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?
2. TCP Recap
3. Modular congestion control
4. Deterministic Packet Discard
5. The ETCP Project
6. Wrapping Up
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/](http://caia.swin.edu.au/)
- Currently a PhD candidate in telecomms eng at CAIA (2007-)
  - Main focus on transport protocols
- FreeBSD user since 2003, developer since 2008
  - Experimental research, software development, home networking, servers and personal desktops
1. Who is this guy?

2. TCP Recap
   - Jargon
   - Key Facts
   - Where are we today
   - Open issues

3. Modular congestion control

4. Deterministic Packet Discard

5. The ETCP Project

6. Wrapping Up
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
  - 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

¹See RFC2001
Key Facts

Vanilla FreeBSD 7.0 - 80 RTT, 10Mbps

- Slow start
- Fast retransmit/
  Fast recovery
- Congestion avoidance

Flow 1 cwnd

 BSDCan 2009

http://www.caia.swin.edu.au
lastewart@swin.edu.au
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
  - 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

²Nice 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
   - Motivation
   - KPI/API/Configuration
   - Case studies: H-TCP and CUBIC
   - Usage
   - TCP Testbed
   - A Few Results
4. Deterministic Packet Discard
5. The ETCP Project
6. Wrapping Up
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 *);
Minor ABI-breaking additions to struct tcpcb

```c
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
  - 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

Usage

```
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
```
Usage

```
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

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
```

 BSDCan 2009

 http://www.caia.swin.edu.au

 lastewart@swin.edu.au
TCP Testbed

Host A -> drop-tail queue -> RTT/2 delay -> drop-tail queue -> RTT/2 delay

Host A -> Router

Host C

Host B

Host D

Endace DAG 3.7GF

BSDCan 2009
http://www.caia.swin.edu.au
lastewart@swin.edu.au
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

![Graph showing CDF for newreno, htcp, and cubic](image-url)

- BSDCan 2009
  - http://www.caia.swin.edu.au
  - lastewart@swin.edu.au
1. Who is this guy?

2. TCP Recap

3. Modular congestion control

4. Deterministic Packet Discard

5. The ETCP Project

6. Wrapping Up
Deterministic Packet Discard (DPD)

- Patch against FreeBSD 8.x IPFW/Dummynet
- BSD licenced source
- 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

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
   - Project Recap
   - SIFTR
   - SIFTR demo
   - Appropriate Byte Counting
   - Reassembly Queue
   - Autotuning
6. Wrapping Up
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://freebsdfoundation.org/
SIFTR

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

6 See README in SIFTR distribution for specific details
SIFTR

TCP In | TCP Out

IPv4/6 in IPv4/6 out

L2 In | L2 Out

Application

Socket API

User Space

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()
Packet enters

TCP Packet?

true

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

Packet exits

network thread(s)

pkt_manager thread

counter++

counter = (counter % ppl)

counter == 0?

generate & write
log message

thread(s)

Packet enters

Packet exits

possible lock contention

Legend

thread

thread(s)

Packet enters

Packet exits

counter++

counter = (counter % ppl)

counter == 0?

generate & write
log message

Packet enters

Packet exits

possible lock contention

Legend

false

true
Let’s see what we can see!
Appropriate Byte Counting (ABC)

- Committed to FreeBSD 8.x as r187289
- Relatively straightforward patch
- Mostly a TCP bug fix
- Some interesting side effects...
Appropriate Byte Counting (ABC)

100ms RTT, 10Mbps, 62500 byte queue

- noabc
- 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
1 Who is this guy?
2 TCP Recap
3 Modular congestion control
4 Deterministic Packet Discard
5 The ETCP Project
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://people.freebsd.org/~lstewart/
- http://lists.freebsd.org/pipermail/freebsd-net/
Acknowledgements

- The FreeBSD Foundation
- Dan Langille, et. al.
- FreeBSD community
- Cisco Systems

BSDCan 2009  http://www.caia.swin.edu.au  lastewart@swin.edu.au
Questions?