Added ability to disable ssthresh caching (like web100)

```
net.ipv4.tcp_no_metrics_save = 1
```

Linux 2.6 Issues

- "tcp segmentation offload" issue:
  - Linux 2.6 (< 2.6.11) has bug with certain Gigabit and 10 Gig ethernet drivers and NICs that support "tcp segmentation offload",
    - These include Intel e1000 and ixgb drivers, Broadcom tg3, and the s2io 10 GigE drivers.
    - To fix this problem, use *ethtool* to disable segmentation offload:

```
ethtool -K eth0 tso off
```

  - Bug fixed in Linux 2.6.12

Networks For Non-Networkers

Linux 2.6.12-rc3 Results

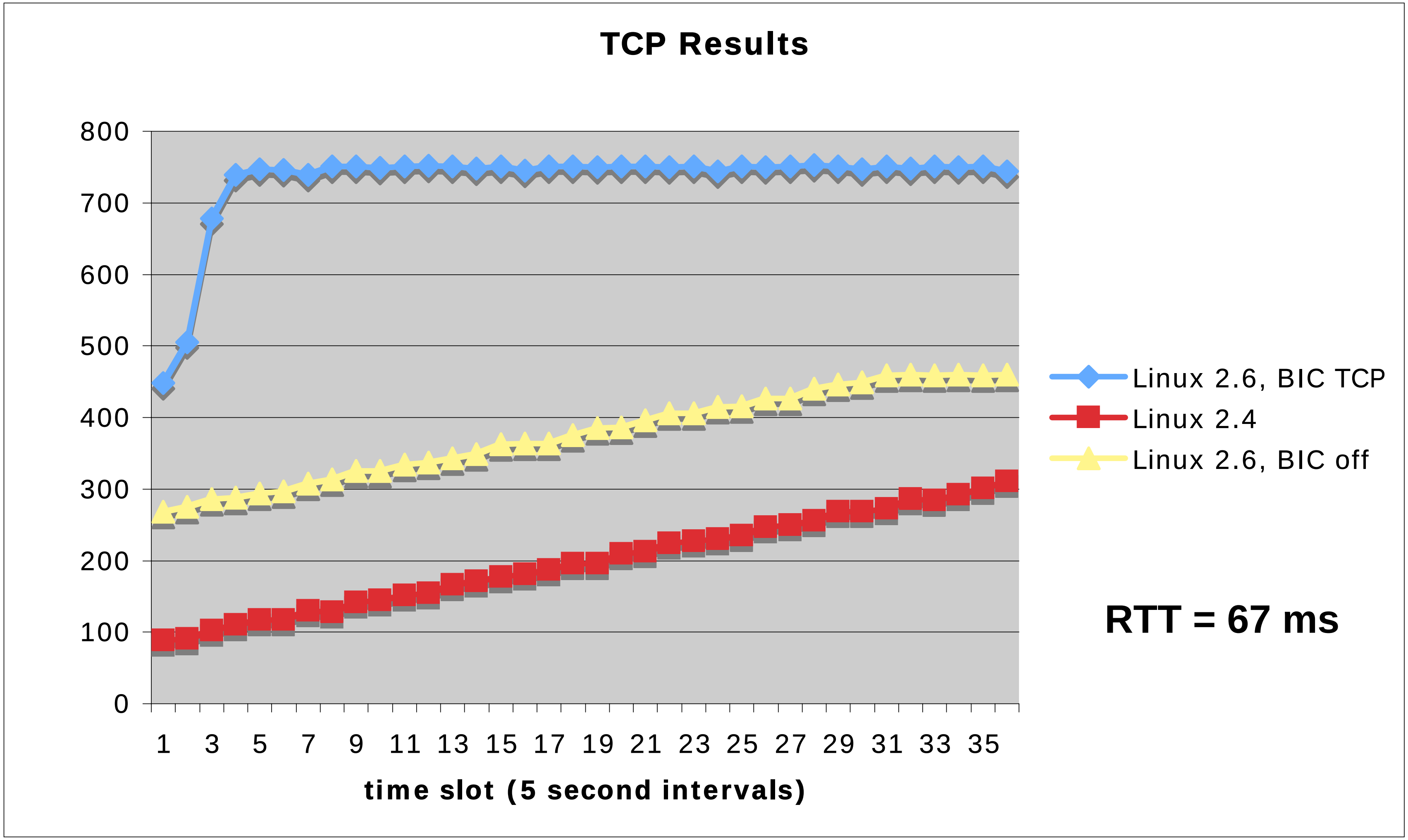
Path	Linux 2.4	Linux 2.6 with BIC	Linux 2.6, no BIC
LBL to ORNL RTT = 67 ms	300 Mbps	700 Mbps	500 Mbps
LBL to PSC RTT = 83 ms	300 Mbps	830 Mbps	625 Mbps
LBL to IE RTT = 153 ms	70 Mbps	560 Mbps	140 Mbps

Results = Peak Speed during 3 minute test

Note: BIC is ON by default in Linux 2.6

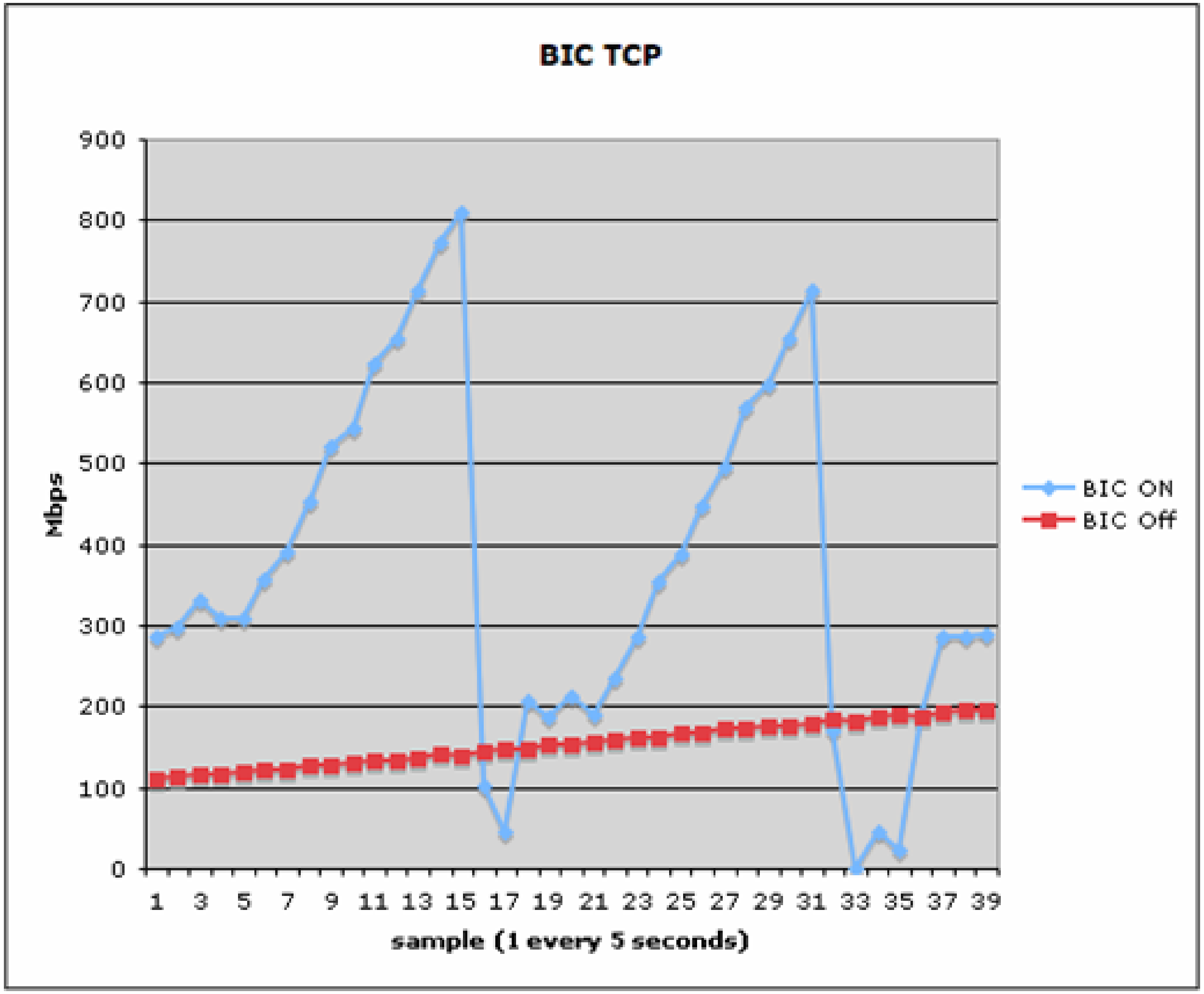
Sending host = 2.8 GHz Intel Xeon with Intel e1000 NIC

Linux 2.6.12-rc3 Results



Remaining Linux BIC Issues

- But: on some paths BIC still seems to have problems…



RTT = 83 ms

Application Performance Issues

Networks For Non-Networkers

Techniques to Achieve High Throughput over a WAN

- Consider using multiple TCP sockets for the data stream
- Use a separate thread for each socket
- Keep the data pipeline full
  - use asynchronous I/O
    - overlap I/O and computation
  - read and write large amounts of data (> 1MB) at a time whenever possible
  - pre-fetch data whenever possible
- Avoid unnecessary data copies
  - manipulate pointers to data blocks instead

Use Asynchronous I/O

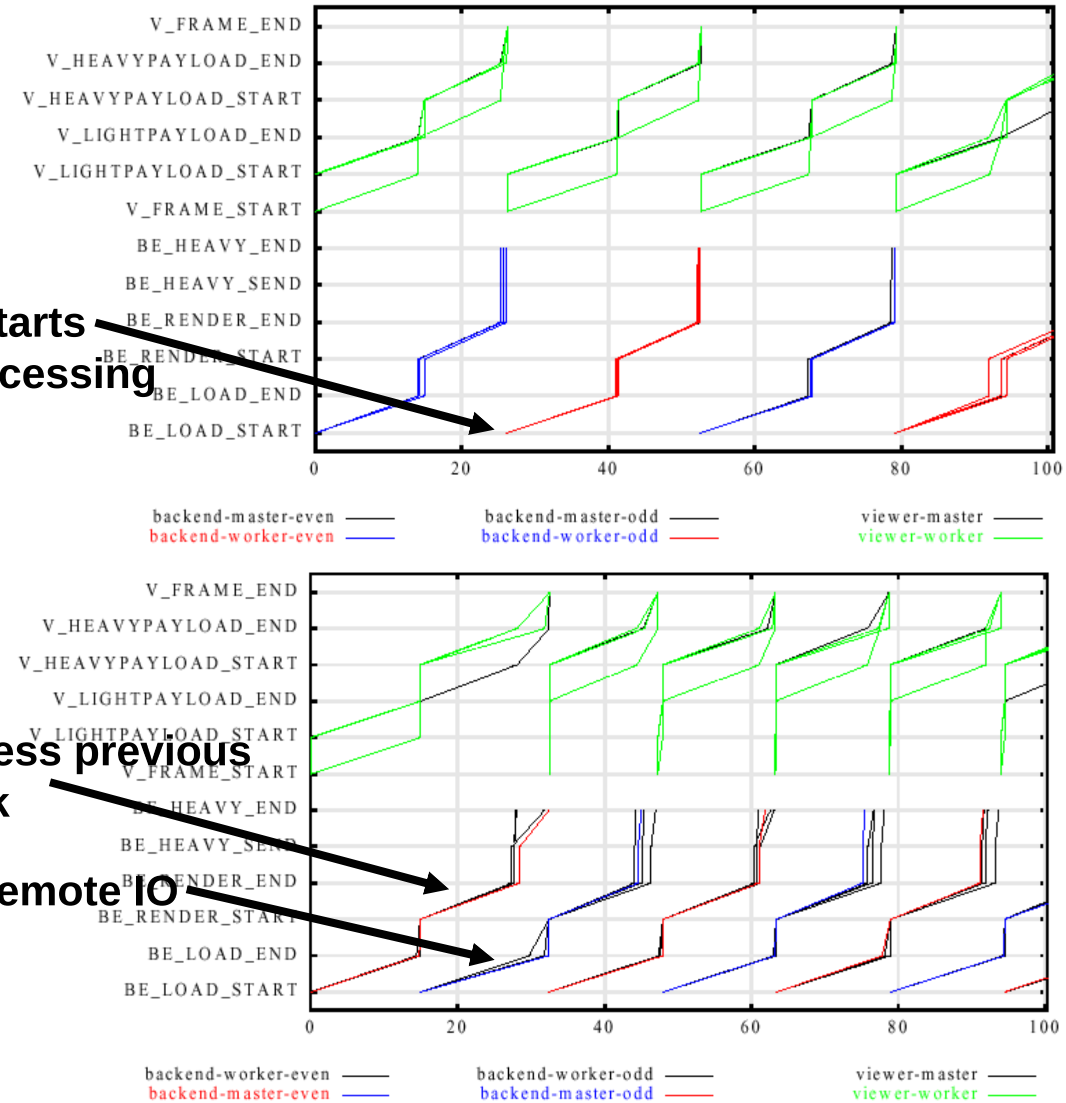
- I/O followed by processing
- overlapped I/O and processing

almost a 2:1 speedup

Next IO starts when processing ends

process previous block

remote IO



Throughput vs. Latency

- Most of the techniques we have discussed are designed to improve throughput
- Some of these might increase latency
  - with large TCP buffers, OS will buffer more data before sending it
- Goal of a distributed application programmer:
  - hide latency
- Some techniques to help decrease latency:
  - use separate control and data sockets
  - use TCP_NODELAY option on control socket
    - combine control messages together into 1 larger message whenever possible on TCP_NODELAY sockets

scp Issues

- Don’t use scp to copy large files!
  - scp has its own internal buffering/windowing that prevents it from ever being able to fill LFNs!
- Explanation of problem and openssh patch solution from PSC
  - http://www.psc.edu/networking/projects/hpn-ssh/

Conclusions

- The wizard gap is starting to close (slowly)
  - If max TCP buffers are increased
- Tuning TCP is not easy!
  - no single solution fits all situations
    - need to be careful TCP buffers are not too big or too small
    - sometimes parallel streams help throughput, sometimes they hurt
  - Linux 2.6 helps a lot
- Design your network application to be as flexible as possible
  - make it easy for clients/users to set the TCP buffer sizes
  - make it possible to turn on/off parallel socket transfers
    - probably off by default
- Design your application for the future
  - even if your current WAN connection is only 45 Mbps (or less), some day it will be much higher, and these issues will become even more important

For More Information

http://dsd.lbl.gov/TCP-tuning/

- links to all network tools mentioned here
- sample TCP buffer tuning code, etc.

BLTierney@LBL.GOV