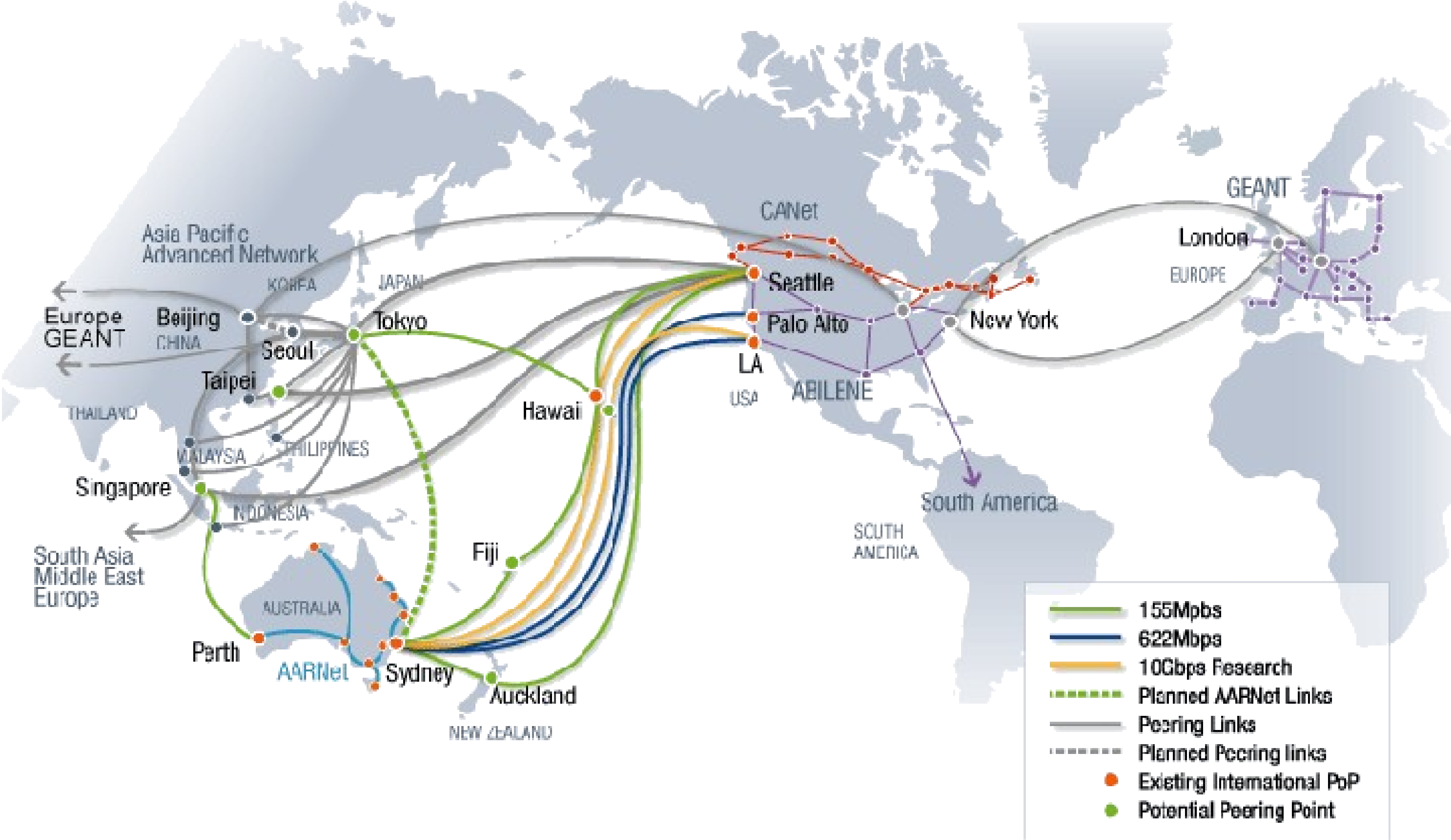
# Large file transfers using TCP

4th e-VLBI workshop, 2005-07-12
Australia Telescope National Facility, Marsfield

Glen Turner

aarnet
Australia's Academic and Research Network

# →AARNet's links to overseas research networks



# →Topics

- Tuning TCP for long fat pipes
- Exotic TCP
- Hardware choices

# →1. Tuning TCP for long fat pipes

- Two tasks
  - –Reserving buffer memory
  - –Enabling TCP features

# →Reserving buffer memory

- TCP's Window is the amount of unacknowledged data in the pipe
- The window size is a measure of throughput, since

  $$throughput = window \div round\ trip\ time$$

  and *round trip time* is reasonably constant for a connection
- We need enough buffer to feed a fully opened window, otherwise throughput will drop

# →Calculating buffer memory

- We desire

  *tcp throughput* = *link bandwidth*

  and

  *window* = *congestion window*

  Since

  *tcp throughput* = *window* ÷ *round trip time*

  We get

  ***congestion window* = *bandwidth* × *round trip time***

- This result is so important it has a name

  *bandwidth–delay product*

# →Estimating the bandwidth–delay product

- We know the smallest link in the path is 1Gbps
- Estimate round-trip time using *ping*

```
$ ping -s 9000 -M dont www.geant.net
PING newweb.dante.net (62.40.101.34) 9000(9028) bytes of data.
9008 bytes from www.dante.net (62.40.101.34): icmp_seq=0 ttl=49 time=324 ms
9008 bytes from www.dante.net (62.40.101.34): icmp_seq=1 ttl=49 time=324 ms
9008 bytes from www.dante.net (62.40.101.34): icmp_seq=2 ttl=49 time=324 ms
```

- Calculate bandwidth–delay product

$$1{,}000{,}000{,}000\text{bps} \div 8 \times 0.324\text{s} = 39\text{MiB}$$

  – This 80MB is of kernel memory: it doesn't swap
- The bandwidth–delay product is linear
  – In the above example, 800MB for 10Gbps

# →Configuring buffer in Linux

- Edit */etc/sysctl.conf*
- Increase maximum allowable socket buffers

```
net.core.wmem_max = 40500000
net.core.rmem_max = 40500000
```

- Increase TCP buffers

```
net.ipv4.tcp_wmem = 4096 65536 40500000
net.ipv4.tcp_rmem = 4096 87380 40500000
```

- Automate buffer tuning

```
net.ipv4.tcp_moderate_rcvbuf = 1
```

- Allowing a large buffer doesn’t allocate a large buffer until it is needed
- Best documentation is in the Web100 kernel patch file *README.web100*

# →Configuring buffer in Windows Xp

- Registry settings
  `[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters]`
- Increase window size

```
GlobalMaxTcpWindowSize        40500000
TcpWindowSize                 40500000
```

# →Operating system defaults are woeful

- Windows Xp 8KB
  - 10Mbps ethernet
  - Dial-up
- Linux 32KB
  - 100Mbps ethernet
  - 3Mbps ADSL

# →2. Tuning TCP features - TCP operation

- TCP lifetime
  - Slow start to discover path bandwidth (the ‘window’ of packets which can be on the wire)
    - Additive increase
  - Incoming Acks form an “ack clock” of timestamp samples of the round-trip time
    - Small modifications to window
- Congestion appears as loss
  - Reduce window
    - Multiplicative decrease, important for stability
  - Re-enter slow start

# →Enabling TCP features – Window scaling

- Recall that the window controls the throughput of the connection. The *window* field in the TCP header is 16 bits.
- Window scaling
  - *window* = $2^n$ × *window* in TCP header
- *n* = 7 for fat pipe transfers
- Linux
  - Window scaling is on by default
  - *n* = 2 by default in recent kernels, because stupid ADSL routers broke
  - So

```
net.ipv4.tcp_adv_win_scale = 7
```

# →Enabling TCP features - Timestamps

- A *timestamp* and *timestamp echo* is added to the TCP header so that the round-trip time can be more accurately estimated
- So the Ack clock which controls transmission is more accurate and thus bandwidth increases
- Linux
  - On by default
- Windows, off by default

```
Tcp1323Opts    3
```

# Enabling TCP features – Selective Acknowledgement

- SACK allows individual segments to be Acknowledged
  - One segment loss doesn't lead to invalidation of all later data in the pipe
- This leads to a more complex data structure in sender and receiver, so important to test that throughput is sustained

# →Enabling IP features – MTU

- Maximum transfer unit is the largest packet size on the path
- Usually 1500
- Need 9000 (jumbo frames) to have a hope at 10Gbps
  - Mathis' formula sets 4Gbps upper bound on 1500 bytes frames
  - Avoid oversubscribing reciever CPU
  - Neterion testing
    - Quad Opteron uses 30% of CPUs for TCP input processing with 10Gbps and 9000 byte frames
    - Fails to clear I/O buffers at 1500 byte frames

# →Enabling IP features – Explicit congestion notification

- Mechanism to differentiate loss and congestion
- Congestion has an unavoidable multiplicative decrease if Internet is to avoid congestion collapse
- Loss has no implication for stability, so missing frame can be retransmitted and throughput preserved
- TCP Reno also re-enters slow start

# →IP features – Router queuing – Fair queuing

- Router should maintain illusion that TCP flow is the only flow on the path
- As large falls in available bandwidth hurt throughput
- Fair queuing

# →IP features – Router queuing – RED

- Synchronisation
  - Congestion is a shared event
  - Leads to oscillation
  - TCP doesn’t dampen this as well as possible
- Random early drop an attempt to limit synchronisation

# →Network design

- Ack compression
  - –Ack clock distorted
  - –Long queues are bad
- Asymmetric path
  - –Ack clock causes transmit at wrong time
  - –May not be enough bandwidth for Ack back-channel
    - 1Gbps: 2Mbps @ 9000
    - 1Gbps: 6Mbps @ 1500
    - 10Gbps: 20Mbps @ 9000

# →What's still wrong with TCP?

- Sawtooth-shaped throughput as increases in throughput are probed
  - Poor burstiness control
- Recovery from congestion takes a very long time'
- Slow start is very slow
  - Additive increase
  - Takes a long time to count to 10Gbps
    - 70min at 7.5Gbps

# →Exotic TCP

- “TCP compatible”
  - Implements congestion control
- “TCP friendly”
  - ... and is fair to TCP flows on the same links

# →Optimising Reno TCP

- Use window scale, timestamps and explicit congestion notification options
  - Set window scale for 10Gbps
- Test performance of Selective Acknowledgment
  - Complex data structure at receiver
- Set buffer to bandwidth-delay product
  - Hopefully automatically
- Use large MTU
- Have zero loss, looking at about $10^{-13}$ for undersea links

# →BIC-TCP

- Slow start wastes a lot of probes to find bandwidth
- We could do a binary search with those probes
  - Half are wasted, but that's still better than slow start
- When close logarithmically (ie, slowly) approach target bandwidth
  - Avoid sawtooth
- On congestion revert to previous binary search low value and re-probe new bandwidth
  - Multiplicative decrease but rapid recovery if bandwidth unchanged (ie, late Ack arrival was loss)
-

# →Experience with BIC

- TCP compatible
- TCP friendly except at dial-up speeds
  - Not a real-life problem as few dial-ups connect more than one host
- Default in recent Linux kernels
  - No one noticed :-)

# →FAST TCP

- Not only an implementation, but an architectural renovation
- Independent algorithms for
  - loss recovery
  - window control (RTT timescale)
  - burstiness control (sub-RTT)
- Unlike TCP which conflates them, making improving any one of them difficult
- Not finished
  - Not yet TCP friendly
  - Does not yet discover increased bandwidth

# →FAST congestion control

- Based on queuing delay rather than loss
  - A continuum, so more data for decisions than loss's binary value
  - So less inclined to oscillate
- Equation based
  - So same algorithm used in bandwidth discovery as used in steady state operation
  - Explicit recognition that we are seeking an equilibrium of flow dynamics

# →FAST window control

- Returning Acks maintain a smoothed estimate of queuing delay and a loss indicator
- Window is set by equation which updates window based on bounded proportion of change of RTT estimate
    - Estimates distance from equilibrium and if close makes only small changes to window size

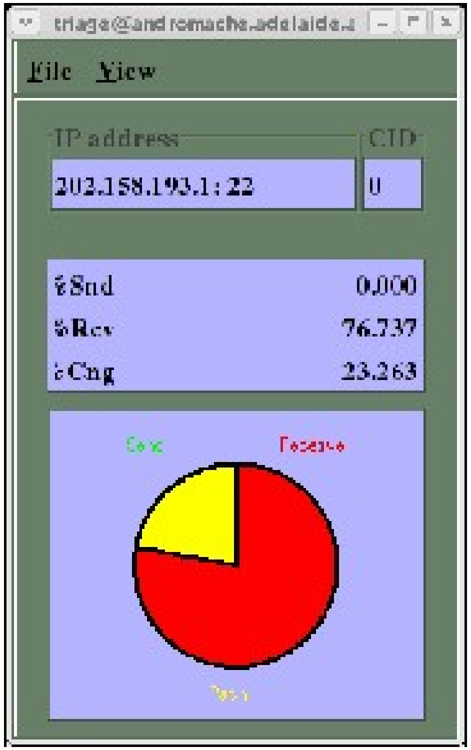
# →Competing efforts to FAST

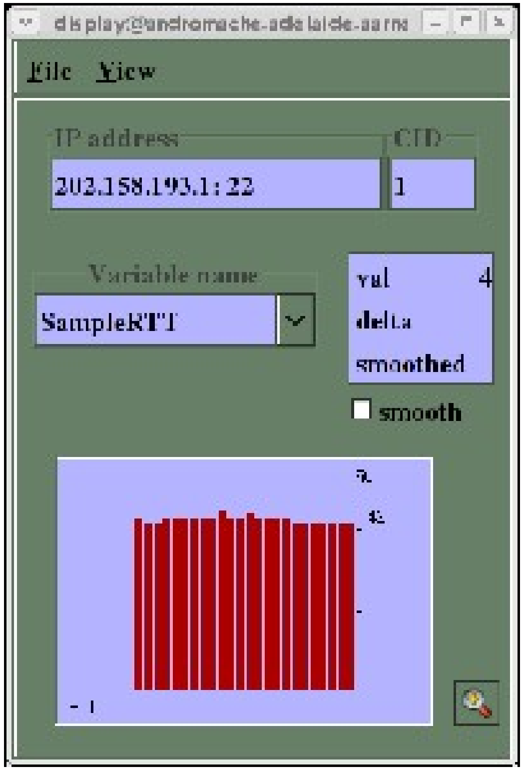
- FAST have a software patent
- Which they are not making royalty-free
- So won’t be in Linux, or perhaps even Windows
- Interest in similar renovations with no patent claims

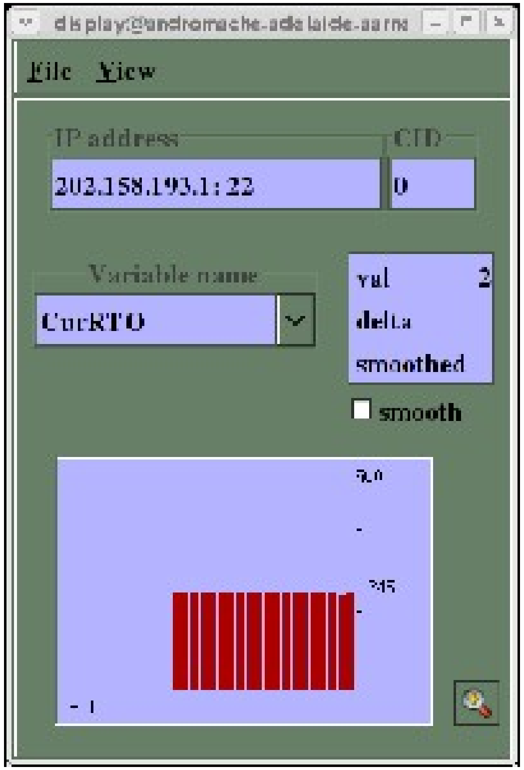
# →Others

- HDTCP and STCP
  - These both have differing gain functions then standard TCP
  - Making response to congestion more rapid
  - Allowing a closer estimate of the link bandwidth
  - But have all the other problems of TCP
- Westwood TCP
  - Uses incoming Acks to estimate packet rate and initialise slow start settings upon loss
  - Good for lossy environments like WLANs

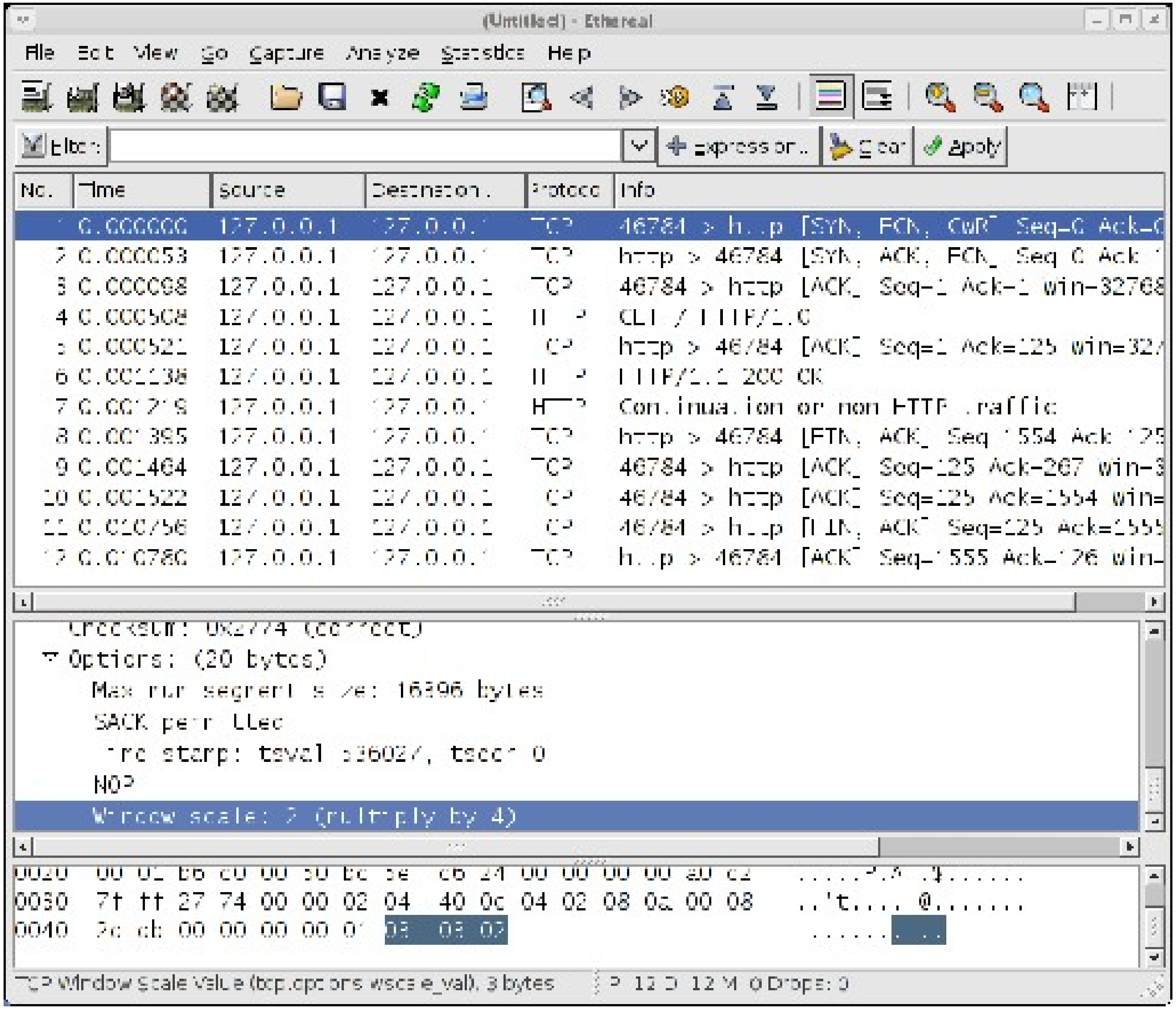
# →Web100 *www.web100.org*



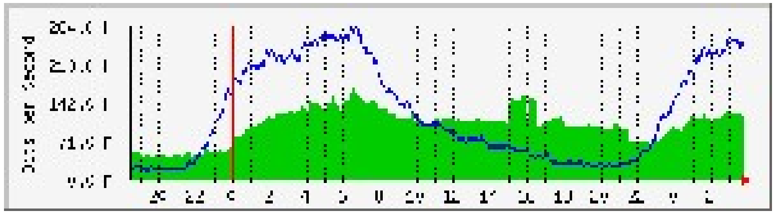



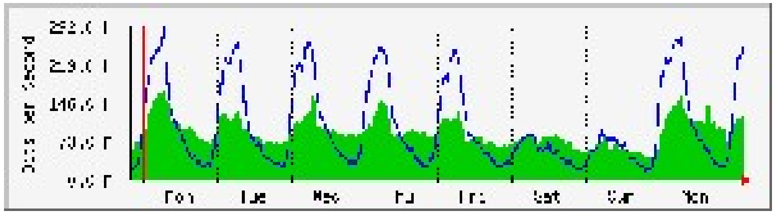
# →Ethereal

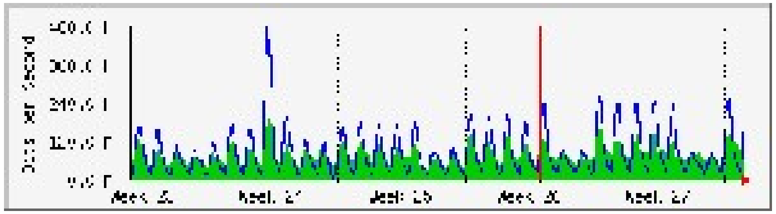


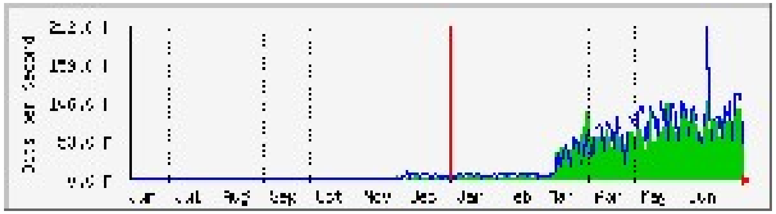
(Untitled) - Ethereal
File Edit View Go Capture Analyze Statistics Help
Filter: Expression... Clear Apply
No. Time Source Destination Protocol Info

# →MRTG, rrdtool, drraw and NetSNMP









# →System considerations

# →Fundamental limitation

- Latency
  - –Hard to get over speed of light
  - –Major limitation to distributed computing

# →Software

- Run recent kernel
  - So cutting edge Linux distribution
    - Fedora Core
    - Debian unstable or Ubuntu
  - Or Windows Longhorn beta
    - Released at various Microsoft “partner” events
- Apply the Web100 patches
- Acknowledge that the load is on the receiver
  - Even so, a good candidate for data reformatting
- But the sender can only send unaltered bytes from disk
  - *sendfile()* - Apache actually sends bulk data fast

# →CPU

- AMD Opteron, for the next two years until Intel get their act together
- Large MTU reduces the number of packets processed by the TCP receiver
- PCI bus
  - PCI-X 1.0    7.5Gbps
  - PCI-X 2.0    10Gbps

# →Disk drives

- SATA disk runs at 1.5Gbps
  SATA II disk runs at 3.0Gbps
- SAS runs over SATA link layer
- Native command queuing is a huge win for servers, but not for one single process doing a sequential read
- Speed/capacity/physical size trade-off
  - 7200RPM 3.5in is about 180GB
  - 15000RPM 2.5in is about 72GB
- Form factor is about to move to 2.5in, that is 60TB per rack
  - Implications for power density in computer room
  - Implications for connect technologies
- Need to ensure disk runs at full rate (acoustics)

```
hdparm -M 254 /dev/...
```

# →IDE versus SCSI

- Now SATA II versus SAS
- SATA II was designed to better SCSI/SAS
- Somewhat pointless, since manufacturers are using SAS v SATA to segment the marketplace into server and client
    - Except for Western Digital

# →Disk attach

- No obvious winner
  - Fiber Channel is slow
  - SATA can be switched, but no product
  - iSCSI has a lot of overhead
  - ATAoE looks attractive, but is there enough CPU to run this, the TCP stack and the disk subsystem?

# →Ethernet adapters

- Features: checksumming, TSO, interrupt coalescing, lots of buffer, jumbo frames
- 1Gbps Intel Pro/1000 Server MT
- 10Gbps Neterion (was S2IO) Xframe II
- State-full TCP offload adapters are not really suitable for this task, since these cannot take advantage of high performance variants of TCP protocols
  - Chelsio T210
- Duds: RealTek 8110- and 8160-series of GbE NiCs, early Intel GbE NICs

# →Disk subsystems

- A single SATA II or SAS15,000RPM disk will do about 750Mbps sustained
  - –Noting that throughput varies across the disk
- So we need to write to multiple disks simultaneously to improve the throughput
- RAID0 striping
- RAID1 + RAID0 striping and mirroring
- A very good RAID5
  - –And most RAID5 controllers are poor

# →End