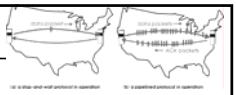
Internet Programming & Protocols Lecture 9

TCP bandwidth-delay and Long Fat Pipes

Performance tools

Bandwidth-delay product



- Maximum bandwidth of TCP connection is

  min(receiver's RCVBUF, sender's SNDBUF) / RTT
- Given RTT and bottle-neck link speed, calculate the TCP buffer space needed to fill the pipe by multiplying bandwidth x delay
  - RTT = 100 ms, bandwidth = 45 mbs → $45x10^6$ x 0.1 → 4.5 Mb buffers
  - RTT = 100 ms, bandwidth = 100 mbs → need 1.2 MBytes buffers
  - RTT = 100 ms, bandwidth = 1 Gbs → need 12 Mbytes buffers
  - Or given your OS has default buffer size of 64 Kbytes and RTT 100 ms, then your TCP connection will go no faster than 64K/0.1 = 640 KBs or 5 mbs ☹
- For your own client/server you could have a setsockopt() to set buffer sizes, but if your client (e.g., browser) is talking to a server that you have no control over, then you are bounded by server's buffer sizes
- Each socket has its own SNDBUF/RCVBUF, default size has grown over the years (16KB, 32KB, 64KB) – check your OS! OK for LAN
- Too small a buffer size limits throughput, too big consumes system resources and possibly adds to congestion and delay

Long fat pipes

- Evolution to high speed links and NIC (gigabit and higher) over long delay paths has created some problems for TCP
  - Window size in TCP header is only 16 bits (65,535) but
    - RTT = 100 ms, bandwidth = 100 mbs → need 1.2 MBytes buffers
  - 32-bit sequence numbers can wrap (ambiguous packet numbers?)

Network	link speed	time to wrap
ARPANET	56kbs	3.6 days
DS1	1.5mbs	3 hours
DS3	45 mbs	380 s
100T	100 mbs	170 s
GigE	1 gbs	17 s
OC192	10 gbs	1.7s

  - Huge amount of data "in flight", loss recovery problematic
    - Need more info about missing packets (SACK … later)
- RFC 1323 -- extensions for high performance ('92) ★

Window scale TCP option

- For windows bigger than 64KB, TCP option in SYN packet can set a "scale" factor for window field
- Both hosts must support window scaling (SYN-ACK carries window scale info for other end)
- 3-byte TCP option (type=3, lth=3, scale)
  - Scale value (left shift) from 0 to 14, so up to 1 GigaByte window
- TCP "remembers" scale factor in socket data structure

```
160.91.212.75.34243 > 160.36.58.221.5001: S 2370639492:2370 639492(0) win 5840
    <mss 1460,sackOK,timestamp 273760095 7 0,nop,wscale 8>
0x0000: 4500 003c 6f39 4000 4006 7bda a05b d44b E..<o9@.@.{..[.K
0x0010: a024 3add 85c3 1389 8d4d 1684 0000 0000 .$:......M......
0x0020: a002 16d0 877e 0000 0204 05b4 0402 080a .....~..........
0x0030: a32c 79bd 0000 0000 0103 0308           .,y.........
 160.36.58.221.5001 > 160.91.212.75.34243: S 3684589243:3684 589243(0) ack
    2370639493 win 5792 <mss 1460,sackOK,timestamp 237253051 273760095
    7,nop,wscale 7>
0x0000: 4500 003c 0000 4000 3a06 f113 a024 3add E..<..@.:....$:.
0x0010: a05b d44b 1389 85c3 db9e 5ebb 8d4d 1685 .[.K......^..M..
0x0020: a012 16a0 0d65 0000 0204 05b4 0402 080a .....e..........
0x0030: 0e24 31bb a32c 79bd 0103 0307           .$1..,y.....
```

Tcptrace note: you have to see SYN packets to know how to interpret window field

TCP timestamp option

- TCP option to include 32-bit time stamp in every packet (+12 bytes)
- Return ACK carries original time stamp (plus receiver's time stamp)
- Value is usually tick counter of TCP's 500 ms timer (100 ms newer OS)
- (type=8, length=10, myval, yourval)
- Timestamp can be "prepended" to sequence number for PAWS (prevention against wrapped sequence number 2^{32} → 2^{64}) and help RTT estimates -- mostly a "reliability" extension

```
160.91.212.75.34243 > 160.36.58.221.5001: S 2370639492:2370 639492(0) win 5840
    <mss 1460,sackOK, timestamp 2737600957 0, nop,wscale 8>
0x0000: 4500 003c 6f39 4000 4006 7bda a05b d44b E..<o9@.@.{..[.K
0x0010: a024 3add 85c3 1389 8d4d 1684 0000 0000 .$:......M......
0x0020: a002 16d0 877e 0000 0204 05b4 0402 080a .....~..........
0x0030: a32c 79bd 0000 0000 0103 0308           .,y.........
 160.36.58.221.5001 > 160.91.212.75.34243: S 3684589243:3684 589243(0) ack
    2370639493 win 5792 <mss 1460,sackOK, timestamp 237253051 2737600957,
    nop,wscale 7>
0x0000: 4500 003c 0000 4000 3a06 f113 a024 3add E..<..@.:....$:.
0x0010: a05b d44b 1389 85c3 db9e 5ebb 8d4d 1685 .[.K......^..M..
0x0020: a012 16a0 0d65 0000 0204 05b4 0402 080a .....e..........
0x0030: 0e24 31bb a32c 79bd 0103 0307           .$1..,y.....
```

OS tuning for high performance TCP

- Your OS may be configured to be network-challenged
  - Doesn't support RFC 1323 extensions
  - Limits max size of SNDBUF and RCVBUF
- System manager may be able to fix these things
  - Incantations vary by OS and major release
  - Configuration options to
    - Enable RFC 1323
    - Set default SNDBUF/RCVBUF (don't mess with this probably)
    - Set max limits for SNDBUF/RCVBUF setsockopt()
    - Enable path MTU discovery
    - Tweak MTU (careful)
  - See OS tuning web sites
- You may have no control of the target host… sigh
  - Though with tcpdump you can observe its window size and RFC 1323 options
  - Some applications may be "network-aware" and have tuning options

Option usage	
MSS	99%
Window scale	15%
SACK	79%
Timestamp	13%

OS tuning incantations

- FreeBSD max buffer size sysctl -w kern.maxsockbuf=524288
- Linux
  - **echo 1 > /proc/sys/net/ipv4/tcp_timestamps**
  - **echo 1 > /proc/sys/net/ipv4/tcp_window_scaling**
  - **echo 1 > /proc/sys/net/ipv4/tcp_sack**
  - **echo 8388608 > /proc/sys/net/core/wmem_max**
  - **echo 8388608 > /proc/sys/net/core/rmem_max**
  - **echo "4096 87380 4194304" > /proc/sys/net/ipv4/tcp_rmem**
  - **echo "4096 65536 4194304" > /proc/sys/net/ipv4/tcp_wmem**
- Windows XP registry
  HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
  GlobalMaxTcpWindowSize="256960"
  Tcp1323Opts="1"
- See PSC tuning table

Concept Collection

- **ACK/NAK cumulative ACK**
- **Bandwidth-delay product**
- **Best effort**
- **Bit error rate**
- **Checksums**
- **Client/server/concurrent/iterative**
- **CIDR**
- **CSMA/CD**
- **Datagram vs reliable stream**
- **Exponential backoff**
- **Flow control**
- **fragmentation**
- **Layers/encapsulation**
- **Maximum segment lifetime(MSL)**
- **MTU MSS/MTU discovery**
- **Network mask**
- **Packet switching vs circuit-based promiscuous**
- **Routing**
- **RTT**
- **Self-clocking**
- **Sliding window**
- **Subnets/supernets**
- **Switch vs hub**
- **TTL**

Measuring network performance

- If you are a network administrator, you're probably interested in overall utilization, traffic patterns, and trends, e.g. for capacity planning
  - Visual tools, alarms etc. (commercial network analyzers/managers)
- If you are building network applications, you may want to observe your application's traffic and competing traffic.
- If you are a network protocol designer, you may want to see the contents of every packet in your protocol
- If you are experiencing "poor network" performance you may want tools to monitor and probe the network paths and monitor the effects of your TCP tuning
- Network metrics include utilization, bandwidth, latency, jitter, losses
- Measurements tasks
  - Data collection
  - Analysis
  - Presentation
  - interpretation

Network measurement tools

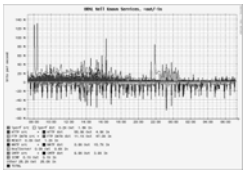
- Monitoring tools (passive)
  - Cisco netflows
  - tcpstat
  - tcpdump/tcptrace/xplot
- Benchmarking tools (active)
  - ttcp
  - iperf
  - netperf
  - DBS
- These are freely available on web

Cisco netflow

- Router can collect statistics on each flow
  - src, src port, dst, dst port, proto, packets, bytes, duration, int. in, int. out
  - Router spits these records out via UDP to a collector host
  - Tools/scripts for summarizing and graphing
- NOC tool and some use for intrusion detection
  - See Internet2 NOC data fastest flows, fattest flows, etc.

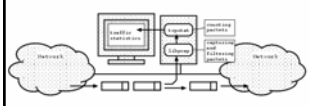
proto/port usage internet2

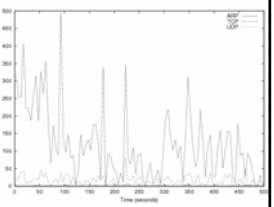
TCP	**87%**
UDP	**13%**
iperf	**21%**
http	**13%**
nntp	**11%**
ftp	**3%**
shoutcast	**3%**



tcpstat

- Passive tool based on libpcap (like tcpdump)
  - Can collect direct from NIC (root acces) or from a tcpdump file
- Periodically reports flow summaries, pkts/proto/sec
- Example
  - tcpstat –r rawdata.dmp –o "%r %A %T %U %l %b\n" > tom.log
  - Ascii records: timestamp # ARP #TCP #UDP netload bits/sec
  - Use gnuplot to parse and plot





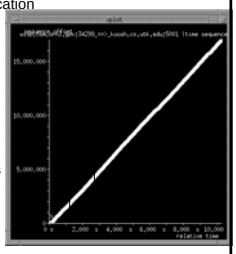
Lots of graphical tools to display current network traffic real-time.

tcptrace and xplot

- Statistics and graphs on individual TCP flow from tcpdump file
- Graphs can be very handy in visualizing TCP/path problems
  - View with xplot
  - Bandwidth vs time (instantaneous and average)
  - RTT vs time
  - Sequence/ACK vs time
    - advertised window
    - loss, timeout
    - SACK
    - retransmission
  - Window vs time
- View differs whether tcpdump was running at sender or receiver side
  - Best to do tcpdump on sender side, most of the TCP control happens at sender side
- ~dunigan/ipp05/bin/ try it, you'll like it. ☺

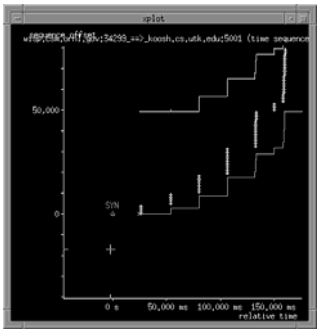
Sequence number vs time plot

- Collect a tcpdump file of your network application
- tcptrace –G –zxy –y –T –A100 file.dmp
- Generates a bunch of *.xpl files
  - a2b_rtt.xpl a2b_tsg.xpl a2b_tput.xpl
- xplot a2b_tsg.xpl
  - Plots sequence number vs time
  - Slope of line is datarate
  - Bumps in line are loss events or app. pauses
  - Can zoom in with mouse "box"
  - Zoom back out with left mouse click
  - Can get postscript plots



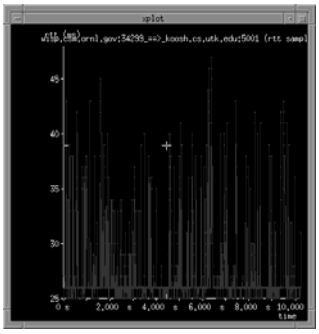
xplot of TCP startup

- Zoom in on lower left
- Green line "last ack'd"
- White dots, pkts transmitted
- Yellow: recvr window + ACK ticks
- Distance between green and yellow is available window
- Distance between steps is RTT
- Observe cumulative ACK
- Y-axis is Bytes
- TCP slow-start in action



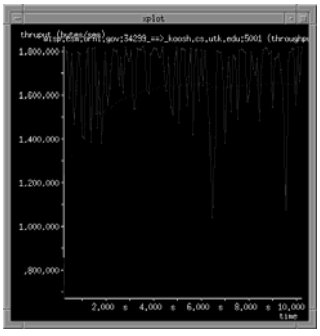
RTT xplot

- RTT (ms) vs time
- A bit of jitter from queuing delays
- Often you will see RTT grow just before a packet is lost

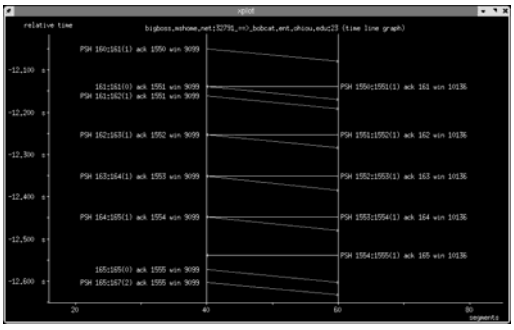


xplot of throughput

- Red is instantaneous data rate
- Blue is average data rate
  - End-point of blue is what app. reports as the data rate
- Ramp up due to slow-start



Timeline graph

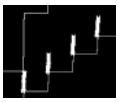


tcptrace flow summary tcptrace –rl file.dmp

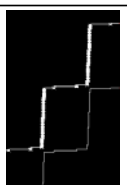
```
a->b:                              b->a:
     total packets:          12534          total packets:          7616
     ack pkts sent:          12533          ack pkts sent:          7616
     pure acks sent:             2          pure acks sent:         7614
     unique bytes sent:   18144208          unique bytes sent:         0
     actual data pkts:       12531          actual data pkts:          0
     actual data bytes:   18144208          actual data bytes:         0
     rexmt data pkts:            0          rexmt data pkts:           0
     rexmt data bytes:           0          rexmt data bytes:          0
     outoforder pkts:            0          outoforder pkts:           0
     pushed data pkts:          15          pushed data pkts:          0
     SYN/FIN pkts sent:        1/1          SYN/FIN pkts sent:       1/1
     req 1323 ws/ts:           Y/Y          req 1323 ws/ts:          Y/Y
     adv wind scale:             8          adv wind scale:            7
     req sack:                   Y          req sack:                  Y
     throughput:            801108 Bps       throughput:                0 Bps

     RTT samples:             7569          RTT samples:               2
     RTT min:                160.8 ms       RTT min:                 0.0 ms
     RTT max:                273.2 ms       RTT max:                 0.0 ms
     RTT avg:                168.0 ms       RTT avg:                 0.0 ms
     RTT stdev:               16.9 ms       RTT stdev:               0.0 ms
```

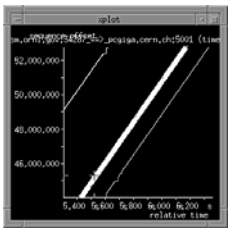

xplot diagnosis – buffer limited



SNDBUF limited ☹

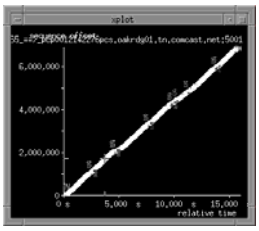


RCVBUF limited ☹

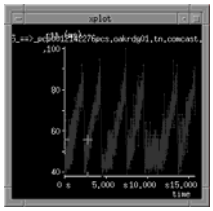


NIC or path bandwidth limited

xplot lost packets diagnosis (cable modem limiter)

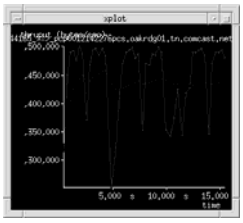


Numerous loss events as cable box drops packets to limit bandwidth



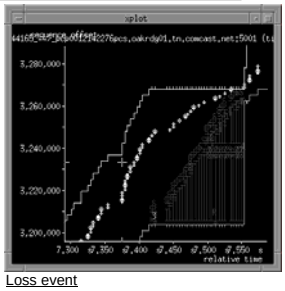
RTT grows as queue builds at modem box – input rate is higher than customer's limit rate.

Cable tests



Bandwidth clamping at 4 mbs

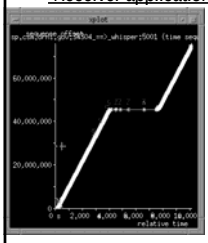
but average throughput 3.5 mbs



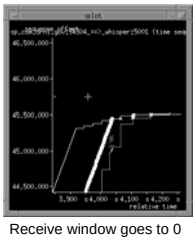
Loss event

ACK's stop advancing
purple SACK blocks
green 3 dup
red R retransmits
available window closes

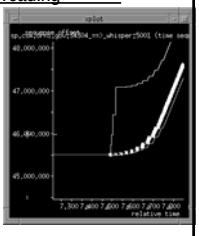
Receiver application pauses… then resumes reading



Application pauses for several seconds. Purple Z is sender doing window probes.



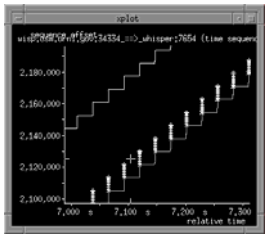
Receive window goes to 0



Application starts reading data again and window is re-opened, but TCP slow-starts, ACK clocking has been lost ☹ In the old days, blast window packets!

Sender application pauses

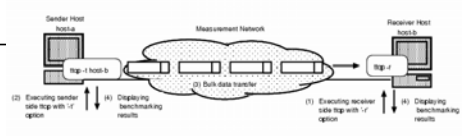
- interactive, query/response applications will not have high data rates
  - Sender or receiver goes idle
- Example, client requesting 8K records, wait for data, request next 8k
  - Better batch or pipeline requests, e.g., newer http protocols
  - xplot looks like SNDBUF limited
- Our main interest will be sustaining high throughput



Network measurement tools

- Monitoring tools (passive)
  - Cisco netflows
  - tcpstat
  - tcpdump/tcptrace/xplot
- Benchmarking tools (active) ←
  - ttcp
  - iperf
  - netperf
  - DBS
- These are freely available on web

ttcp



- Simple command line thoughput test (ttcp.c)
- Specify port, SND/RCVBUF sizes, record size, number of records
- Input/out can be from/to "sink" (-s) or stdin/stdout (compressed file)
  - Good for links with compression. ttcp –t target.host < file.tgz -- Careful you may be measuring I/O performance instead of network performance.
- Start receiver ttcp –r –s –b 2000000
- Then start sender ttcp –t –s –b 2000000 target.host.ugh
  - Defaults: port 5001, 2K record size, 8K records
- UDP option (ttcp –t –s –u), sender just blasts, receiver should report goodput (ttcp –r –s –u)
- No "duration", control with number of records (-n)
- Single-shot, restart to do another test

ttcp

```
Usage: ttcp -t [-options] host [ < in ]
       ttcp -r [-options > out]
Common options:
        -l ##   length of bufs read from or written to network (default 8192)
        -u      use UDP instead of TCP
        -p ##   port number to send to or listen at (default 5001)
        -s      -t: source a pattern to network
                -r: sink (discard) all data from network
        -A      align the start of buffers to this modulus (default 16384)
        -O      start buffers at this offset from the modulus (default 0)
        -v      verbose: print more statistics
        -d      set SO_DEBUG socket option
        -b ##   set socket buffer size (if supported)
        -f X    format for rate: k,K = kilo{bit,byte}; m,M = mega; g,G = giga
Options specific to -t:
        -n##    number of source bufs written to network (default 2048)
        -D      don't buffer TCP writes (sets TCP_NODELAY socket option)
Options specific to -r:
        -B      for -s, only output full blocks as specified by -l (for TAR)
        -T      "touch": access each byte as it's read
```

```
 ttcp -r -s
ttcp-r: buflen=8192, nbuf=2048, align=16384/0, port=5001  tcp
ttcp-r: socket
ttcp-r: accept from 160.36.58.221
ttcp-r: 16777216 bytes in 1.62 real seconds = 10125.66 KB/sec +++

ttcp -t -s wisp.csm.ornl.gov
ttcp-t: buflen=8192, nbuf=2048, align=16384/0, port=5001  tcp  ->
wisp.csm.ornl.gov
ttcp-t: socket
ttcp-t: connect
ttcp-t: 16777216 bytes in 1.27 real seconds = 12871.61 KB/sec +++
```


iperf

- Command line with persistent server
- Options for buffer size, record size, duration, UDP/TCP, parallel streams, interval status reports
- Sometimes hard to compile (C++)
- Start the server iperf –s –w 2m
- Start the client iperf –w 2m –c target.host
  - Defaults: port 5001, 10 second test, 8 KB record size
- UDP has rate option (-b) iperf –u –b 8m –c target.host
  - Good for friendly probe of available bandwidth
  - Reports losses, dups, out of order

Iperf TCP example

```
iperf -w 2m -c whisper
------------------------------------------------------------
Client connecting to whisper, TCP port 5001
TCP window size:  4.0 MByte (WARNING: requested  2.0 MByte)
------------------------------------------------------------
[  3] local 160.91.212.75 port 34347 connected with 160.36.58.221 port
5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.3 sec   113 MBytes  92.1 Mbits/sec
```

```
iperf -s -w 2m
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size:  4.0 MByte (WARNING: requested  2.0 MByte)
------------------------------------------------------------
[  6] local 160.36.58.221 port 5001 connected with 160.91.212.75 port
34347
[ ID] Interval       Transfer     Bandwidth
[  6]  0.0-10.2 sec   113 MBytes  92.3 Mbits/sec
```

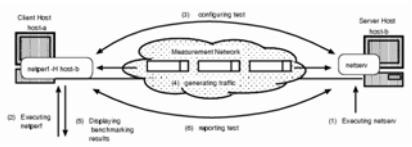

Iperf UDP example

```
iperf -s -u
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 64.0 KByte (default)
------------------------------------------------------------
[  3] local 192.168.1.4 port 5001 connected with 160.36.58.221 port 33113
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0-10.0 sec  7.50 MBytes  6.29 Mbits/sec  0.703 ms    1/ 5352 (0.019%)
[  3]  0.0-10.0 sec  1 datagrams received out-of-order
[  4] local 192.168.1.4 port 5001 connected with 160.36.58.221 port 33114
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  4]  0.0-10.0 sec  7.50 MBytes  6.29 Mbits/sec  0.427 ms    3/ 5352 (0.056%)

iperf -u -b 6m -i 1 -c catv
------------------------------------------------------------
Client connecting to catv, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 64.0 KByte (default)
------------------------------------------------------------
[  3] local 160.36.58.221 port 33114 connected with 69.252.162.198 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3] -0.0- 1.0 sec   771 KBytes   6.3 Mbits/sec
[  3]  1.0- 2.0 sec   768 KBytes   6.3 Mbits/sec
[  3]  0.0-10.0 sec   7.5 MBytes   6.3 Mbits/sec
…
[  3] Sent 5352 datagrams
```

recall our TCP cable test only got 3.4 mbs

netperf



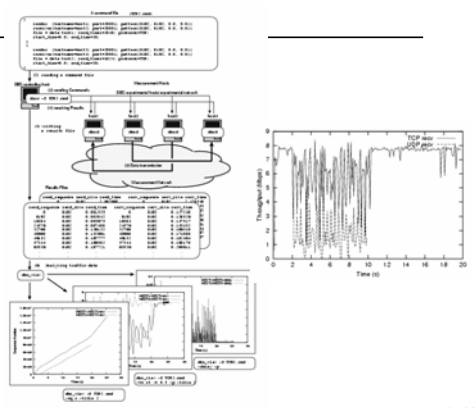
- Advantage: persistent netserver
  - Client uses a "control" TCP connection to configure server receiver
    - Set buff sizes, write size, duration, anti-compression option
  - Server spawns off receiver process (can't spawn a transmitter ☹)
  - Results from server returned over control socket
- Server: netserver –p 12865
- Client: netperf -H server.host -- -m 8192 -M 8192 -s 100000 -S 100000

DBS

Distributed Benchmark System

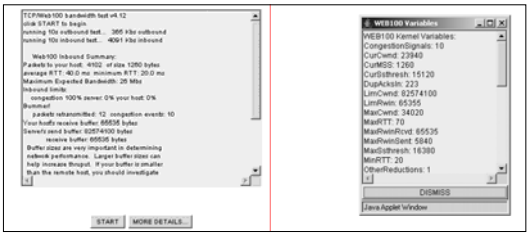
- Can evaluate multiple TCP sessions in parallel on multiple hosts
- Besides data rate, can evaluate interactions of flows
  - Fairness
  - Loss/recovery
  - Delay/jitter
- Controlled by a command file
  - Test commands and parameters for each host
  - When to start, what to do,
- Daemon processes (dbsd) running on participating hosts
- Client machine drives tests from command file via dbsc command
- Results collected to disk files
- dbs_view used to display graphs and charts of tests

DBS



Browser-based testers

- Lots of browser-based tools for evaluating your OS configuration and testing bandwidth inbound and outbound
- http://whisper.cs.utk.edu:7123/



Also down-loadable tools for "tuning" your network parameters – careful.

Selecting the right tool

- Probably use monitoring tools in conjunction with bandwidth testers
  - tcpdump/tcptrace/xplot
- iperf is most popular today
- Use multiple pairs of hosts to induce packet loss (send a blast of UDP)
- Test environments
  - standalone testbed
  - Testbed with emulator
    - Introduce loss and delay
    - NISTnet or dummynet … later
  - Test on the Internet
    - Unsure of background traffic
  - Know how the test nodes' OS TCP parameters are configured
  - Be "friendly"
- There are oodles of free network test tools!

Our tool set

- ping/traceroute
- ifconfig/netstat
- strace
- lsof
- dig
- ethereal tcpdump/tcptrace/xplot
- ttcp/iperf/netperf

Things that slow TCP down

- SNDBUF limits
- RCVBUF limits
- NIC speed or bottleneck link speed
- Packet loss
- Application “protocol”

- For more info on using tcpdump/tcptrace/xplot see NLANR page on TCP debugging
- Web100 project (instrumented Linux kernel) provides additional TCP flow monitoring

Next time ...

- TCP RTT estimation
- Tiny packets – delayed ACKs, Nagle, silly windows
- TCP timers
- TCP slow-start

assignment 4 and 5