TCP performance

LinuxSA
Adelaide
2003-09-16

Glen Turner
glen.turner@aarnet.edu.au

aarnet
Australia's Academic &
Research Network
www.aarnet.edu.au

Part 1
TCP performance

LinuxSA
Adelaide
2003-09-16

Glen Turner
glen.turner@aarnet.edu.au

aarnet
Australia's Academic &
Research Network
www.aarnet.edu.au

Topics

TCP primer

Limits to TCP performance

Application program, equipment, and network design implications

Using Web100

Design of Web100

TCP primer

What is TCP?

"Transmission control protocol"

Specified in *RFC793*

- Free online at
  http://www.ietf.org/rfc/rfc793.txt

RFCs are not like ISO standards

- Once issued RFCs are not updated, they are amended by other RFCs

- So IEEE802.3-2002 is the Ethernet standard

- But RFC793 amended by ..., ..., is the TCP standard

What does TCP do

Provides a connection which is

- Simple
- Multiplexed
- Bidirectional
- Reliable
- Flow controlled
- Multiplexed

Simple connection

TCP just provides a stream of bytes

- No record boundaries
- Says nothing about latency or jitter

Only one feature

- "Urgent" data for Ctrl+C

Simple connection

Some higher layer is responsible for implementing record boundaries

- Usually the application, the TCP/IP suite of protocols has no formal session or presentation layers

- Increasingly an XML library ("presentation") over HTTP or BEEP ("session")

Multiplexed

One link can carry simultaneous connections

One host can have many connections, even to the same remote host

Connection is defined by
(*label_space*,
*src_af, src_addr, src_port,
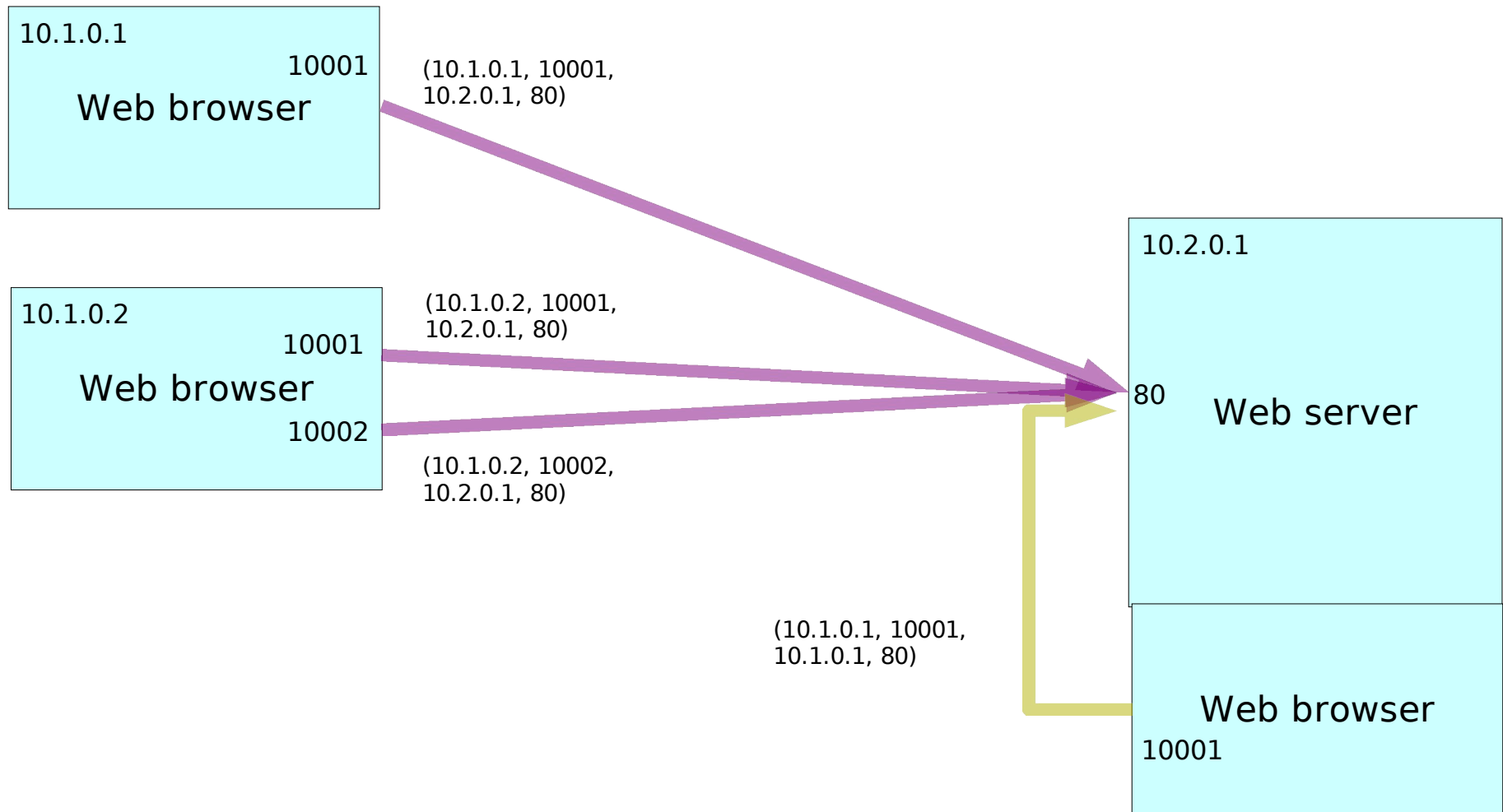dst_af, dst_addr, dst_port*)

Ports

Ephemeral ports provide multiplexing

- Allocated on demand by operating system
- `netstat -t -n`

Well-known ports provide application selection

- Web servers listen on *src_port* = 80
- `netstat -t -n -l`

Multiplexing example

10.1.0.1

10001

Web browser

(10.1.0.1, 10001,
10.2.0.1, 80)

10.1.0.2

10001

Web browser

10002

(10.1.0.2, 10001,
10.2.0.1, 80)

(10.1.0.2, 10002,
10.2.0.1, 80)

10.2.0.1

80

Web server

(10.1.0.1, 10001,
10.1.0.1, 80)

Web browser

10001

Reliability

Lost or corrupted data is retransmitted

- That's why there is nothing said about latency

Loss is detected by a late Ack for a packet

Flow control

Do not over-run receiver's hardware or
buffering

TCP header

```
+---------------------------+---------------------------+
|        Source Port        |     Destination Port      |
+---------------------------+---------------------------+
|                   Sequence Number                     |
+-------------------------------------------------------+
|                 Acknowledgment Number                 |
+-------+-------+-+-+-+-+-+-+---------------------------+
| Data  | Res-  |C|E|U|A|P|R|S|F|                       |
| Offset| erved |W|C|r|c|s|s|y|i|         Window        |
|       |       |R|E|g|k|h|t|n|n|                       |
+-------+-------+-+-+-+-+-+-+---------------------------+
|        Checksum           |      Urgent Pointer       |
[---------------------------+----------------+----------]
[               Options                      |  Padding  ]
[--------------------------------------------+----------]
[                      data                             ]
.                                                       .
.                                                       .
[-------------------------------------------------------]
```

Header – simple connection

Sequence number

- Modulo index into byte stream

Urgent pointer

- For a small amount of out-of-band data
- Not used much, as an early implementation error has lead to divergent interpretations

Header - multiplexing

Source port

Destination port

Header – reliability

Checksum

- error detection

Acknowledgement

- Ack number

Sequence and window

- Unacknowledged data which may need to be retransmitted if an Ack does not arrive in time.

Header – flow control

Window

- Always less than amount of receive buffering available

Limits to TCP performance

Sub-topics

- Bandwidth-delay product
    - Buffers sizing
- Enhancements to TCP
    - Window scaling, timestamps, SACK, ECN
- Mathis' formula
    - Bit error rates, maximum transfer unit

Bandwidth-delay product

Bits of data in transit between hosts is *bandwidth × rtt*

bandwidth is in bits per second

rtt is round-trip delay, in seconds

bdp is usually expressed in MB, so convert formula from b to MB

- $bdp = (bandwidth/8{\times}10^6) \times rtt$

Example bandwidths

Ethernet

- 10Mbps-HD, 100Mbps, 1000Mbps

Leased line

- 128Kbps, 2Mbps, 34Mbps, 155Mbps

ADSL

- Around 3000/256Kbps or 0.5Mbps

Modem

- 56/34Kbps

Example delays

28ms   Adelaide-Sydney

150ms Sydney-US east coast

200ms US east-west

500ms Sydney-Sydney via satelite

Example bandwidth delay product

Take slowest link in path

Take computed round-trip time

Example: Glen's desk

- slowest link in path is fast ethernet
  - *bandwidth* = 100,000,000bps
- *rtt* to www.internet2.edu is 0.240s
  - measured with ping
- *bdp* = (100,000,000/8,000,000) × 0.240
  = 3MB

TCP buffers

To run at full speed TCP may need to buffer an entire window of unacknowledged data

- that is, the value of the bandwidth-delay product

Most operating systems are tuned for 10Mbps LANs "out of the box"

eg: Windows XP has 17,520 bytes (12 packets)

- *rtt* < 2ms at 100Mbps

Linux buffer tuning

/proc/sys/net/core

- rmem_default
  - Default receive window (64KB)
- rmem_max
  - Maximum receive window (64KB)
- wmem_default
  - Default send window (64KB)
- wmem_max
  - Maximum send window (64KB)

RTT with Linux 2.4 defaults

56Kbps modem, no wucking furries

- $64 \times 2^{10} \times 8 = 56 \times 10^3 \times rtt$
  $rtt = 9s$

2Mbps leased line, uh oh

- $64 \times 2^{10} \times 8 = 2 \times 10^6 \times rtt$
  $rtt = 262ms$

10Mbps ethernet, OK for LAN, else bad

- $64 \times 2^{10} \times 8 = 10 \times 10^6 \times rtt$
  $rtt = 52ms$

100Mbps $rtt = 5.2ms$, tuning time

Let's go faster

AARNet3

- slowest link in path is 1Gbps

- but speed of light in fiber unchanged

- so 90% of traffic will have *rtt* > 200ms

Bandwidth product delay ≈ 40MB

So some tuning is needed :-)

What if I ask Linux for a 40MB buffer per connection?

Receiver buffer allocation algorithm

- Initially allocate receive buffer for 4 packets, advertise Window accordingly
- Increase by 2 packets per Ack
- Until rmem_default is reached

Caching

- TCP control information to a host is cached for 10 minutes
    - cwnd, rtt, rttvar, ssthresh, reorder

What if I ask Linux for a 40MB buffer per connection?

Transmit buffer allocation

- Application's problem :-)

Interface output buffer

- Needs to be set accordingly

Configuration

/etc/sysctl.conf

- net.core.rmem_max = 41943040
  ...

/sbin/ifup-local

- if ["${1%%[0-9]*}" = "eth"]
  then
      /sbin/ifconfig "${1}" txqueuelen 1000
  fi

Windows bigger than 64KB

Doesn't the TCP header have 16 bits for Window? Yes.

TCP window scaling option

- Negotiated at connection startup
- Window is 32 bits at hosts, but 16 bits on packet
- Number of bits to right-shift the Window value in the packet
- *RFC1323*

More configuration

sys.net.ipv4.tcp_timestamps=1

sys.net.ipv4.tcp_window_scaling=1

sys.net.ipv4.tcp_sack=1

What are these timestamps

TCP samples one packet per round-trip, uses this to build estimate of RTT and the variance of the estimate

But fat pipes have thousands of packets, so far too few packets are measured for good RTT estimate

If we lose the "RTT sample" packet we are really stuffed

A timestamp allows every packet to contribute to the RTT estimate

Why is an accurate RTT estimate important?

It is used to answer two questions:

- Is it time to send the next packet (assuming I'm facing an open window)
- Is the Ack late, and thus I've dropped a packet
  - and need to enter slow start

Variance of RTT is also maintained

- So we can define "late" better

You can look at the back-flow of Acks as an "Ack clock" which maintains the RTT estimate

What is this SACK

Selective Acknowledgement

We have thousands of packets in the pipe

We drop one

We want to retansmit that one, not the thousands of subsequent packets too

We want to acknowledge individual packets

Also works well for lossy media, WLAN

What is ECN?

Explicit congestion control

Packet is dropped

- Is it congestion?
- Is it corruption/loss?

Re-transmission actions differ:

- Congestion: slow-start; halve data rate
- Loss: retain data rate

Must assume congestion to avoid congestion collapse of Internet

What is ECN?

But if loss if explicity signalled, then there is no ambiguity

Routers mark packets forwarded on a congested link

- Setting IP's ECN bits to a "congestion encountered" value

Receiver reflects this back to sender

- TCP's ECN Echo set until CWR heard

Sender backs off

- Setting TCP's Congestion Window Reduced

What is ECN?

A packet drop on an ECN path is loss

Retransit immediately

- But without the IP "ECN enabled" value
- So if it too is dropped then back off

Some firewalls with stale software reject ECN traffic

- www.news.com.au

Configuration

- net.ipv4.tcp_ecn=1

Mathis' formula

Mathis' formula
TCP throughput

Upper bound on TCP transfer *rate* (bps)

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

Mathis, et al. *The macroscopic behavior of the TCP congestion avoidance algorithm.* CCR, 27 (3), July 1997.

Where

mss Maximum segment size (B), say 1460 B

rtt Round trip time (ms)

   60ms   Adelaide - Sydney
   160ms   Sydney – Seattle

p   Loss probability (%), say 10^{-11}%

Mathis's formula
Example

Ethernet file transfer from US east coast

- *mss* = 1460B, as ethernet MTU is 1500
- *rtt* = 220ms, Adelaide-Sydney-Seattle
- *p* = 10^{-11}%, as 10^{-13} is typical fiber BER

Substituting

- *rate* ≤ 2.1Gbps

Formula for non-mathematicans

To increase rate, in priority order

- Decrease round-trip time
- Decrease loss probability
- Increase maximum segment size

Decrease round-trip time

That is, increase the speed of light in fiber

- Happening, but <%5 improvement per decade

Using the speed of light in free space is limited by earth's curvature

- lasers or microwave radios need "line of sight"

Decrease loss probability

SDH specifies a typical bit error rate of 10^{-13}, flags an link down at 10^{-6}

Can actually do better, by reducing phase errors, but this limits bandwidth

Need a new "typical" specification from ITU-T, this might take a decade from proposal to implementation

Increase maximum segment size

TCP MSS is the link's maximum transmission unit minus the TCP/IP header

Increasing MTU requires a new generation of hosts and routers

Maximum MTU

IPv4 is 64KB

Ethernet MTU is 1500, changing this breaks 10/100 interoperability

Summary

Numbers which are reasonable at 10Mbps are large at 100Mbps and unreasonable at 1Gbps

Throughput (Mbps)	RTT between losses (sec)	Congestion window (packets)	Maximum packet loss (%)
1	5.5	8.3	0.02
10	55.5	83.3	0.0002
100	555.5	833.3	0.000002
1,000	5,555.5	8,333.3	0.00000002
10,000	55,555.5	83,333.3	0.0000000002

More unreasonable numbers

Back-to-back packets on gigabit ethernet

- Maximum size frames: 81,724fps
- Minimum size frames: 1,488,095fps

Not the worst case, as ethernet has a minimum frame size and an inter-packet gap

- An SDH interface could deliver

Even so, no host will cope with 1.5m interrupts per second

Better interface cards

Template and TCP segmentation offload

- Also called "large send offload"

Implies checksum offload

Interrupt combining

- A burst of incoming packets causes only one interrupt

Fast, wider, better PCI bus

- 64-bit PCI-X

Better device drivers

Minimise data copying

Support features of better interface
cards

- e1000 does, tg3 doesn't

Dynamic selection of interrupt versus
polling

- use interrupts when idle
- use polling when busy

Consequences for application design

Sub-topics

- Nagle algorithm
- Transaction protocols and round-trips

Nagle Algorithm

Don't want a packet per keystoke

Algorithm

- is an outgoing Ack likely?
- Yes
    - can we send a full packet?
        - no: try to piggy-back the data onto an Ack
        - yes: send data
- No: send data

Nagle problems

Introduces delay in transaction protocols

Fails for disk I/O on ethernet

- 4KB disk blocks leads to 3 1500B packets
- So an Ack is always pending
- So we always wait

TCP_CORK

New API to make boundaries of upper-layer protocol data unit known to TCP

- When TCP_CORK set only full packets transmitted
- When TCP_CORK unset the remaining data is send immediately

Can also do this by disabling Nagle and using a single write() to send the PDU

sendfile()

New API to send a file over a connection

- Less copying of data than
  - bytes = read(file, buffer, sizeof(buffer)); write(socket, buffer, bytes);
- No switching between kernel-user-kernel space
- Can reuse buffers (the disk I/O buffer gets reused as the network I/O buffer)

Compatible with TCP_CORK

Latency
A fundamental limit

We can fit more packets onto a fiber, but we can't get the to the other end any faster

- 1ms per 200Km

In particular, crossing the Pacific is at least 70ms one-way

Applications and latency

Lots of application protocols have too many round-trips.

HTTP GET

- Fetch HTML
- It refers to a CSS stylesheet, fetch it
- Can now start page render
- It refers to images, fetch them

Say 3 RTTs, roughly 420ms to render a US-hosted we page.

User experience and latency

User interface people suggest 0.3s is the maximal time for a screen update

So an interactive protocol requiring more than two RTTs is bad news

Additional criteria for network equipment

Mathis formula

- MTU
  - At least 9000, if not 64KB
  - Multiple MTU per subnet
- Bit error rate
  - 10^{-13} isn't good enough
- Round-trip time
  - Latency is everything
    - SCCN versus Flag

Consequences for network design

Decrease latency

- Take the direct path
- Versus traffic engineering

Wizard certificate

You are all now TCP tuning wizards

Welcome to the few

- assuming your ISP will tell you about their network

- assuming you have administration access on your computers

- Hmmmm

Wizard gap

But there are millions of computers on the Internet

So we need to reduce the Wizard Gap to mere mortals can get good performance

Web100

Part 2
Web100 Project

LinuxSA
Adelaide
2003-09-16

Glen Turner
glen.turner@aarnet.edu.au

aarnet
Australia's Academic &
Research Network
www.aarnet.edu.au

Basic observation

Modern computers all have capable fast ethernet interfaces

- PC hardware has enough headroom to put a sustained 100Mbps through its fast ethernet interface

- Not so for 1,000Mbps

Yet most users don't come close to their theoretical throughput rates

Some people are lucky

They have the expertise, time and connections to locate the performance bottlenecks and fix them

Web100 is for everyone else

Stating the Web100 goal more formally

Provide the software and tools necessary for end-hosts to automatically and transparently achieve high bandwidth data rates (100 Mbps) over high performance research networks.

Web100 organisation

A NSF-funded project coordinated by Pittsburgh Supercomputer Center

Various other contributors

- National Center for Atmospheric Research
- Computer Science & Mathematics
  Oak Ridge National Labatory
- Data Intensive Distributed Computing Group
  Lawrence Berkeley National Laboratory

and a range of individuals

- Such as the presenter

Net100

Same goals as Web100

But allowed to break the rules

- Be unfair to other network users
- Drive through congestion

Net100 is for "mission centric" networks

- The network is built for the application, so the application has every right to be unfair to other applications

Web100 is for the Internet

Topics

Instrumenting TCP performance

APIs for instrumentation

- /proc/web100
- libweb100
- *TCP-ESTATS-MIB*
    - This is the area of the project I am responsible for implementing

Applications

Instrumenting TCP performance

aarnet

Australia's Academic &
Research Network
www.aarnet.edu.au

Where?

Most operating systems implement TCP entirely in the operating system kernel

- Unixen, VMS, Windows NT/2000/XP

- Not the only option: QNX

Add counters to kernel TCP data structures

Where?

Getting counters out of the kernel

- There is a lot of them
    - not *ioctl()* or other single-variable scheme
    - */proc*, but not widely used
    - */dev/kmem*, but ugly
- There is no single mechanism used by many operating systems

All this had better be hidden from Web100 applications

- libweb100

Connection state

Syn options

Traffic counts

Data sequence

Sender congestion triage

Congestion

What is counted? (2 of 3) Sender

Loss

Path, not Ack clock

Reordering

Round-trip time

Segment size

Buffer size and use

Local reciever

Packet counts

Buffer size and use

Window

How much does all this counting cost

No effect in normal case

Up to 1% of file transfer rate in worst case

- Lots of small packets on fast ethernet

Performance testing has been extensive

- A lot of the current Web100 users do massive file transfers and won't accept a performance hit

APIs for instrumentation

aarnet
Australia's Academic &
Research Network
www.aarnet.edu.au

Overview
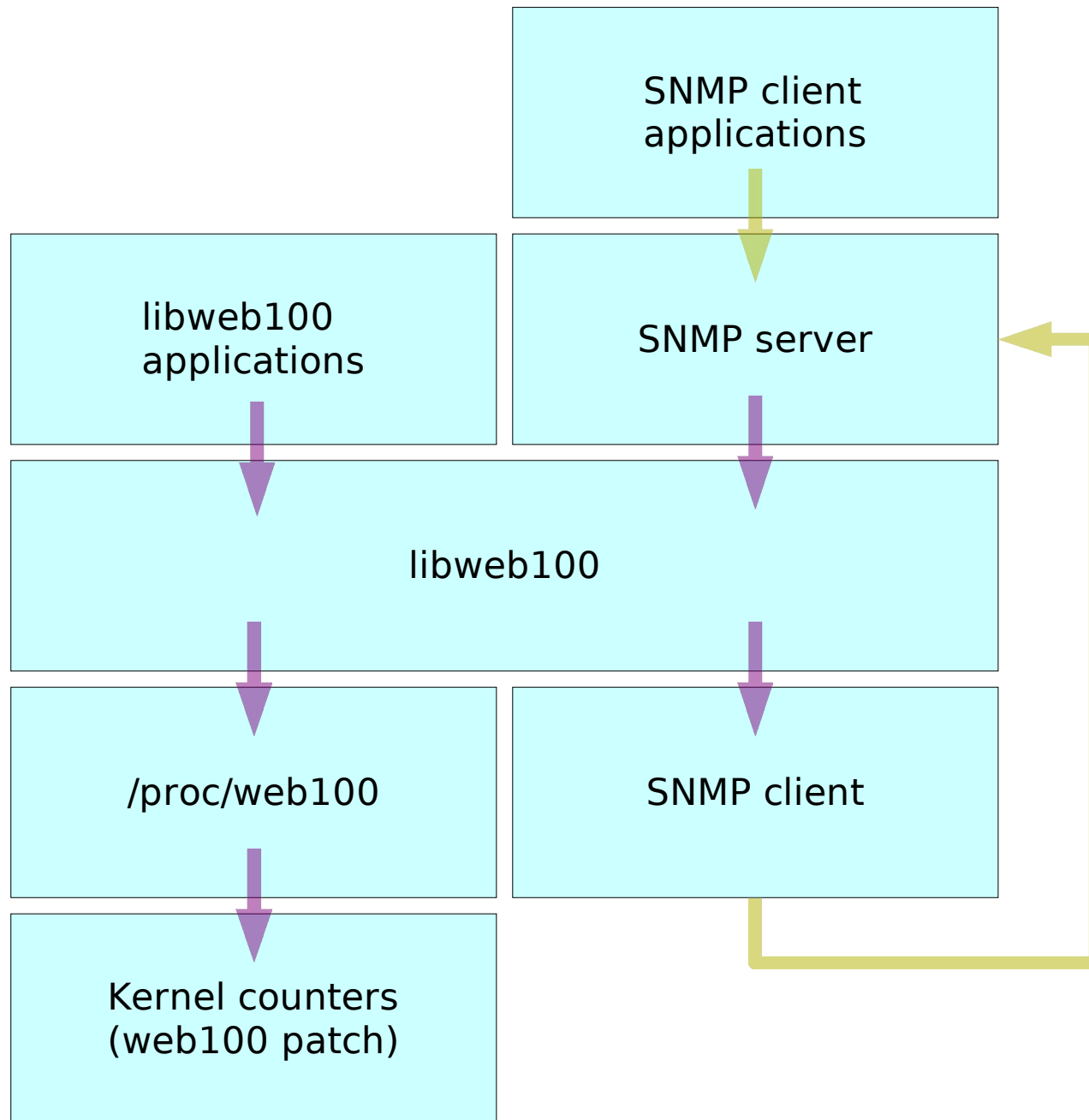
/proc/web100

Linux has a filesystem for kernel-user interaction - /proc

Originally a text version of UNIX SystemV's /proc

/proc/web100/header

Identifies kernel running Web100 and the variables collected

This allows some independence between kernel versions and library versions

For example, the kernel patch and library were extended to support IPv6.

- Previous library versions work with newer kernel code
- Newer library versions work with older kernel patches

/proc/web100/<cid>

Most high perfomance computing people only want to tune a single connection

- AARNet mirror an exception

Allow efficient access to a particular connection

- So all variables for a connection are under /proc/web100/*connection_identifer*

/proc/web100/<cid>/<group>

Web100 has a lot of variables

We want to pass these over in groups

- Not one at a time, as we want a snapshot of counters at a single moment

- Not all at once, as Web100 would do ridiculously large kernel/user I/O

Group design careful to collect related variables together, so all are snapshotted at same time

libweb100

Offers same application programing interface on all operating systems

- Implemented using /proc/web100 on Linux

Offers same build environment on all machines

- Using web100-config command

Portability a major aim so that analysis tools can be easily ported and widely used

libweb100 data heirarchy

Variables access by following heirarchy

- *agent*, a machine
  - Usually local
  - Can also be remote, using SNMP
- *connection*, a TCP session
- *group*, a related set of variables
- *variable*, **location** of a instrument within a snapshot
  - Not it's value
- *instrument*, value of a TCP counter (only used wher "varaible" confusing)

Why odd treatment of variables?

Allows parsing to be done once

- Convert variable name to offset within the snapshot

But used many times

- Take a snapshot, ask for the value at the saved offset

Applications can efficiently carry textual representations of variable names

- Hardcoding these in a header reduces inter-version portability

Snapshots

web100_snap()

- Take a snapshot of all variables for one (agent, connection, group) tuple
- All the instruments in a snapshot are collected in the same instant

web100_snap_read()

- Parse a snapshot, returning variable values

Read a variable for a connection

```
#include <web100/web100.h>
...
int connection_id = argv[1];
char *variable_name = argv[2];
...
agent = web100_attach(WEB100_AGENT_TYPE_LOCAL, NULL);
web100_find_var_and_group(agent, variable_name, &group, &variable_p);
connection = web100_connection_lookup(agent, connection_id);
...
snap = web100_snapshot_alloc(group, connection);
web100_snap(snap);
web100_snap_read(variable_p, snap, &variable_value);
web100_snapshot_free(snap);
...
return variable_value;
```

Other libweb100 features
Snapshot logging

Can send a snapshot to a log file

And pull it back and parse it at convenience

Separate logging and analysis

- Easy to patch existing code (httpd, ftpd) to generate log records

Logged snapshots are architecture-dependent

- Could fix, but log disk usage would double on 32b machines

Other libweb100 features
Snapshot deltas

Returns a snapshot with the changes between two snapshots

All nasty semantic issues dealt with by library

- Zero versus null values

- Counter wraps

- Quantisation

Other libweb100 features
Remote agents

agent is usually WEB100_AGENT_TYPE_LOCAL

Working on WEB100_AGENT_TYPE_REMOTE

- Using SNMP to obtain variables of remote end
  - Access, athentication, authorisation, accounting issues are punted to SNMP
- Data transmitter carries more interesting set of TCP varaibles

libweb100 environment

C library style error handling

- Functions return `WEB100_ERROR`
- `web100_perror()`, `web100_strerror()`

Application build environment

- `cc `web100-config --cflags` -o example.o example.c`
- `cc `web100-config --libs` -o example example.o`

Installation is easy (uses GNU automake)

- `./configure; make;`
  `su -c 'make install; ldconfig; makewhatis -u -w'`

TCP Extended Statistics MIB
What is a MIB?

A "management information base" is a set of counters and tables

- Formally expressed in computer-parsable format

- Expression carries some semantic information

  - Data types, ranges, etc

- Each variable compiles into a Object Identifier

SNMP can read and write Object Identifier values

TCP Extended Statistics MIB

A standards-track management information base for TCP statistics

- *draft-ietf-tsvwg-tcp-mib-extension*

Part of the Web100 project

- New variables propogate back through libweb100 library and /proc/web100 kernel patch
- Used within libweb100 for remote agents

TCP-ESTATS contents

A reasonably direct correspondence between libweb100 and *TCP-ESTATS-MIB*

- cid becomes tcpEStatsConnectionIndex
- Groups become tables

Add SNMP-specific data

- Conformance
- Control

Why use a ConnectionIndex?

SNMP presents data in a ascending order of MIB identifier

The identifier of a varaible in a table includes the index

So if we used ($zone$, src_addr, src_port, dst_addr, dst_port) as the index we would need to sort the data

- Maintain data structure to map kernel order into sorted order

- Kernels don't do a user-space callback when a conenction established

Why use a ConnectionIndex?

Worse still, after all this work most SNMP applications don't care about order of table rows

- They are doing a full table scan to collect statistics

Better to use the kernel identifier

- Kernel returns connections in this order
- libweb100 calls this cid
- *TCP-ESTATS-MIB* calls this tcpEStatsConnectionIndex

Maintaining the MIB

SNMP MIB is automatically built from the *Kernel Instrumentation Set* documentation

- Avoids typos
- Records current alignment of Web100 and TCP-ESTATS-MIB

Don't allow public to read these varaibles

Privacy

- tcpEStatsConnectRemAddress

Connection hijaaking

- tcpEStatsDataSndUna,
  tcpEStatsDataSndNxt,
  tcpEStatsDataSndMax,
  tcpEStatsDataSendInitial,
  tcpEStatsDataRcvNxt,
  tcpEStatsDataRecInitial

MIB tables are off by default

Easy way for vendors to ship *TCP-ESTATS* MIB in operating systems

Especially as many implementations ship allowing public to read all variables

MIB table off doesn't imply kernel stops counters

- This might actually cost more CPU

Applications

A graphical tool to analyse connection performance

- select a connection

Cmdline	PID	UID	Local addr	Port	Remote addr	Port	CID	State
ftp	4937	501	202.14.0.224	32801	192.42.62.2	21	62	ESTBLSH
	0	0	202.14.0.224	32824	192.42.62.2	60271	100	UNKNOWN
	0	0	127.0.0.1	32825	127.0.0.1	631	101	TMEWAIT
	0	0	127.0.0.1	631	127.0.0.1	32825	102	UNKNOWN
	0	0	202.14.0.224	32826	192.42.62.2	60274	103	UNKNOWN

Close   Refresh

File   View

IP address: 192.42.62.2: 60289   CID: 124

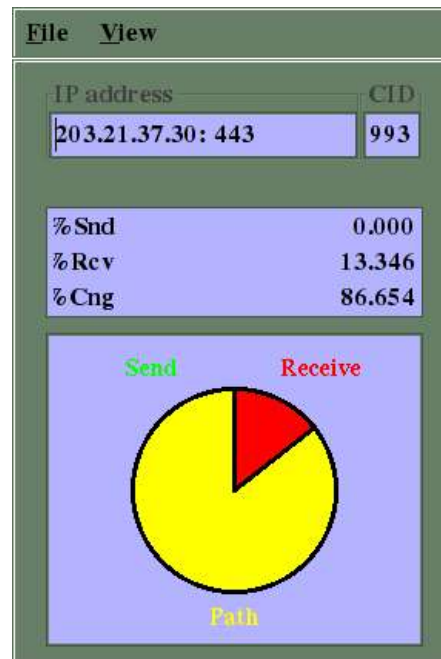| Variable name	Value	Delta
State	CloseWait	
SACKEnabled	3	
TimestampsEnabled	1	
NagleEnabled	1	
ECNEnabled	0	
SndWinScale	0	
RcvWinScale	7	
ActiveOpen	1	
MSSRcvd	0	
WinScaleRcvd	0	

- explore variable values

gutil (2 of 4)

- graph changes over time
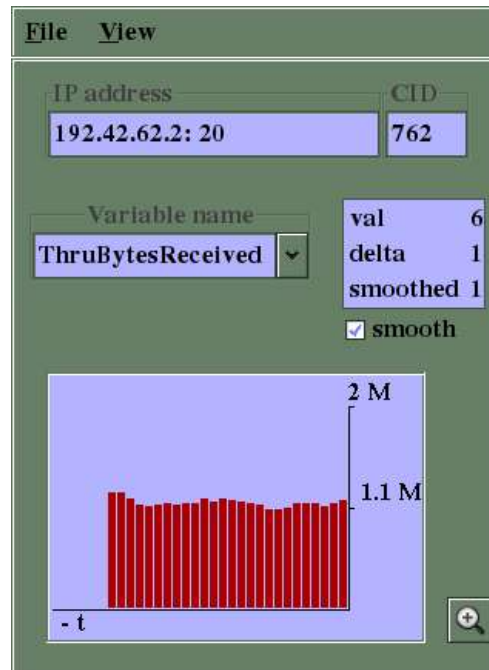
gutil (3 of 4)

- find bottlenecks
  - *triage*: is the bottleneck sender, receiver or path

- tune send and receive buffers

WAD

"word-around daemon"

Patches TCP stack variables to improve goodput

- Allows longer-term flow tuning than can be done by kernel

- Allows TCP's fairness to be defeated

  - Good for mission-oriented networks (DoE, NASA)

  - Bad for wide deployment

- In effect, move TCP policy to user space

WAD actions

Watch buffer usage and increase if this will improve throughput

On mission-centered networks

- Disable delayed Acks
- Faster slow-start
- Lessen multiplicative decrease

Implement alternative TCP policies

- Vegas
- Floyd's High Speed TCP

WAD is really user-space TCP policy

Coding TCP policy in the kernel is a technical mistake but probably socially necessary

webd

During and at close of socket, archive all Web100 variables

Allows a corpus of TCP performance to be built

Existing tools

Tools like *ttcp* and *iperf* report basic statistics

Extended to include Web100 variables

What we have learned about TCP performance

aarnet

Australia's Academic &
Research Network
www.aarnet.edu.au

Scaling TCP

Numbers which are reasonable at 10Mbps get large at 100Mbps and unreasonable at 1Gbps

Throughput (Mbps)	RTT between losses (sec)	Congestion window (packets)	Maximum packet loss (%)
1	5.5	8.3	0.02
10	55.5	83.3	0.0002
100	555.5	833.3	0.000002
1,000	5,555.5	8,333.3	0.00000002
10,000	55,555.5	83,333.3	0.0000000002

Quality of engineering
Link layer bit error rates

Bit error rates for correctly configured microwave and fiber are around 10^{-13}

PDH/SDH link layers will allow 10^{-6} before declaring the link down

Common misconfigurations result in bit error rates of 10^{-3} or more

- Synchronous link clocking

- Analogue power budgets

- Ethernet autoconfiguration not used

Quality of engineering Transfer to USA

Every link is good

$$rate \leqslant \frac{1460}{260} \times \frac{1}{\sqrt{10^{-11}}}$$

$rate \leq$ 1,776Mbps

Let's say one of the links has a bit error rate of 10^{-6}

$$rate \leqslant \frac{1460}{260} \times \frac{1}{\sqrt{10^{-4}}}$$

$rate \leq$ 0.56Mbps

Quality of engineering Lessons

A single TCP session can't fill a AU-US 10Gbps link

TCP throughput is sensitive to the quality of network engineering

– Error rates of 0.001% are unacceptable

– But lots of real life links have 3% error rates, fortunately the longer the link the (usually) better the engineering

– A new design criteria for long-haul links

TCP buffers

The "current congestion window" is the amount of data the operating system transmits

The optimal *buffer* size is 2 × *bandwidth* × *delay*

- which is *bandwidth* × *RTT*

Take a fast ethernet (100Mbps) machine on a GbE WAN to Sydney (50ms)

$buffer$ = (100,000÷8) × 0.050 = 625KB

A GbE-connected host needs 6.25MB!

Congestion window and path loss

TCP congestion window $\qquad cwnd \approx \dfrac{1.2}{\sqrt{p}}$

10Gbps, ethernet MTU, 100ms RTT

- – 83,333 segments in flight
- – To retain performance, at most one congestion event per 5,000m packets (or 1h40m)

Even at 1Gbps the congestion window makes sustained use of gigabit wireless impossible

Hardware for 100Mbps

Modern computers have no design bottlenecks at 100Mbps

Lots of bottlenecks at 1Gbps

- Lucky to get 690Mbps

Hardware for 1Gbps

Hardware can help

- Simply more memory, starting at 10GB

- Bus bandwidth

- TCP checksumming in NIC

- Segmentation offload (SOE)

  - Pass NIC TCP template header and 64KB data

- Interrupt combining

- Page alignment on receive

Quality of device driver counts a lot

- Intel e1000 versus Broadcom 5700

Consequnces for Web100

Not interested in Gbps rates

- Protocols have problems doing 0.1Gbps
- Solution space is different

Empower a user to discover common network engineering faults

– High bit error rates on WAN links

- Clocking
- Analogue power budget

– Ethernet autonegotiation misconfiguration

Automatically set buffer sizes

The end

Glen Turner
glen.turner@aarnet.edu.au

aarnet
Australia's Academic &
Research Network
www.aarnet.edu.au

Part 3
Ethernet misconfiguration

LinuxSA
Adelaide
2003-09-16

Glen Turner
glen.turner@aarnet.edu.au

aarnet
Australia's Academic &
Research Network
www.aarnet.edu.au

Ethernet auto-negotiation

aarnet

Australia's Academic &
Research Network
www.aarnet.edu.au

Issue: Ethernet auto-negotiation

Do one of

- Set Speed and Duplex at both ends of ethernet link
- Set auto-negotiation at both ends of ethernet link

Issue: Ethernet auto-negotiation

Do not

- Disable auto-negotiation, setting Speed and Duplex at one end of link and expect auto-negotiation to force those settings upn the remote port

Example

- A-end config: Auto=off, Speed=100, Duplex=full

- B-end config: Auto=on

- B-end actual: Speed=100 (from clock on Rx), Duplex=half

Issue: Ethernet auto-negotiation

Link layer result

- – B-end has packet
- – Waits for A-end to finish Tx
- – Sends packet
- – But A-end doesn't listen for collisions and sends packet
- – B-end detects collision, signals "late collision"
- – B-end requeues packet, increments 16-stage exponential backoff

Issue: Ethernet auto-negotiation

IP result

- Ethernet layer eventually signals "delay drop" and drops packet
- IP queues may have backlogged and packets dropped

TCP result

- Hosts using this link face a varying round-trip time, can't form good RTT estimate
- Packets drops signal congestion
- Performance dies as TCP rarely leaves congestion avoidance mode

Part 4
Linux configuration in detail

LinuxSA
Adelaide
2003-09-16

aarnet

Australia's Academic &
Research Network
www.aarnet.edu.au

Glen Turner
glen.turner@aarnet.edu.au

Setting buffer size

aarnet
Australia's Academic &
Research Network
www.aarnet.edu.au

TCP buffers
Setting buffer

Within an application

```
int buffer_size = 128 * 1024;

setsockopt(s, SOL_SOCKET, SO_SNDBUF,
    &buffer_size);

setsockopt(s, SOL_SOCKET, SO_RCVBUF,
    &buffer_size);
```

This can fail, as the operating system has limits to the size of buffer space

TCP buffers
Setting buffer in Linux

Linux uses 64KB by default

- $64 = (1,000,000 \div 8) \times RTT$

- Fast ethernet users should worry when $RTT > 5$ms

- GbE users worry when $RTT > 0.5$ms

- Obviously room for considerable improvement

TCP buffers
Setting buffer in Linux

Increase buffer size

- net.core.rmem_max = 8388608
  net.core.wmem_max = 8388608
  net.core.rmem_default = 65536
  net.core.wmem_default = 65536

Increase autotuning range

- net.ipv4.tcp_rmem = 4096 87380 8388608
  net.ipv4.tcp_wmem = 4096 65536 8388608
  net.ipv4.tcp_mem = 8388608 8388608 8388608
  net.ipv4.tcp_rmem = 4096 87380 8388608

TCP buffers
Interfaces also have buffering

Increase transmit queue

- `ifconfig eth0 txqueuelen 1000`

Increase receive queue

- `net.core.netdev_max_backlog=2500`