SWINBURNE UNIVERSITY OF TECHNOLOGY

Improving the FreeBSD TCP Implementation

An update on all things TCP in FreeBSD and how they affect you

Lawrence Stewart

lastewart@swin.edu.au

Centre for Advanced Internet Architectures (CAIA)
Swinburne University of Technology

Outline

1. Who is this guy?
2. TCP Recap
3. Modular congestion control
4. Deterministic Packet Discard
5. The ETCP Project
6. Wrapping Up

Detailed outline (section 1 of 6)

1 Who is this guy?

Who is this guy (and who let him past security)?

- BEng (Telecomms and Internet Technologies) 1st class honours / BSci (Comp Sci and Software Eng) (2001-2006)
- Centre for Advanced Internet Architectures, Swinburne University (2003-2007)
  - Research assistant/engineer during/after studies
  - `http://caia.swin.edu.au/`
- Currently a PhD candidate in telecomms eng at CAIA (2007-)
  - Main focus on transport protocols
  - `http://caia.swin.edu.au/cv/lstewart/`
- FreeBSD user since 2003, developer since 2008
  - Experimental research, software development, home networking, servers and personal desktops

Detailed outline (section 2 of 6)

1 Who is this guy?

2 TCP Recap

3 Modular congestion control

4 Deterministic Packet Discard

5 The ETCP Project

6 Wrapping Up

2 TCP Recap

- Jargon
- Key Facts
- Where are we today
- Open issues

TCP jargon

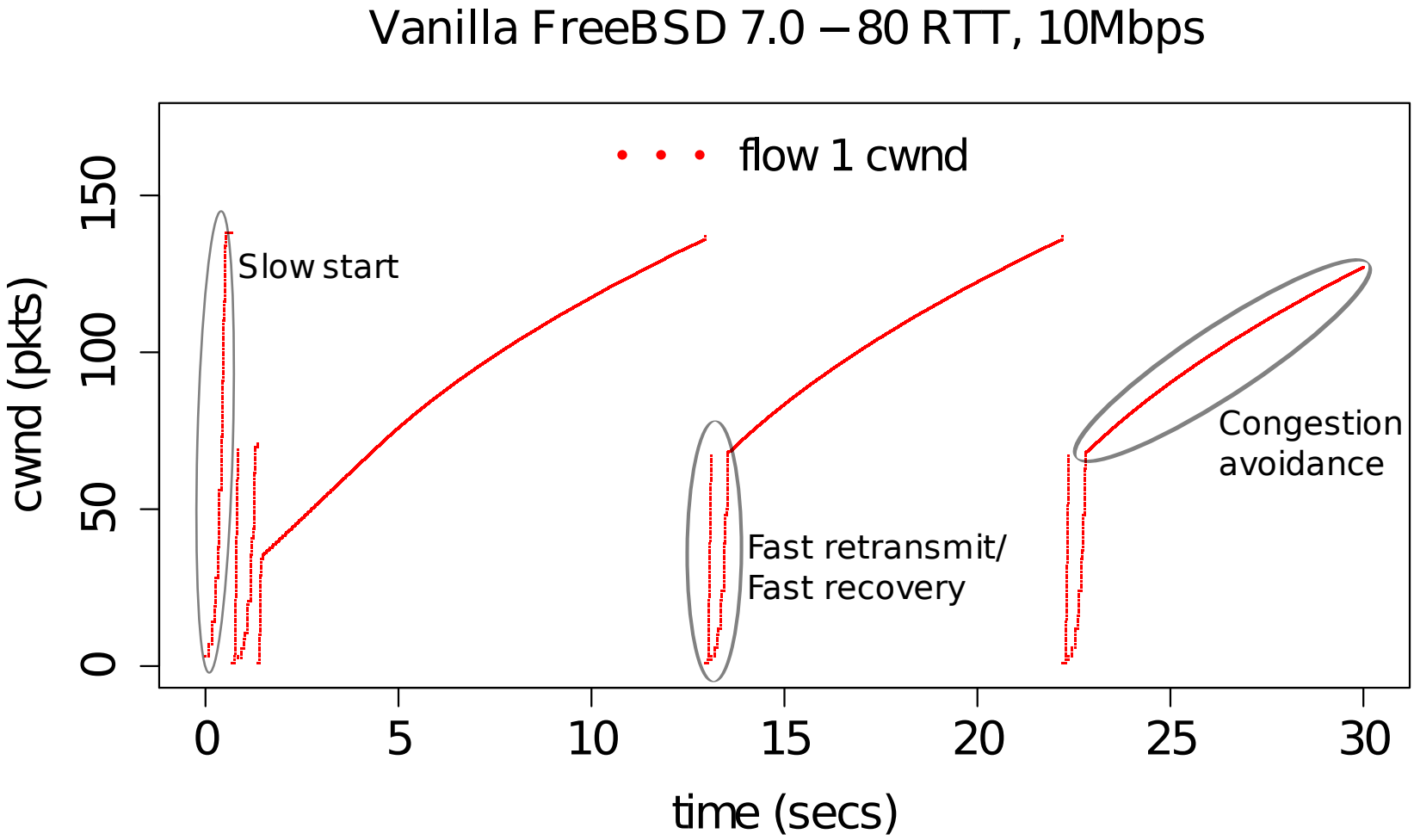
cwnd	congestion window
MSS	maximum segment size
ssthresh	slow start threshold
ACK	TCP acknowledgment
RTT	round trip time
BDP	bandwidth-delay product
RFC	request for comment
CC	congestion control
tcpcb	TCP control block
RTO	Retransmit timeout

Key Facts

- Core TCP modes of operation [1]
  - Slow start
  - Congestion avoidance
  - Fast retransmit
  - Fast recovery
- Many protocol tweaks and additions along the way
  - SACK, ABC, ECN, window scaling, timestamps, etc.
- RFC 4614 provides a good summary of TCP related RFCs

[1] See RFC2001

Key Facts


Vanilla FreeBSD 7.0 −80 RTT, 10Mbps
flow 1 cwnd
Slow start
Fast retransmit/
Fast recovery
Congestion
avoidance
cwnd (pkts)
0
50
100
150
0
5
10
15
20
25
30
time (secs)

Where are we today

- Many incremental (partially implemented) improvements
- State of the CC union
  - NewReno is defacto standard with warts (LFN, wireless)
  - Many new proposals
  - BSD still uses NewReno
  - Linux uses CUBIC
  - Windows Vista uses CTCP
- TCP/IP stack enhancements e.g.
  - CSO/TSO/LRO/TOE
  - Various locking/caching tricks
  - Socket buffer autotuning

Open issues

- High-speed CC algorithms [2]
  - FAST, HS-TCP, H-TCP, CTCP, CUBIC, etc.
- Delay based CC algorithms
- How do we compare and evaluate TCPs?
- CSO/TSO/LRO/TOE obscure behaviours
- Testing/verification of TCP/IP stack behaviour

[2] Nice summary:
http://kb.pert.geant2.net/PERTKB/TcpHighSpeedVariants

Detailed outline (section 3 of 6)

1 Who is this guy?

2 TCP Recap

3 Modular congestion control

4 Deterministic Packet Discard

5 The ETCP Project

6 Wrapping Up

3 Modular congestion control

- Motivation
- KPI/API/Configuration
- Case studies: H-TCP and CUBIC
- Usage
- TCP Testbed
- A Few Results

Motivation

- Facilitates:
  - TCP CC research
  - Standardisation process
- Catering to specialised applications
  - Select most appropriate CC algorithm for the task
- Ultimately a better Internet (hopefully!)

KPI/API/Configuration

- Defined in `<netinet/cc.h>`

```
/* specify one of these structs per CC algorithm */
struct cc_algo {
char name[TCP_CA_NAME_MAX];
int (*init) (struct tcpcb *tp);
void (*deinit) (struct tcpcb *tp);
void (*cwnd_init) (struct tcpcb *tp);
void (*ack_received) (struct tcpcb *tp, struct tcphdr *th);
void (*pre_fr) (struct tcpcb *tp, struct tcphdr *th);
void (*post_fr) (struct tcpcb *tp, struct tcphdr *th);
void (*after_idle) (struct tcpcb *tp);
void (*after_timeout) (struct tcpcb *tp);
STAILQ_ENTRY(cc_algo) entries;
};
```

KPI/API/Configuration

- Housekeeping

```
/* called during TCP/IP stack initialisation on boot */
void cc_init(void);

/* dynamically registers a new CC algorithm */
int cc_register_algorithm(struct cc_algo *);

/* dynamically deregisters a CC algorithm */
int cc_deregister_algorithm(struct cc_algo *);
```

KPI/API/Configuration

- Minor ABI-breaking additions to struct tcpcb

```
struct tcpcb {
    ....

/* CC function pointers to use for this connection */
struct cc_algo *cc_algo;

/* connection specific CC algorithm data */
void    *cc_data;
};
```

KPI/API/Configuration

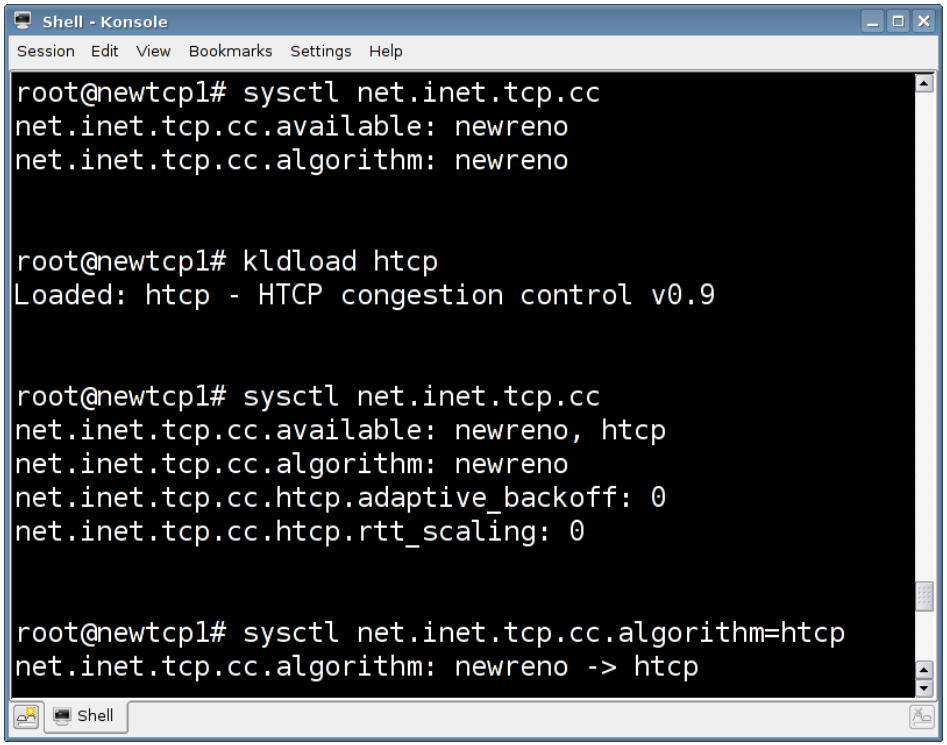
- New `net.inet.tcp.cc` sysctl tree with variables:
  - `available`: comma-separated list of available CC algorithms
  - `algorithm`: current system default CC algorithm
- Removed `net.inet.tcp.newreno` sysctl variable
- New socket option `TCP_CONGESTION` defined in `tcp.h`
  - Override system default CC algorithm using setsockopt(2)
  - Same as Linux define e.g. Iperf -Z option works

Case studies: H-TCP and CUBIC

- High-speed TCP variants
- Implemented as FreeBSD kernel modules [3]
- H-TCP
  - 591 line C file
  - ~280 lines of actual source code of which:
  - ~100 lines is housekeeping/support code
  - ~180 lines is core H-TCP code
- CUBIC
  - 412 line C file, 200 line header file
  - ~300 lines of actual source code of which:
  - ~145 lines is housekeeping/support code
  - ~155 lines is core CUBIC code

[3] Available from: `http://caia.swin.edu.au/urp/newtcp/tools.html`

Usage


Shell - Konsole
Session Edit View Bookmarks Settings Help
root@newtcp1# sysctl net.inet.tcp.cc
net.inet.tcp.cc.available: newreno
net.inet.tcp.cc.algorithm: newreno
root@newtcp1# kldload htcp
Loaded: htcp - HTCP congestion control v0.9
root@newtcp1# sysctl net.inet.tcp.cc
net.inet.tcp.cc.available: newreno, htcp
net.inet.tcp.cc.algorithm: newreno
net.inet.tcp.cc.htcp.adaptive_backoff: 0
net.inet.tcp.cc.htcp.rtt_scaling: 0
root@newtcp1# sysctl net.inet.tcp.cc.algorithm=htcp
net.inet.tcp.cc.algorithm: newreno -> htcp
Shell

Usage

Shell - Konsole

Session Edit View Bookmarks Settings Help

```
root@newtcp1# sysctl net.inet.tcp.cc
net.inet.tcp.cc.available: newreno
net.inet.tcp.cc.algorithm: newreno


root@newtcp1# kldload htcp
Loaded: htcp - HTCP congestion control v0.9


root@newtcp1# sysctl net.inet.tcp.cc
net.inet.tcp.cc.available: newreno, htcp
net.inet.tcp.cc.algorithm: newreno
net.inet.tcp.cc.htcp.adaptive_backoff: 0
net.inet.tcp.cc.htcp.rtt_scaling: 0


root@newtcp1# sysctl net.inet.tcp.cc.algorithm=htcp
net.inet.tcp.cc.algorithm: newreno -> htcp
```

Shell

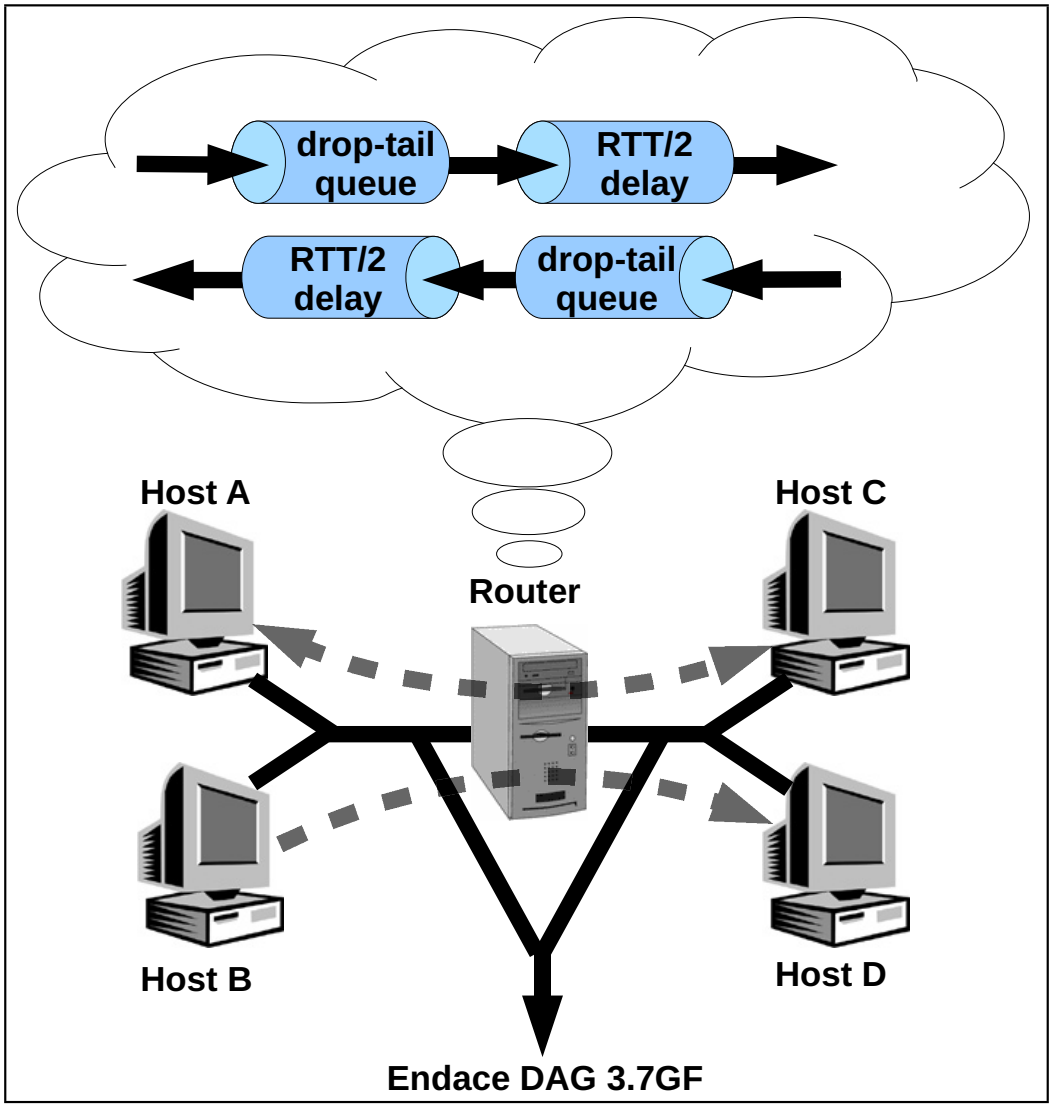
Shell - Konsole

Session Edit View Bookmarks Settings Help

```
root@newtcp1# sysctl net.inet.tcp.cc
net.inet.tcp.cc.available: htcp, newreno
net.inet.tcp.cc.algorithm: htcp
net.inet.tcp.cc.htcp.adaptive_backoff: 0
net.inet.tcp.cc.htcp.rtt_scaling: 0


root@newtcp1# kldunload htcp
Unloaded: htcp - HTCP congestion control v0.9


root@newtcp1# sysctl net.inet.tcp.cc
net.inet.tcp.cc.available: newreno
net.inet.tcp.cc.algorithm: newreno


root@newtcp1#
```

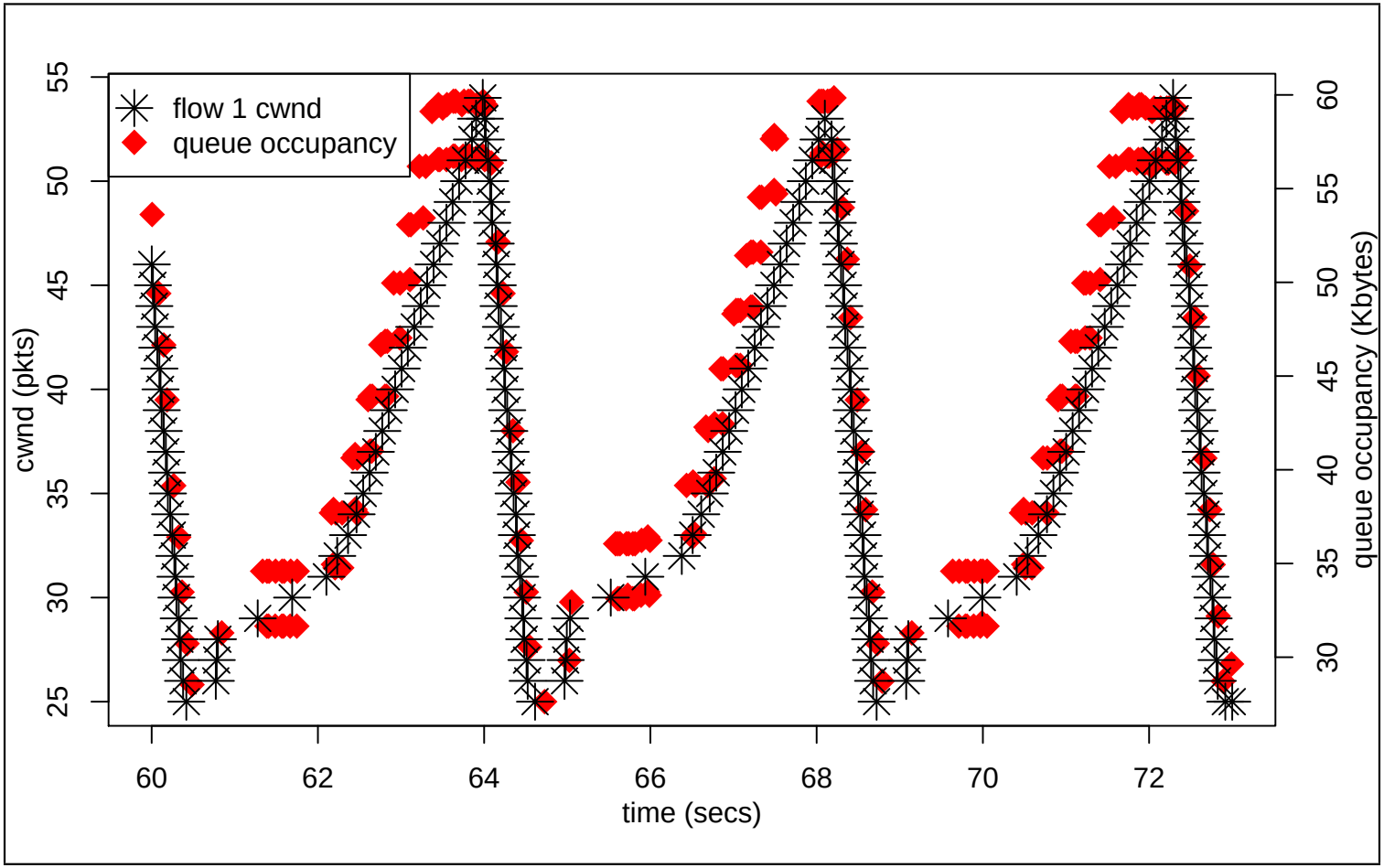
Shell

TCP Testbed


drop-tail queue
RTT/2 delay
RTT/2 delay
drop-tail queue
Host A
Host C
Router
Host B
Host D
Endace DAG 3.7GF

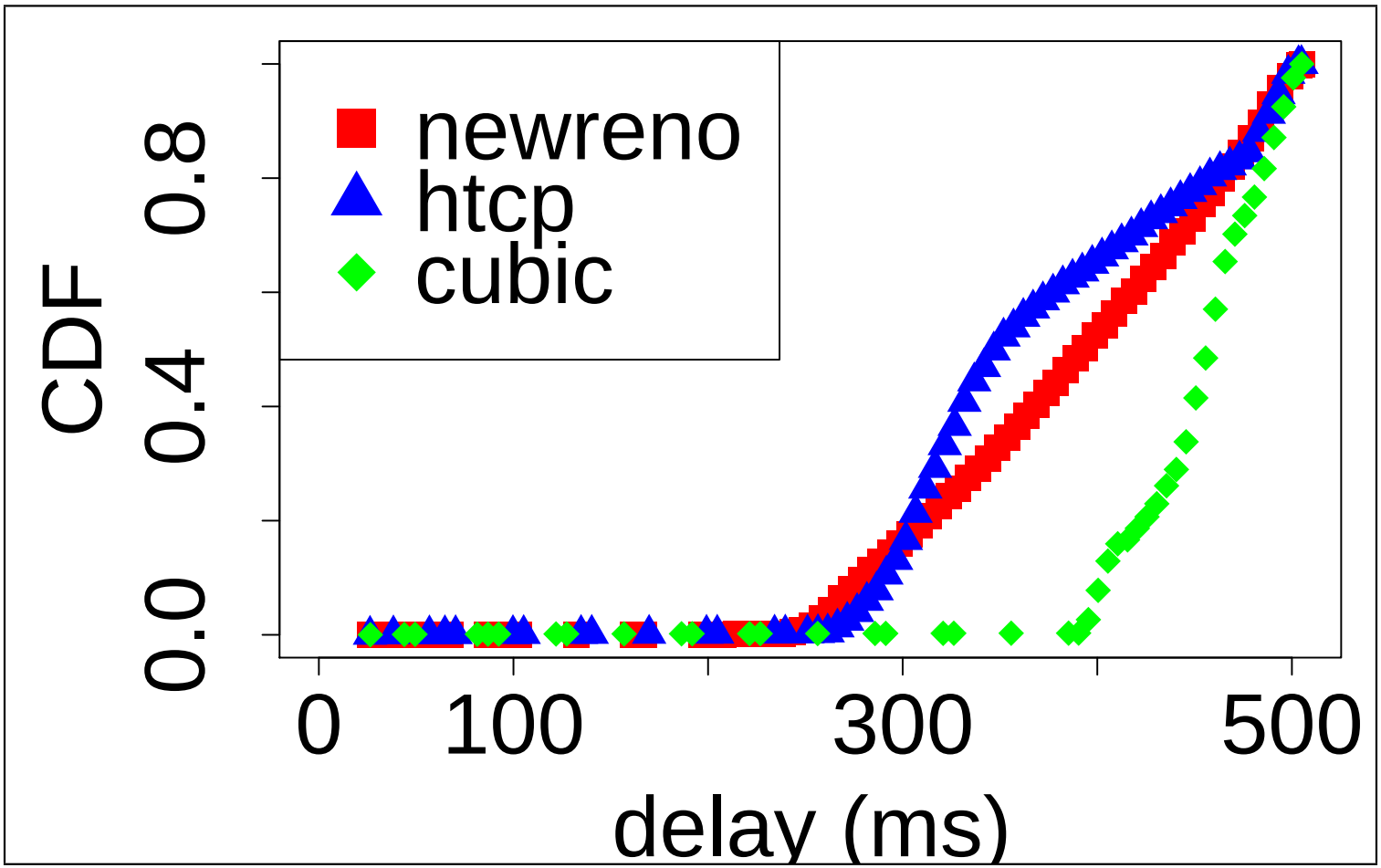
A Few Results

- 1 TCP flow, H-TCP, 100ms RTT, 1Mbps, 60000 byte queue



A Few Results

- Induced delay; 1 TCP flow, 50ms RTT, 1Mbps, 60000 byte queue



Detailed outline (section 4 of 6)

4 Deterministic Packet Discard

Deterministic Packet Discard (DPD)

- Patch against FreeBSD 8.x IPFW/Dummynet
- BSD licenced source [4]
- Useful for protocol (not just TCP!) verification and testing
- Adds 'pls' (packet loss set) option for dummynet pipes
- e.g. ipfw pipe 1 config pls 1,5-10,30 would drop packets 1, 5-10 inclusive and 30
- Need to catch up with Luigi's work
- Lower priority, but hope to commit to 7.x and 8.x soon

[4] Available from `http://caia.swin.edu.au/urp/newtcp/tools.html`

Detailed outline (section 5 of 6)

1 Who is this guy?

2 TCP Recap

3 Modular congestion control

4 Deterministic Packet Discard

5 The ETCP Project

6 Wrapping Up

5 The ETCP Project

- Project Recap
- SIFTR
- SIFTR demo
- Appropriate Byte Counting
- Reassembly Queue Autotuning

Project Recap

- Development project funded by FreeBSD Foundation
  - Implement TCP Appropriate Byte Counting
  - Implement TCP reassembly queue autotuning
  - Integrate SIFTR into FreeBSD
  - Characterise changes on our TCP testbed
- Should finish up by July 2009
- `http://caia.swin.edu.au/freebsd/etcp09/`
- `http://freebsdfoundation.org/`

SIFTR

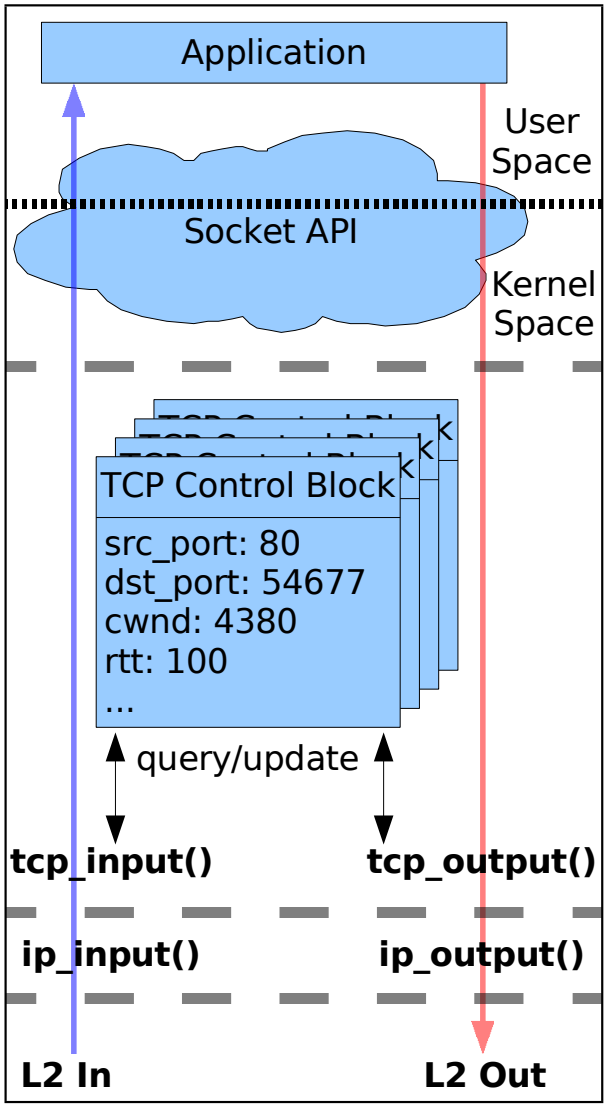
- Statistical Information For TCP Research
- FreeBSD [6,7,8] kernel module
- BSD licenced source [5]
- Similar base concept to Web100
- Event triggered (not poll based)
- Currently logs 25 different variables to file as CSV data [6]
- Plan to integrate into base system for 8.x
- Work on v1.2.x sponsored by the FreeBSD Foundation

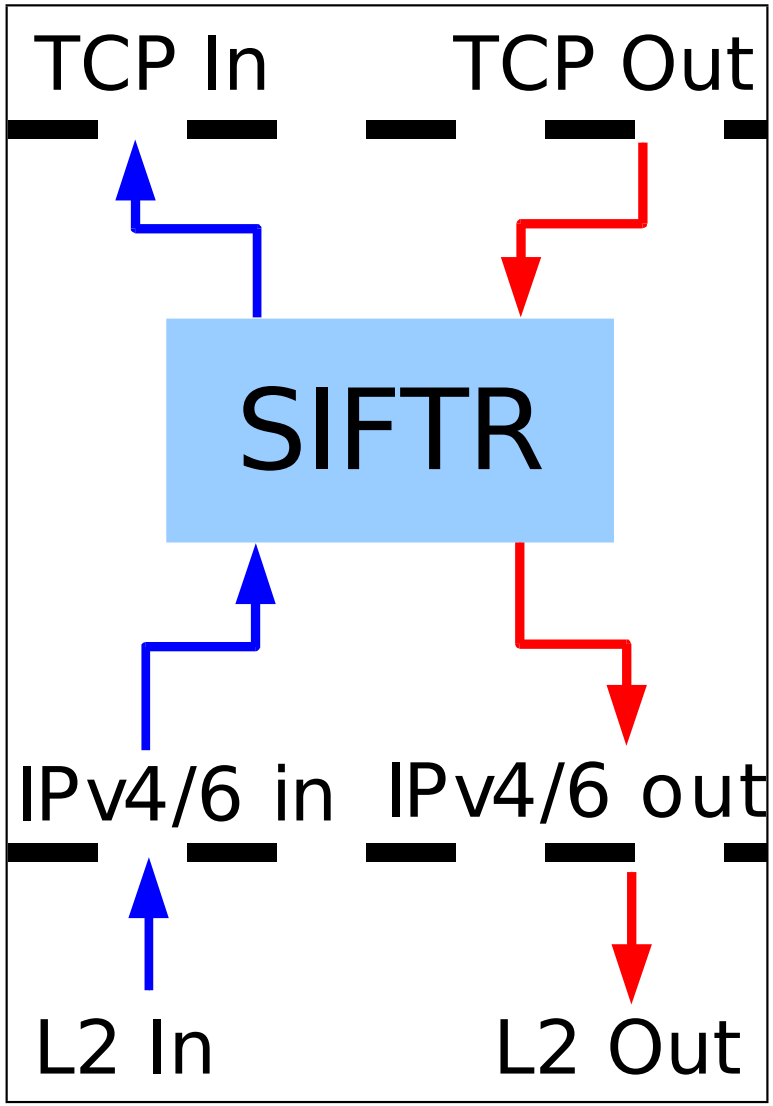
---

[5] Available from: `http://caia.swin.edu.au/urp/newtcp/tools.html`

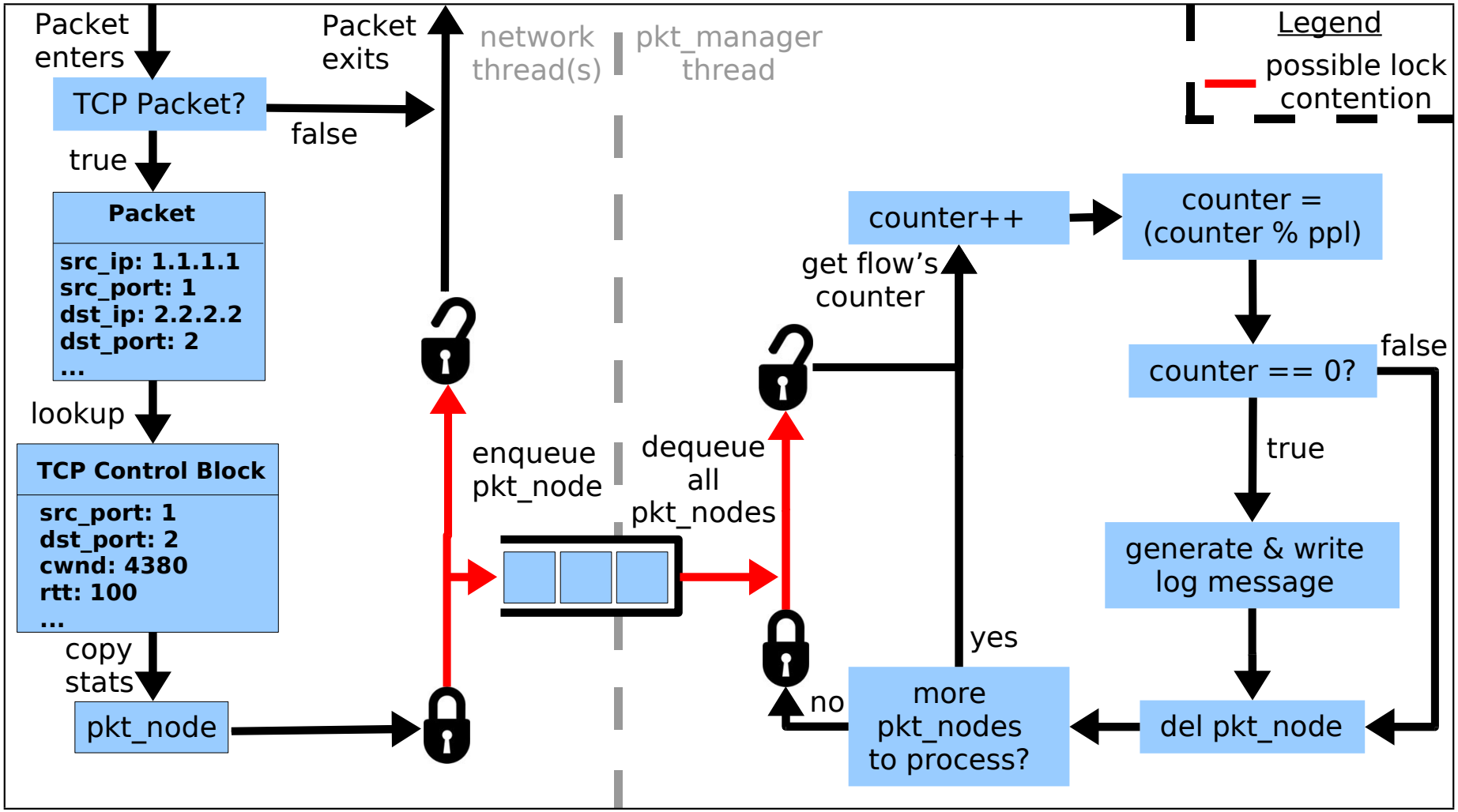
[6] See README in SIFTR distribution for specific details

SIFTR


Application
User Space
Socket API
Kernel Space
TCP Control Block
src_port: 80
dst_port: 54677
cwnd: 4380
rtt: 100
...
query/update
tcp_input()
tcp_output()
ip_input()
ip_output()
L2 In
L2 Out


TCP In
TCP Out
SIFTR
IPv4/6 in
IPv4/6 out
L2 In
L2 Out

SIFTR


Packet enters
TCP Packet?
true
false
Packet exits
Packet
src_ip: 1.1.1.1
src_port: 1
dst_ip: 2.2.2.2
dst_port: 2
...
lookup
TCP Control Block
src_port: 1
dst_port: 2
cwnd: 4380
rtt: 100
...
copy stats
pkt_node
network thread(s)
pkt_manager thread
enqueue pkt_node
dequeue all pkt_nodes
get flow's counter
counter++
counter = (counter % ppl)
counter == 0?
false
true
generate & write log message
del pkt_node
more pkt_nodes to process?
yes
no
Legend
possible lock contention

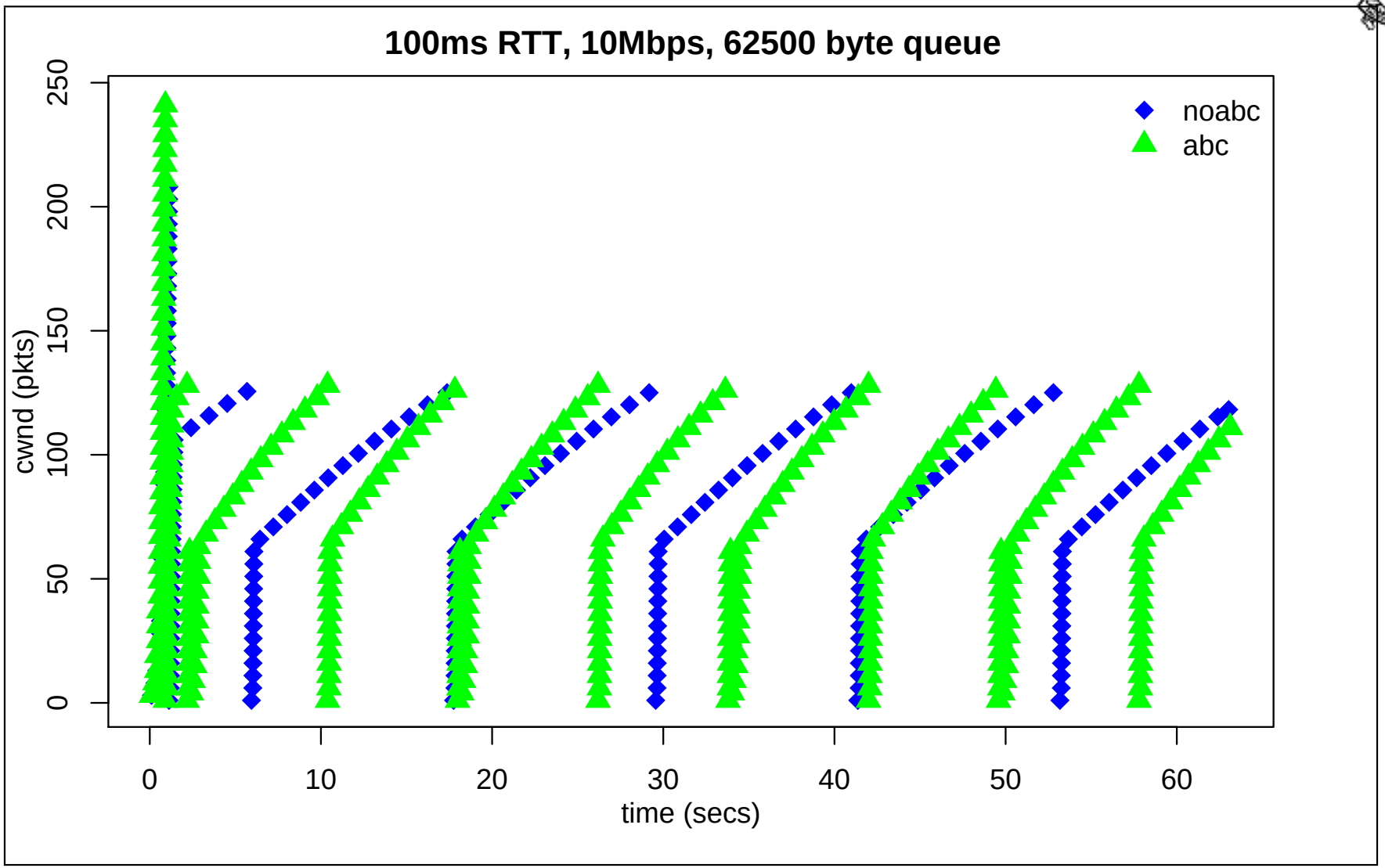
SIFTR demo

Let’s see what we can see!

Appropriate Byte Counting (ABC)

- Committed to FreeBSD 8.x as r187289
- Relatively straight forward patch
- Mostly a TCP bug fix
- Some interesting side effects...

Appropriate Byte Counting (ABC)



Reassembly Queue Autotuning

- TCP reassembly queue tuning is inherently connection specific
- Current method is wasteful and can severely damage TCP performance
- Aim to do away with net.inet.tcp.reass.maxqlen
- Adapt reassembly queue based on connection dynamics
- Somewhat akin to socket buffer auto tuning
- Currently WIP (building on Andre's work)
- Sponsored by the FreeBSD Foundation

Detailed outline (section 6 of 6)

1 Who is this guy?

2 TCP Recap

3 Modular congestion control

4 Deterministic Packet Discard

5 The ETCP Project

6 Wrapping Up

6 Wrapping Up

- Future work
- Additional reading
- Acknowledgements
- Questions

Ideas for future work

- TCP specific:
  - RTT estimator
  - Share CC between TCP/SCTP (Randall et. al.)
  - Comprehensive RFC compliance check
  - Fix slow-start, FR/FR
- TCP/IP stack in general:
  - Framework for dealing with CSO/TSO/LRO/TOE
  - DTRACEesque instrumentation
  - Testing framework <- next big project I want to tackle

Further Information

- http://caia.swin.edu.au/urp/newtcp/
- http://caia.swin.edu.au/freebsd/etcp09/
- http://people.freebsd.org/~lstewart/
- http://lists.freebsd.org/pipermail/freebsd-net/

Acknowledgements

- The FreeBSD Foundation

- Cisco Systems

- Dan Langille, et. al.
- FreeBSD community

Fin

Questions?