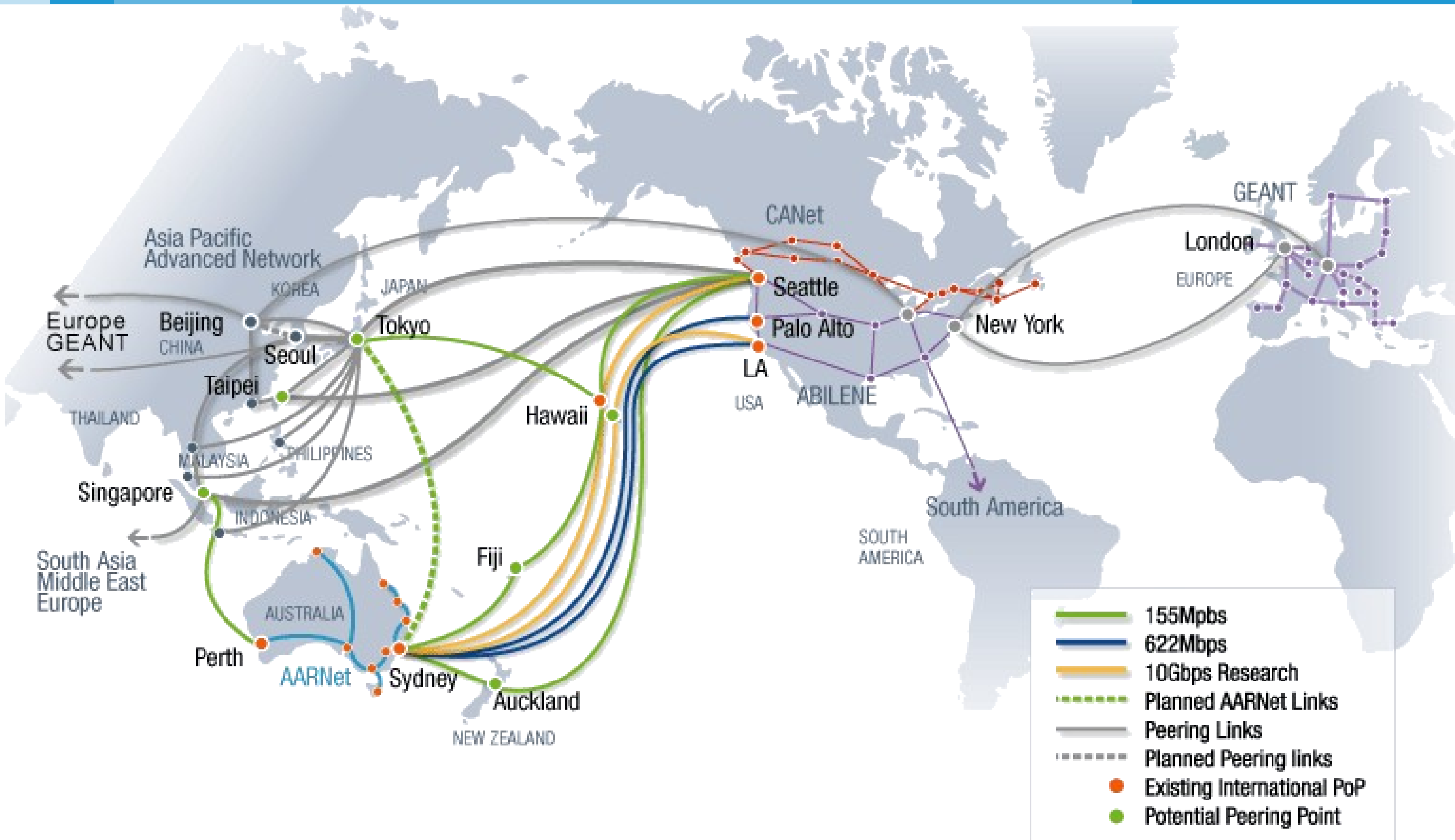
Achieving high performance file transfers across gigabit networks

QUESTnet, 2005-07-07
Coolum

Glen Turner

aarnet
Australia's Academic and Research Network

→AARNet's links to overseas research networks



→Packet

Link layer	Network layer	Transport layer	
Ethernet	IP	TCP	Data

→Filling the pipe and window size

```
+--------------------------------+--------------------------------+
|          Source port           |        Destination port        |
+--------------------------------+--------------------------------+
|                         Sequence number                         |
+-----------------------------------------------------------------+
|                     Acknowledgement number                      |
+--------+------------+----------+--------------------------------+
| H len |            |U A P R S F|             Window             |
+--------+------------+----------+--------------------------------+
|            Checksum            |         Urgent pointer         |
+--- --- --- ---+--- --- --- ---+--- --- --- ---+--- --- --- ----+
| Option=NOP     | Option=NOP     | Option=TS      | Opt len=10     |
+--- --- --- ---+--- --- --- ---+--- --- --- ---+--- --- --- ----+
|  Timestamp option: timestamp value                              |
+--- --- --- --- --- --- --- --- --- --- --- --- --- --- --- ----+
|  Timestamp option: timestamp echo reply                         |
+--- --- --- --- --- --- --- --- --- --- --- --- --- --- --- ----+
|                              Data                               |
.                                                                 .
+-----------------------------------------------------------------+
```

aarnet

→Window size: how much to send

- The advertised window reflects the amount of free buffer in the host
  - Small amounts of free buffer are not advertised
    - “silly window syndrome”
- The transmitter can safely send this much data

→Window size constrains throughput

- Let's advertise 64KB of window
  - The naïve maximum
- But a 1Gbps pipe 100ms long can contain 12,200KB of data
- The *Window scale* TCP option can be negotiated at the start of a connection
  - $65536 \times 2^{s} = 12{,}200\text{KB}$
    $s = 7$
- The window size is a measure of throughput, since

  $$\textit{throughput} = \textit{window} \div \textit{round trip time}$$

  and *round trip time* is reasonably constant for a connection

aarnet

→How much buffer do we need?

- We need enough to "fill the pipe"
  - 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

→Calculating buffer

- Discover round trip time

```
$ ping www.internet2.edu
  64 bytes from www.internet2.edu: time=242 ms
  64 bytes from www.internet2.edu: time=241 ms
  64 bytes from www.internet2.edu: time=241 ms
  64 bytes from www.internet2.edu: time=242 ms
```

- Calculate bandwidth–delay product

  $1\text{Gbps} \div 8 \times 0.242\text{s} = 29\text{MB}$

- Operating systems defaults

–Windows Xp	8KB	0.02% of 29MB
–Linux	32KB	0.11% of 29MB

→Configuring buffer in Linux

- Edit */etc/sysctl.conf*
- Increase window scaling

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

- Increase maximum allowable socket buffers

```
net.core.wmem_max = 30408704
net.core.rmem_max = 30408704
```

- Increase TCP buffers

```
net.ipv4.tcp_wmem = 4096 65536 30408704
net.ipv4.tcp_rmem = 4096 87380 30408704
```

- Automate buffer tuning

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

- 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]`
- Enable window scaling and timestamps

```
Tcp1323Opts                 3
```

- Increase window size

```
GlobalMaxTcpWindowSize      30408704
TcpWindowSize               30408704
```

aarnet

→Transmit scheduling and round trip time

```
+-------------------------------+-------------------------------+
|          Source port          |        Destination port       |
+-------------------------------+-------------------------------+
|                        Sequence number                        |
+---------------------------------------------------------------+
|                     Acknowledgement number                    |
+-------+-----------+-----------+-------------------------------+
| H len |           |U A P R S F|             Window            |
+-------+-----------+-----------+-------------------------------+
|           Checksum            |         Urgent pointer        |
+--- --- --- ---+--- --- --- ---+--- --- --- ---+--- --- --- ---+
| Option=NOP    | Option=NOP    | Option=TS     | Opt len=10    |
+--- --- --- ---+--- --- --- ---+--- --- --- ---+--- --- --- ---+
|  Timestamp option: timestamp value                            |
+--- --- --- --- --- --- --- --- --- --- --- --- --- --- --- ---+
|  Timestamp option: timestamp echo reply                       |
+--- --- --- --- --- --- --- --- --- --- --- --- --- --- --- ---+
|                              Data                             |
.                                                               .
+---------------------------------------------------------------+
```

→When should the sender transmit the next segment?

- So it arrives just as the receiver is ready for it
- We need a good estimate of the round trip time
- Each Ack contains a sample of the round trip time

$$rtt = \text{now() - Timestamp echo reply}$$

- We smooth this estimate of the round trip time

$$srtt_{n+1} = \alpha\, rtt + (1 - \alpha) srtt_n$$

- And calculate the variance of the estimate

$$rttvar_{n+1} = \beta\,|\,rtt - srtt_{n+1}| + (1 - \beta)\, rttvar_n$$

- There is an Ack for every two packets send, so you can look upon this estimate as being maintained by an “Ack clock”

aarnet

→When should the sender re-transmit a segment

- When the Ack is delayed
- Most TCP implementations use an Ack receive time out of

$$rto = srtt + 4 \times rttvar$$

- Which is why TCP is bad on wireless LANs. The jitter from the CTS/RTS operation of 802.11 leads to ($4 \times rttvar$) overwhelming the estmate.
- You see *rto*s of up to 10s
- It then takes TCP a long time to detect a single packet loss, a common event in a WLAN

aarnet

→Ack back-traffic

- A 1Gbps file transfer will generate
  - 13Mbps or Acks with ethernet frames
  - 2Mbps of Acks with jumbo frames
- If BGP path asymmetry leads to the Acks being congested then the transfer is slow
- Particularly a problem with Internet2 sites, as these connect to a commodity ISP and I2 (I2 doesn't do commodity traffic, unlike AARNet)
  - Traffic goes down 10Gbps research link
  - Acks come up commodity link. If that link is 2-10Mbps there can be problems

→Network design and RTT estimate

- Elements of network design can degrade the RTT estimate, leading to poorer performance than the “headline” 1Gbps
  - Loss
    - RTT estimate is only good in TCP steady state
  - Jitter needs to be low
    - Router design
    - Link load
  - Symmetric paths
    - BGP configuration
  - Queuing and Ack compression
    - Queuing algorithms and QoS packet sizes

aarnet

→Congestion control and loss

```
+-------------------------------+-------------------------------+
|         Source port           |        Destination port       |
+-------------------------------+-------------------------------+
|                        Sequence number                        |
+---------------------------------------------------------------+
|                     Acknowledgement number                    |
+-------+-----------+-----------+-------------------------------+
| H len |           |U A P R S F|             Window            |
+-------+-----------+-----------+-------------------------------+
|           Checksum            |         Urgent pointer        |
+--- --- --- ---+--- --- --- ---+--- --- --- ---+--- --- --- ---+
| Option=NOP    | Option=NOP    | Option=TS     | Opt len=10    |
+--- --- --- ---+--- --- --- ---+--- --- --- ---+--- --- --- ---+
|  Timestamp option: timestamp value                            |
+--- --- --- --- --- --- --- --- --- --- --- --- --- --- --- ---+
|  Timestamp option: timestamp echo reply                       |
+--- --- --- --- --- --- --- --- --- --- --- --- --- --- --- ---+
|                              Data                             |
.                                                               .
+---------------------------------------------------------------+
```

→Congestion control

- TCP’s congestion control is why the Internet is stable
- Avoid congestion collapse
  - Detect congestion and slow down radically
  - Don’t send lots of packets until congestion can be detected
  - Congestion is detected by Ack timeouts

→Life of a TCP connection

- “Slow start” probing to find RTT estimate
- TCP enters steady state with an incoming Ack for every two outgoing segments, this Ack clock determining when the next segment is sent
- Occasional probes to see if more bandwidth available

→Loss

- An Ack is lost or late (*rto* has expired)
- "Slow start" is re-entered to determine a new RTT estimate
  - Throughput plummets
  - As bandwidth increases TCP takes longer to return to a steady state which uses the maximum bandwidth
  - At 1Gbps this can be an hour
- If three Acks arrive for the same segment, then a single packet has been lost. Re-transmit it and halve the window
  - Throughput reduces

→Synchronisation

- Congestion or loss effects lots of connections simultaneously
- They all back off, they all probe, they all ramp up, congestion re-occurs
  - Oscillation of throughput
  - All protocols which have a long-tailed distribution are vulnerable to oscillation, not just TCP
- We want to distribute network events in time
  - Can't do much about loss
  - For congestion let's drop packets with an increasing random probability as we get near the end of the queue
  - Random early detect

aarnet

→Loss or congestion?

- Routers drop packets when they exceed the queue

```
               fair queue                       drop with p    drop all
       |--------------------------------|---------------|
```

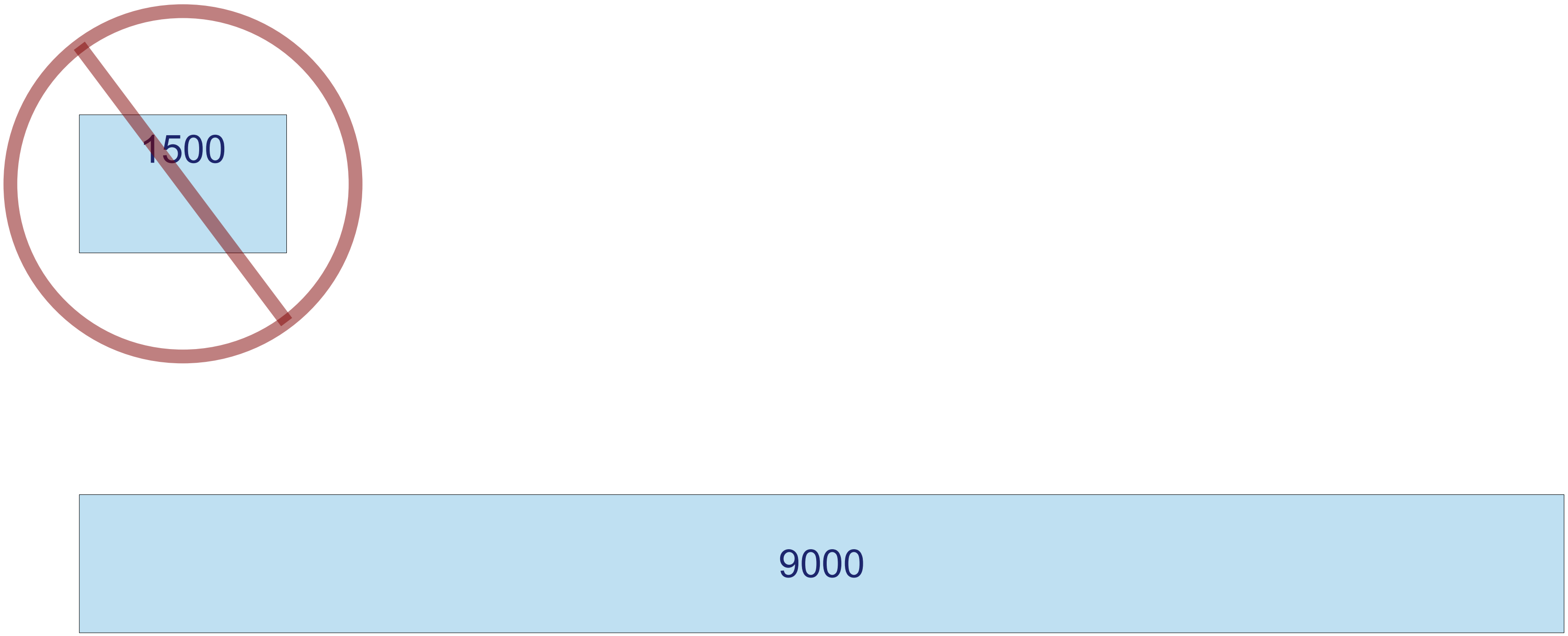
- Loss drops packets too
  - These could be re-transmitted immediately
- TCP must treat a late Ack as congestion not as loss
  - Otherwise congestion collapse of the Internet
- Explicit congestion notification allows TCP to be signalled about congestions
  - Presumably all other late Acks are loss

```
             fair queue       mark ecn with p  drop with p  drop all
     |----------------------|---------------|-------------|
```

→Network design and loss

- Banish loss
  - Clocking errors
    - Exactly one clock on a link
  - Ethernet autoconfiguration
    - Turn it on, at both ends
    - Turn it off, at both ends
  - G.703 and optical power budgets
  - Graph CRCs and carrier loss for interfaces
  - Gigabit wireless. Hmmm.
- Use fair queuing with random early detect to control synchonisation
- Support ECN

→Maximum transfer unit



→Path MTU

- The largest packet which can be successfully sent across the path connecting the two hosts
- We want this to be big
  - –The receiver has a lot of work to do per packet so sending less packets is good
  - –Neterion testing, reported last week on netdev
    - MTU=9000 30% CPU on quad Opteron at 10Gbps
    - MTU=1500 Machine dies
  - –Mathis' formula says that increasing MTU is the only realistic option for increasing TCP performance

aarnet

→Mathis' formula

$$rate \leqslant \frac{mss}{rtt} \times \frac{1}{\sqrt{p}}$$

- *mss* is maximum segment size, say 1460B
- *rtt* is round-trip time, say 260ms
- *p* is loss probability, say 10^{-11}%
- *rate* is maximum throughput

aarnet

→Exceeding the upper bound

- Reduce round trip time
  - Speed of light in fiber is dropping by 5% per decade
- Reduce loss probability
  - But increasing DWDM channel bandwidth increases loss probability
  - So manufacturers aim for ITU compliance
- Increase MTU
  - One product cycle from router and switch vendors

→MTU and network design

- Specify jumbo frames in purchases
- Configure routers to use the maximum link MTU on router-router links
  - Want to present all those 9000 bytes to the host, not steal them for MPLS, GRE, etc
- Middle-boxes have less support than switch manufacturers
  - Firewalls especially
- Not all hosts do jumbo frames
  - What are Apple thinking?

→Exotic TCP

→Limitations of TCP

- Additive increase is too slow for long fat pipes
- Multiplicative decrease is too much for most congestion events
- Needs zero loss and zero congestion to achieve link bandwidth
- Packet loss causes oscillation as only signal is arrived or late
- Prone to cause link bandwidth oscillation

→BIC-TCP

- Replaces slow start with a binary search
  - Double the information per probe packet
- Once close it logarithmically approaches upper bandwidth estimate
  - Less overshoot, preventing sawtooth
- Loss causes a return to previous binary search state
- Will rapidly discover free bandwidth
  - Say from another transfer finishing
- TCP-friendly at high speeds, unfair at dial-up speeds
- The default in recent Linux
  - Experience is good

→FAST TCP

- An equation-driven algorithm which uses queuing delay as the measure of congestion
  - Queuing delay is less binary than loss, limiting oscillation
  - It is independent of the window size
  - Can be input to gain equation rather than an algorithm
    - These have better control of gain, limiting sawtooth oscillation at link capacity

→FAST TCP

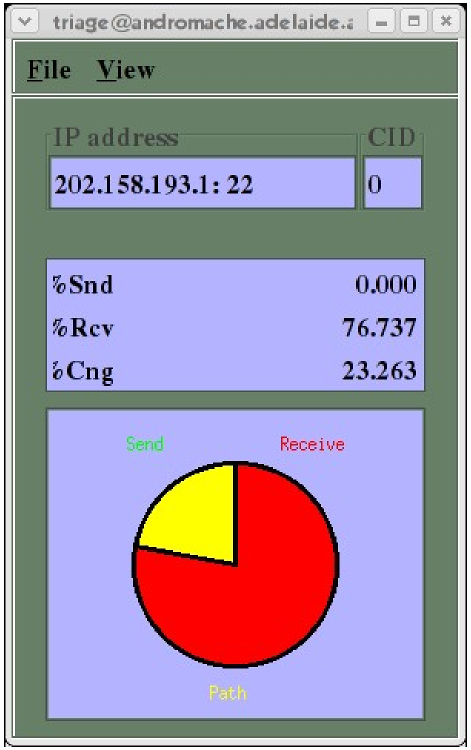
- Not only an implementation, but an architectural renovation
- Independent algorithms for
  - loss recovery
  - window control (RTT timescale)
  - burstiness control (sub-RTT)
- Unlike TCP which conflates them, making improving any one of them difficult
- Not finished
  - Not yet TCP friendly
  - Does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

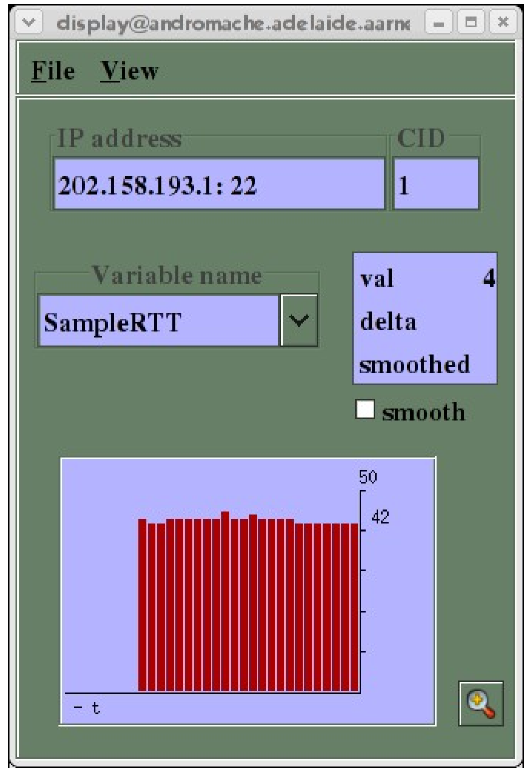
aarnet

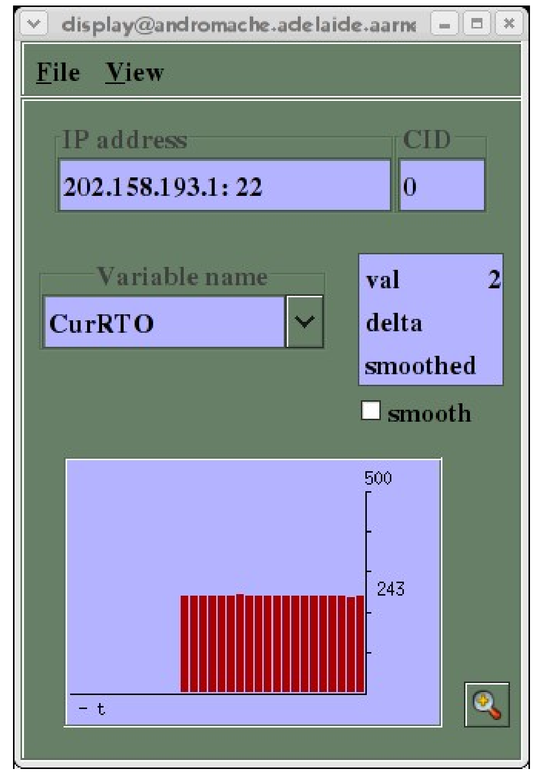
→Others

- 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







→System considerations

→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

aarnet

→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 160GB
  - 15000RPM 2.5in is about 36GB
- 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

→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

aarnet

→Disk subsystems

- A single SATA II 10,000RPM disk will do about 460Mbps
- So we need to write to multiple disks simultaneously to improve the throughput
- RAID0 striping
- RAID1 + RAID0 striping and mirroring
- A good RAID5

aarnet

→PCI bus

- PCI-X 1.0 7.5Gbps
- PCI-X 2.0 10Gbps

aarnet

→End