

Baseline single-flow TCP results for TEACUP v0.4.5 testbed

Grenville Armitage

Centre for Advanced Internet Architectures, Technical Report 140502A

Swinburne University of Technology

Melbourne, Australia

garmitage@swin.edu.au

Abstract—This technical report summarises a basic set of single-flow TCP experiments designed to confirm correct operation of our TEACUP testbed under teacup-v0.4.5 using pfifo (tail-drop) queue management at the bottleneck router. We document plausibly correct behaviour of FreeBSD’s NewReno, Cubic and CDG, Linux’s NewReno and CUBIC and Windows 7’s default TCP. We do not attempt to make any meaningful comparisons between the tested TCP algorithms.

Index Terms—TCP, experiments, testbed, TEACUP

I. INTRODUCTION¹

CAIA has developed TEACUP² [1] to support a comprehensive experimental evaluation of different TCP congestion control algorithms. Our actual testbed is described in [2]. This report summarises preliminary testbed trials confirming expected single-flow TCP behaviour under TEACUP v0.4.5.

We ran NewReno and CUBIC and CDG (v0.1) trials between two FreeBSD hosts, NewReno and CUBIC trials between two Linux hosts, and a trials between two Windows 7 hosts running their default TCP. The bottleneck in each case is a Linux machine using netem and tc to provide independently configurable bottleneck rate limits and artificial one-way delay (OWD).

The rest of the report is organised as follows. Section II summarises the the testbed topology and physical configuration for these trials. Section III summarises the single-flow behaviour versus time at 2Mbps, while Section IV summarises the single-flow behaviour versus time at 10Mbps. Section V presents the impact of base RTT and buffer size on throughput and induced RTT. Section VI concludes and outlines future work.

¹Erratum: Online copy updated July 9th 2014 to clarify use of Windows 7’s default TCP.

²TCP Experiment Automation Controlled Using Python.

II. TESTBED TOPOLOGY AND TEST CONDITIONS

Each trial is a single TCP connection pushing data in one direction for 90 seconds over a path with different emulated delays and bottleneck speeds. This section documents the testbed topology and tested combinations of operating systems, TCP algorithms and path conditions.

A. Hosts and router

Figure 1 (from [2]) shows a logical picture of the testbed’s networks.³ The router provides a configurable bottleneck between three hosts on network 172.16.10.0/24 and three hosts on network 172.16.11.0/24. Each host is a triple-boot machine that can run 64-bit Linux (openSUSE 12.3 with kernel 3.9.8 and web10g patch [3]), 64-bit FreeBSD (FreeBSD 9.2-RELEASE #0 r255898) or 64-bit Windows 7 (with Cygwin 1.7.25 for unix-like control of the host).

For all experiments the bottleneck router runs 64-bit Linux (openSUSE 12.3 with kernel 3.10.18 patched to run at 10000 Hz). We used Host 2 (172.16.10.3) as the data source and Host 4 (172.16.11.2) as the data sink. See [2] for more technical details of the testbed.

B. Operating System and TCP combinations

The experiments cover three different operating systems, and four different TCP algorithms.

- FreeBSD: Newreno⁴, CUBIC⁵ and CDG⁶ (v0.1)
- Linux: Newreno and CUBIC

³Each network is a switched Gigabit Ethernet VLAN on a common managed switch.

⁴http://www.freebsd.org/cgi/man.cgi?query=cc_newreno

⁵http://www.freebsd.org/cgi/man.cgi?query=cc_cubic

⁶http://www.freebsd.org/cgi/man.cgi?query=cc_cdg

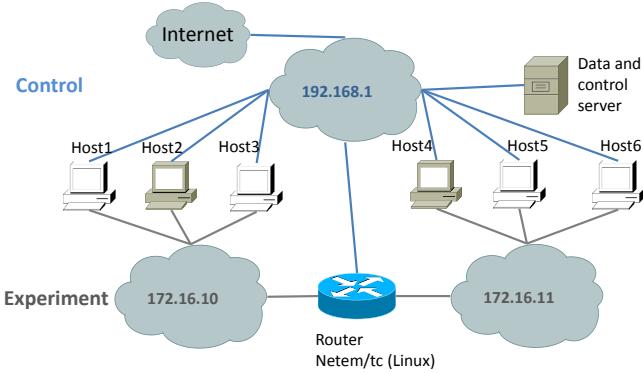


Figure 1. Testbed overview

- Windows 7: default TCP

See Appendices A, B and C for details of the TCP stack configuration parameters used for FreeBSD, Linux and Windows 7 respectively.

C. Emulated path conditions

The bottleneck router uses `netem` and `tc` to concatenate an emulated path with specific one way delay (OWD) and an emulated bottleneck whose throughput is limited to a particular rate.⁷

We emulated the following path conditions:

- 0% intrinsic loss rate
- OWD: 0, 5, 10, 20, 40, 80 and 100ms

The intrinsic loss rate simply means we did not add any additional packet loss above what was induced by congestion of the bottleneck's buffer. The OWD is symmetric, so baseline RTT is always 2*OWD.

We emulated the following bottleneck conditions:

- Bottleneck bandwidths: 2, 6 and 10Mbps
- Bottleneck buffer sizes: 50, 90 and 180 pkts
- Bottleneck AQM: pfifo

The bottleneck implements a buffer (queue) of finite size in packets, and uses simple (pfifo) tail-drop to manage buffer (queue) overflow.⁸

⁷Packets sit in a 1000-packet buffer while being delayed, then in configurable “bottleneck buffer” when being rate-shaped.

⁸Future experiments will introduce and compare the behaviour of FQ_CODEL and PIE.

D. Traffic generator and logging

Each 90-second TCP flow was generated using iperf 2.0.5 [4] on all three OSes, patched to enable correct control of the send and receive buffer sizes [5]. For each flow, iperf requests 600Kbyte socket buffers to ensure cwnd growth was not artificially limited by the maximum receive window.

TCP connection statistics were logged using SIFTR [6] under FreeBSD, Web10g [3] under Linux and TCP EStats under Windows.

Packets captured at both hosts with `tcpdump` were later used to calculate end to end RTT estimates using CAIA's passive RTT estimator, SPP [7], [8].

III. RTT AND CWND VERSUS TIME AT 2MBPS

First we look at the behaviour of RTT and cwnd versus time with a 2Mbps bottleneck rate limit. This allows us to confirm and compare the dynamic behaviours of each TCP implementation in a single-queue environment and observe the impact of bottleneck buffer size.

For conciseness, we show results under the following conditions as illustrative examples – a 40ms base RTT and two bottleneck buffer sizes (50 and 90 packets long). Note that at 2Mbps the full-size 1500 byte packet corresponds to 6 ms. Consequently, given the packet-oriented bottleneck buffer, the congestion-induced component of RTT will jump in multiples of 6 ms.⁹

A. Newreno – FreeBSD and Linux

Figures 2 and 3 show the behaviour of cwnd and RTT over time for for NewReno implementations under FreeBSD and Linux respectively where the bottleneck buffer is 50 packets long. Figures 4 and 5 show the same for a bottleneck buffer that is 90 packets long.

NewReno shows the classic slow start (SS) followed by cyclical movement of congestion avoidance (CA) and fast recovery/fast retransmit (FR). A key difference is visible – FreeBSD overloads (reuses) cwnd during FR (and the rapid drop and re-growth is reflected in the graphs) whereas Linux uses a separate internal variable to track window size during FR (so the Linux cwnd graphs reflect only CA behaviour).

⁹Furthermore, cwnd grows by two packets for every ACK. So as the source emits two new packets back-to-back we can observe jumps of 12 ms (the two newly-queued packets).

Bottleneck buffer size influences the periodicity of both cwnd cycles and peak RTT. The larger bottleneck buffer takes longer to fill, and when full it is ‘longer’ in milliseconds (of queued up packets waiting to be forwarded at 2Mbps).

Linux NewReno cycles half as fast as FreeBSD’s NewReno under the same conditions because the 3.9 Linux kernel defaults to growing cwnd by one for each ACK despite delayed-ACKs being enabled.¹⁰ (The FreeBSD 9.2-RELEASE kernel uses “appropriate byte counting” to compensate for the less frequent ACK arrivals caused by delayed-ACKs.)

B. CUBIC – FreeBSD and Linux

Figures 6 and 7 show the behaviour of cwnd and RTT over time for CUBIC implementations under FreeBSD and Linux respectively where the bottleneck buffer is 50 packets long. Figures 8 and 9 show the same for a bottleneck buffer that is 90 packets long.

CUBIC shows the classic slow start followed by cyclical movement between fast recovery/fast retransmit and CUBIC’s congestion avoidance modes.

C. Windows 7

Figures 10 and 11 show the behaviour of cwnd and RTT over time for Windows 7’s default TCP implementation where the bottleneck buffer is 50 and 90 packets long respectively. The behaviour is as expected, and similar to FreeBSD’s NewReno under these simplified circumstances.

D. CDG – FreeBSD

Figures 12 and 13 show the behaviour of cwnd and RTT over time for FreeBSD’s CDG implementation where the bottleneck buffer is 50 and 90 packets long respectively. CDG shows the expected noisy variation in cwnd around a fairly low absolute value. The resulting absolute RTTs are quite low (making the queuing-related quantisation of RTT values also very distinct).

¹⁰This behaviour is mostly of academic interest, since modern Linux kernels almost always use CUBIC in preference to NewReno.

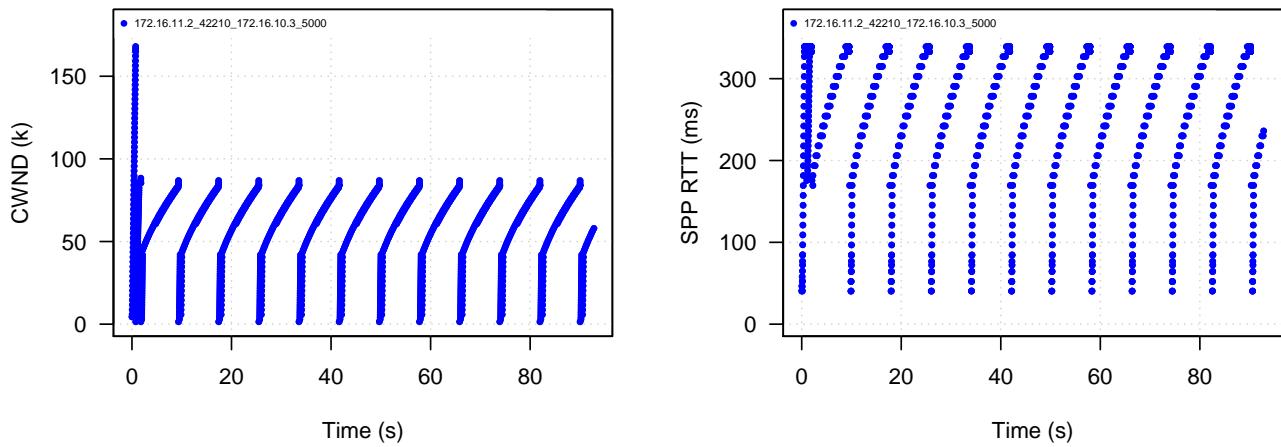


Figure 2. FreeBSD NewReno: 2Mbps bottleneck rate, 40ms base RTT, 50 packet buffer

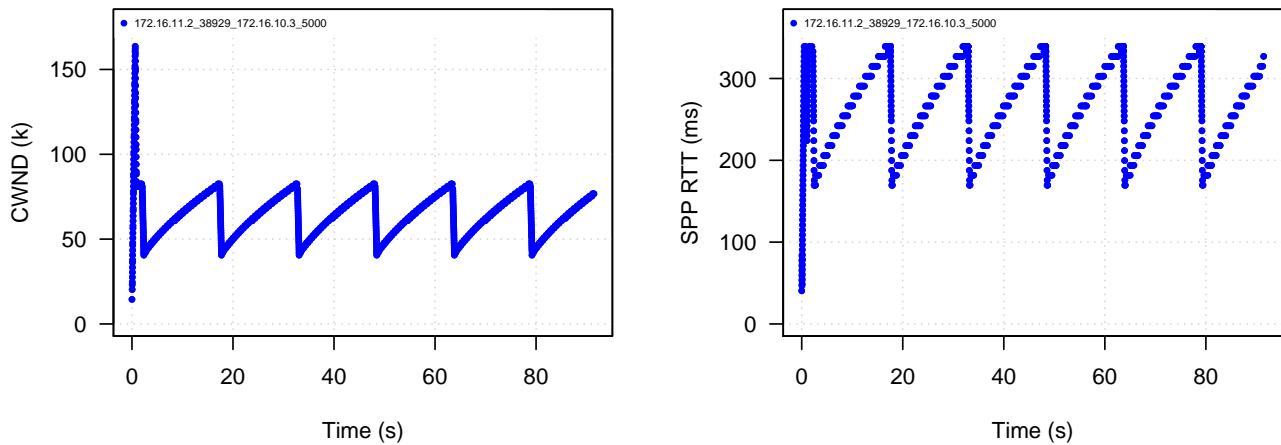


Figure 3. Linux NewReno: 2Mbps bottleneck rate, 40ms base RTT, 50 packet buffer

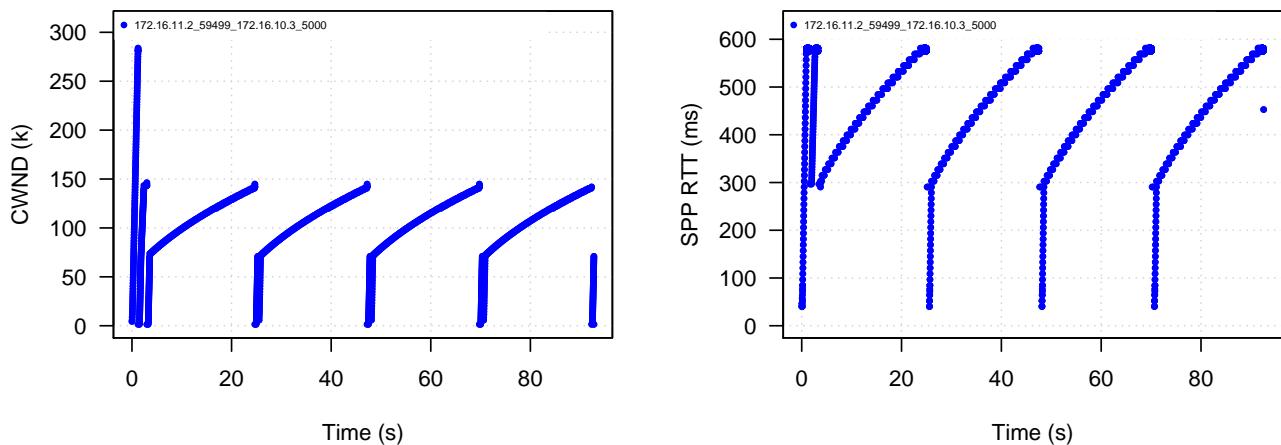


Figure 4. FreeBSD NewReno: 2Mbps, 40ms base RTT, 90 packet buffer

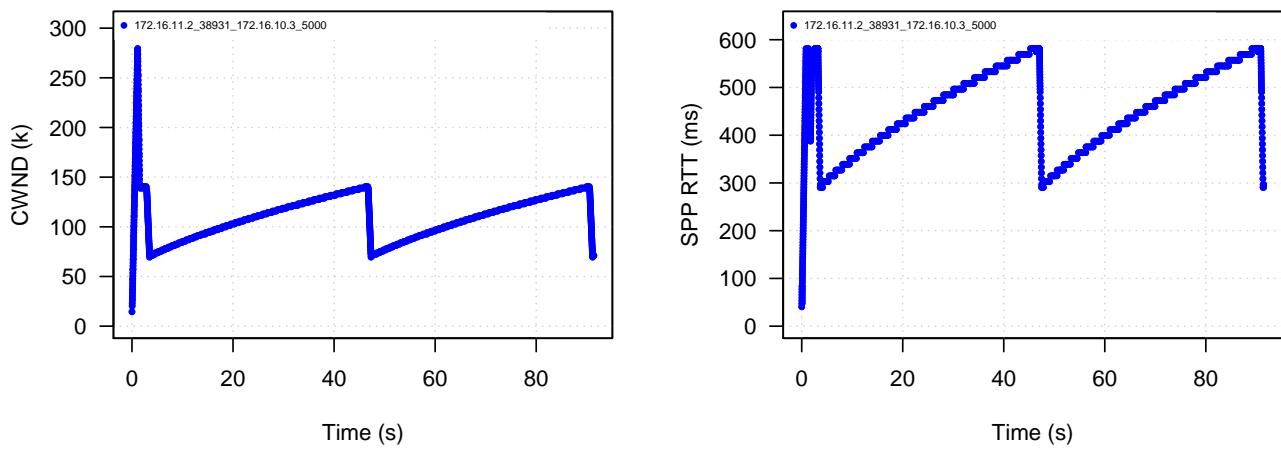


Figure 5. Linux NewReno: 2Mbps bottleneck rate, 40ms base RTT, 90 packet buffer

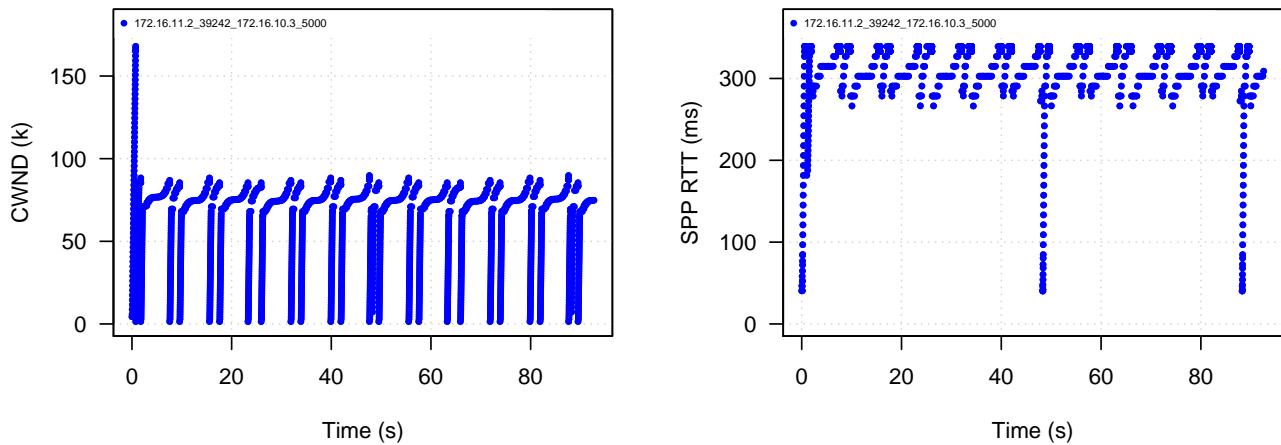


Figure 6. FreeBSD CUBIC: 2Mbps bottleneck rate, 40ms base RTT, 50 packet buffer

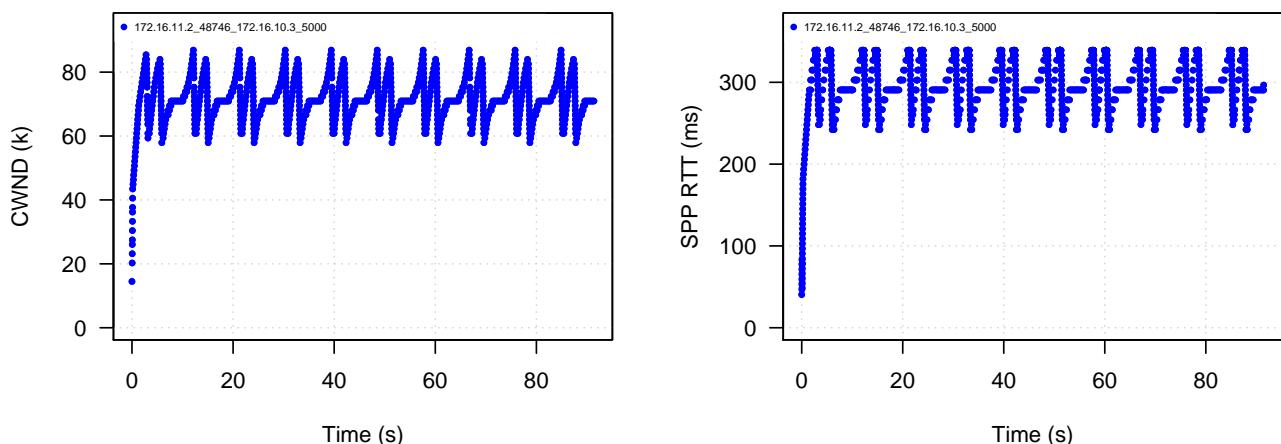


Figure 7. Linux CUBIC: 2Mbps bottleneck rate, 40ms base RTT, 50 packet buffer

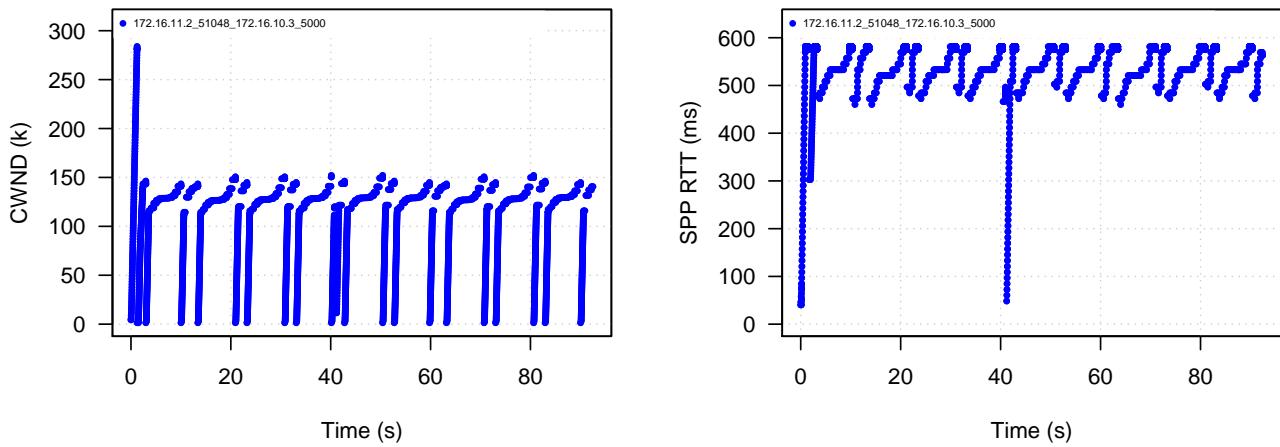


Figure 8. FreeBSD CUBIC: 2Mbps bottleneck rate, 40ms base RTT, 90 packet buffer

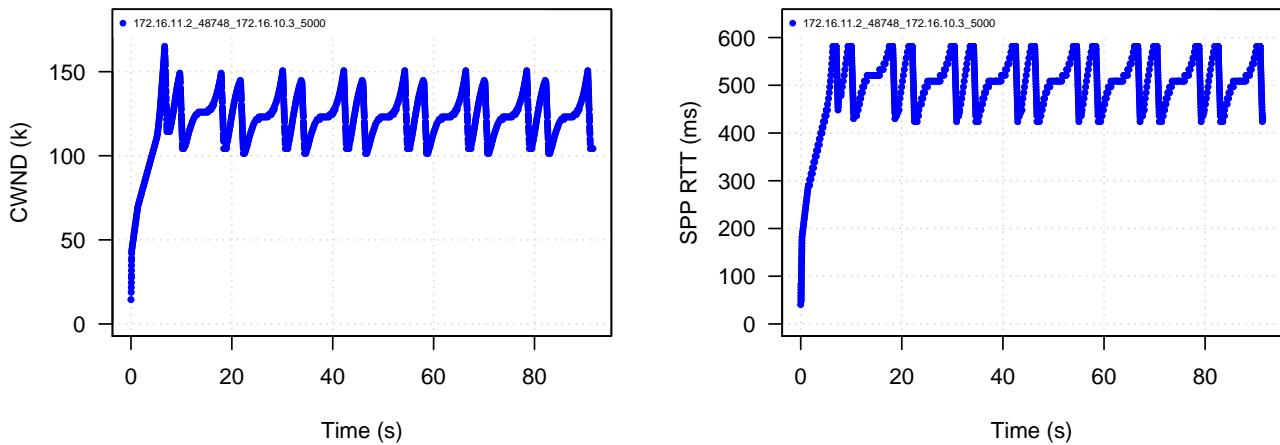


Figure 9. Linux CUBIC: 2Mbps bottleneck rate, 40ms base RTT, 90 packet buffer

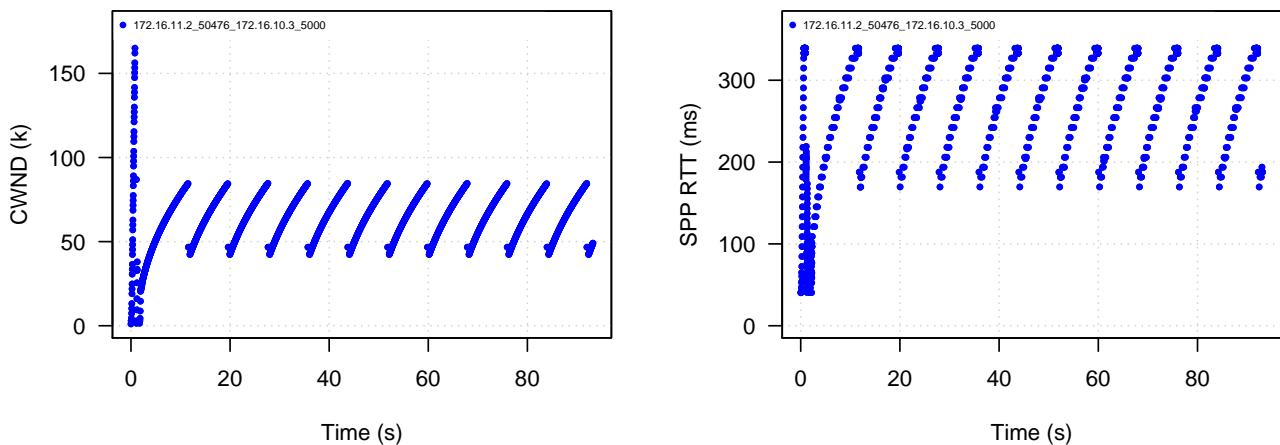


Figure 10. Windows 7 default TCP: 2Mbps bottleneck rate, 40ms base RTT, 50 packet buffer

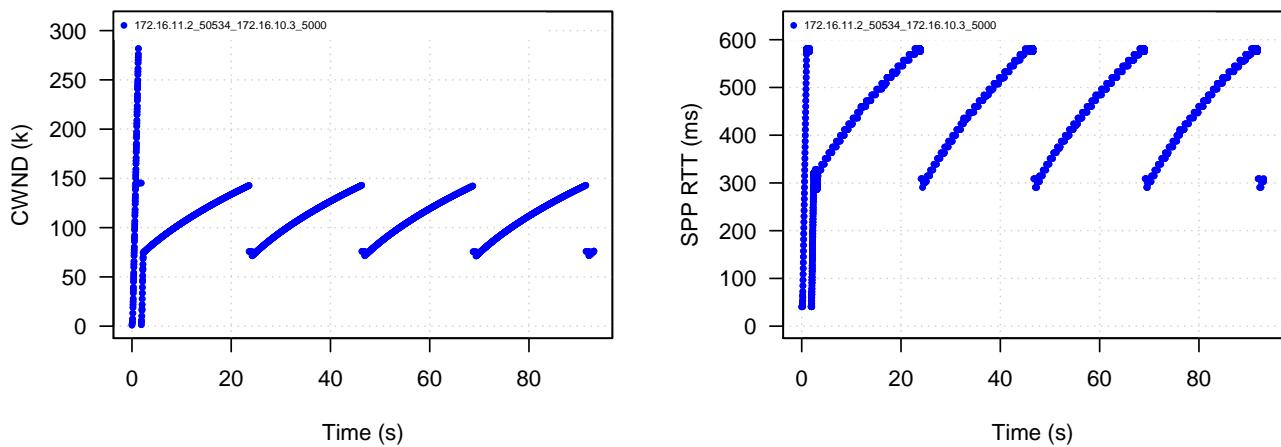


Figure 11. Windows 7 default TCP: 2Mbps bottleneck rate, 40ms base RTT, 90 packet buffer

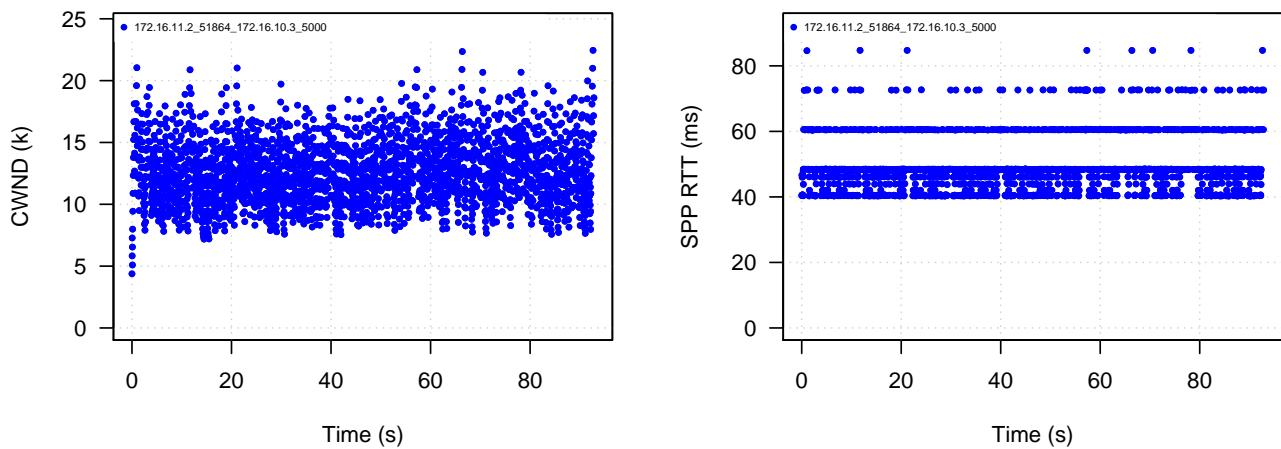


Figure 12. FreeBSD CDG: 2Mbps bottleneck rate, 40ms base RTT, 50 packet buffer

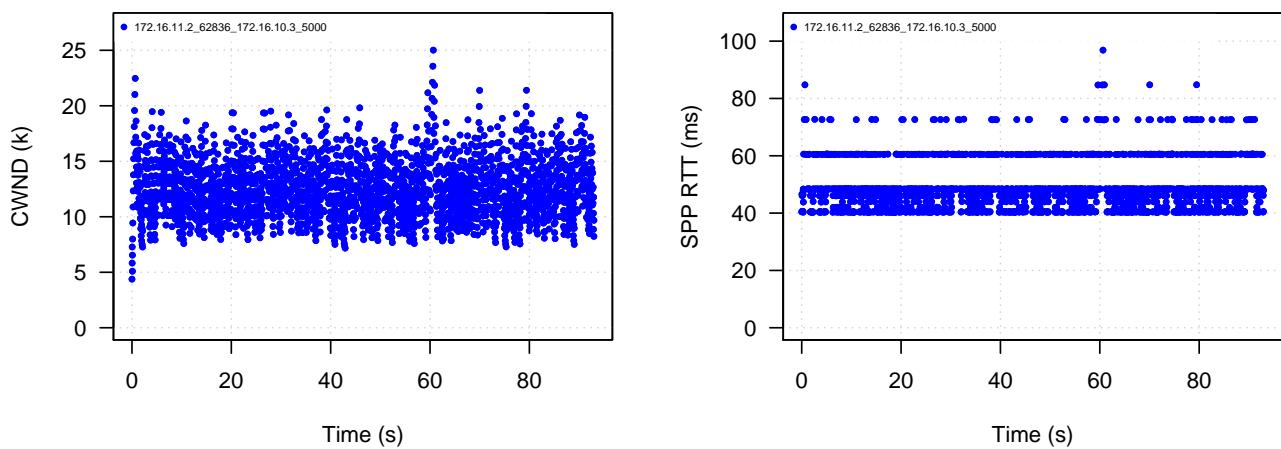


Figure 13. FreeBSD CDG: 2Mbps bottleneck rate, 40ms base RTT, 90 packet buffer

IV. CWND AND RTT VERSUS TIME AT 10MBPS

Now we look at the behaviour of cwnd and RTT versus time for a bottleneck rate limit of 10Mbps, bottleneck buffer of 90 packets and all other parameters as in the previous section. Note that at 10Mbps the full-size 1500 byte packet corresponds to 1.2 ms.¹¹

A. Newreno – FreeBSD and Linux

Figures 14 and 15 show the behaviour of cwnd and RTT over time for NewReno implementations under FreeBSD and Linux respectively where the bottleneck buffer is 90 packets long.

NewReno shows the classic slow start followed by cyclical movement between fast recovery/fast retransmit and congestion avoidance modes. Compared to the 2Mbps scenarios, cwnd cycles faster and the peak RTT is lower (since the ‘full’ buffer drains faster at 10Mbps).

B. CUBIC – FreeBSD and Linux

Figures 16 and 17 show the behaviour of cwnd and RTT over time for CUBIC implementations under FreeBSD and Linux respectively where the bottleneck buffer is 90 packets long.

C. Windows 7

Figures 18 shows the behaviour of cwnd and RTT over time for Windows 7’s default TCP implementation where the bottleneck buffer is 90 packets long. The behaviour is as expected, and similar to FreeBSD’s NewReno under these simplified circumstances.

D. CDG – FreeBSD

Figure 12 shows the behaviour of cwnd and RTT over time for FreeBSD’s CDG implementation where the bottleneck buffer is 90 packets long. As with the 2Mbps case, CDG shows the expected noisy variation in cwnd around a fairly low absolute value. The resulting absolute RTTs are quite low, making the quantisation of RTT values also very distinct.

¹¹Hence the congestion-induced component of RTT will jump in multiples of 1.2 ms per packet, and frequently by 2.4 ms as back-to-back data packets hit the bottleneck.

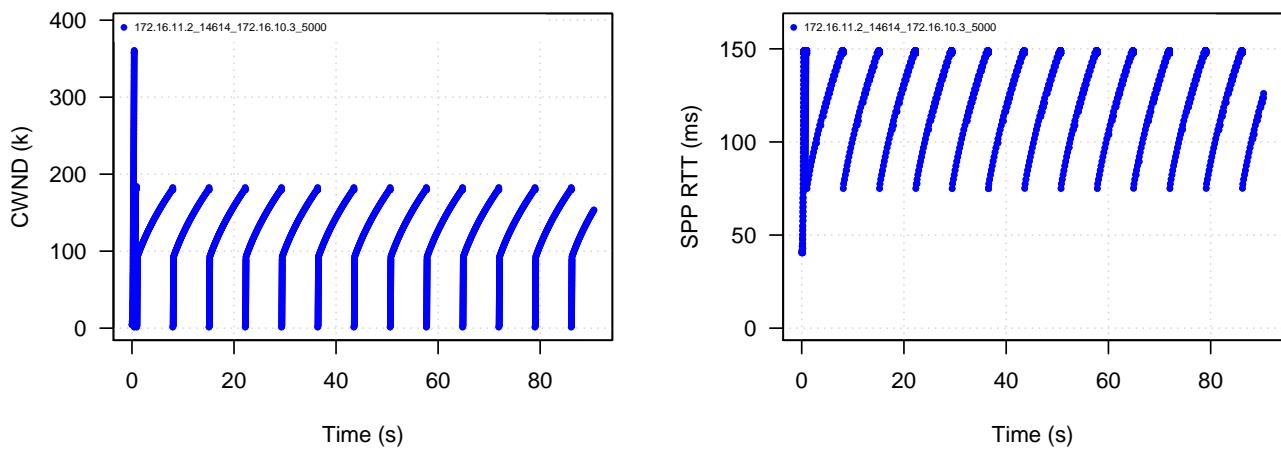


Figure 14. FreeBSD NewReno: 10Mbps, 40ms base RTT, 90 packet buffer

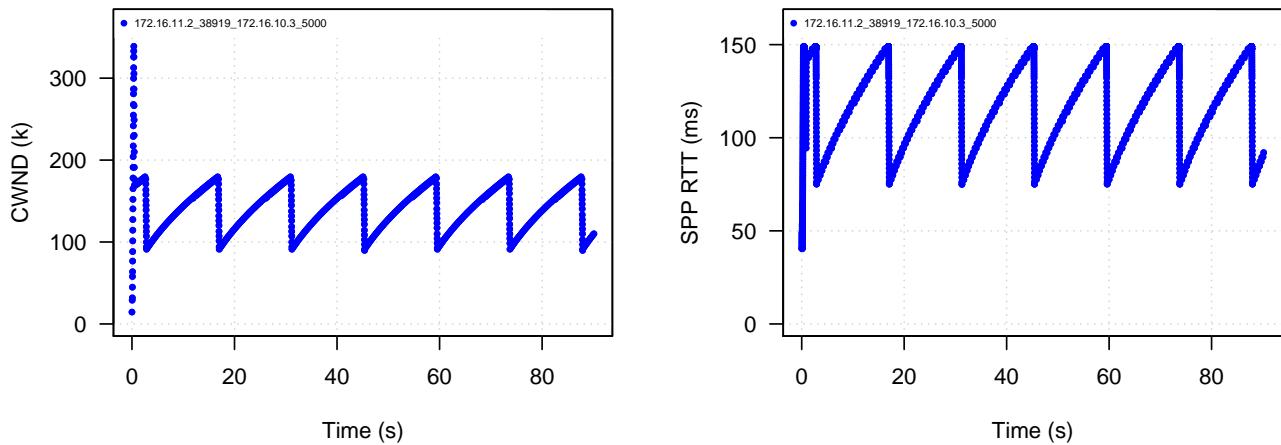


Figure 15. Linux NewReno: 10Mbps bottleneck rate, 40ms base RTT, 90 packet buffer

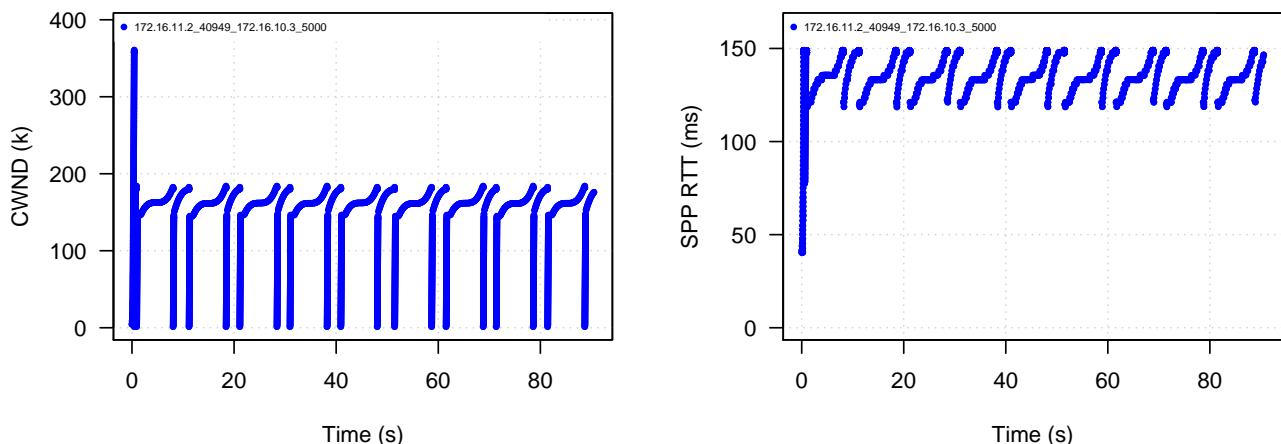


Figure 16. FreeBSD CUBIC: 10Mbps bottleneck rate, 40ms base RTT, 90 packet buffer

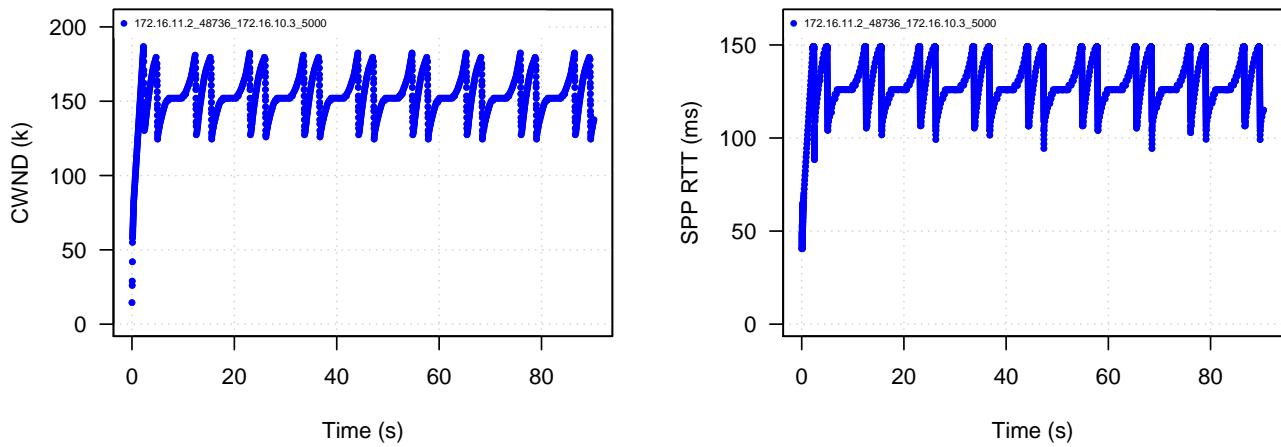


Figure 17. Linux CUBIC: 10Mbps bottleneck rate, 40ms base RTT, 90 packet buffer

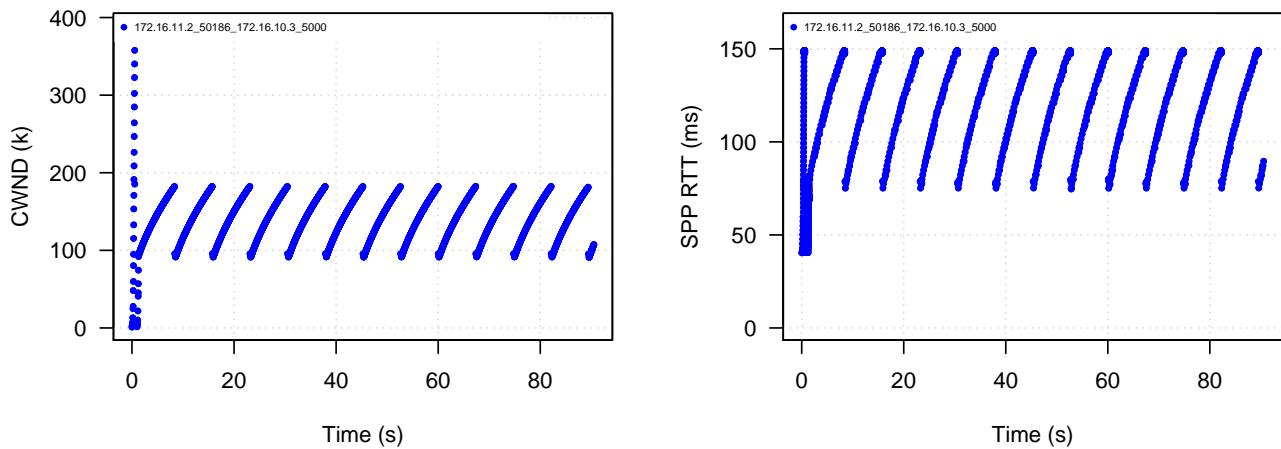


Figure 18. Windows 7 default TCP: 10Mbps bottleneck rate, 40ms base RTT, 90 packet buffer

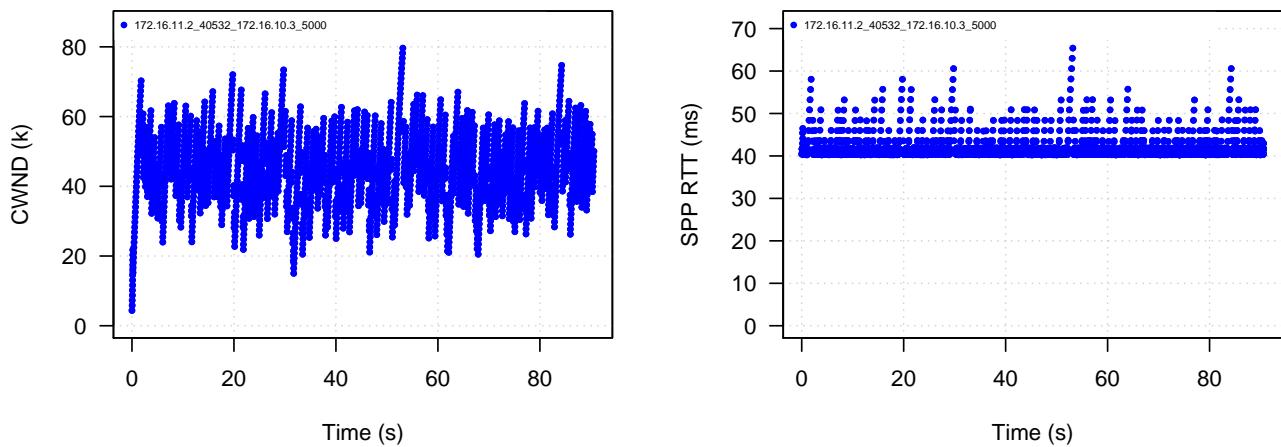


Figure 19. FreeBSD CDG: 10Mbps bottleneck rate, 40ms base RTT, 90 packet buffer

V. THROUGHPUT AND INDUCED RTT VERSUS BASE OWD AND BUFFER SIZE

This section looks at the impact of varying different path parameters – base (intrinsic) OWD, bottleneck buffer size and speed – on achievable throughput and overall RTT. First we look at the default TCPs of Windows 7, FreeBSD and Linux. Then we compare FreeBSD’s CDG and NewReno.

A. Default TCPs: Windows 7, FreeBSD and Linux

1) *Running at 2Mbps*: Figures 20 and 21 provide a side-by-side comparison of induced RTT and throughput distributions for TCP flows using these default algorithms over a 2Mbps path. For clarity we plot results for only a subset of OWD (0, 20, 40 and 100ms) and all three bottleneck buffer sizes (50, 90 and 180 packets).

Figure 21 shows that all three TCPs saturated the 2Mbps path for OWD up to 100ms and buffer size up to 180 packets. Figure 20 shows the median, peak and spread of RTTs increasing as the bottleneck buffer rises from 50 to 180 packets. We can also clearly see the base OWD influencing the minimum RTTs as expected.

2) *Running at 6Mbps*: Running the path at 6Mbps gives results broadly as expected. Figure 22 shows the RTT distributions across the same range of intrinsic OWD and bottleneck buffers when the path runs at 6Mbps. Figure 23 shows all three TCPs effectively saturated the 6Mbps path for OWD up to 100ms and buffer size up to 180 packets.

As with the 2Mbps case, median, peak and spread of RTTs increase as the bottleneck buffer rises from 50 to 180 packets, and the intrinsic OWD has a clear impact on minimum observed RTT. A small drop off for Linux CUBIC is evident at 100ms OWD and 90 or 180 packet buffers – TCP spends proportionally less time congesting the bottleneck buffer itself, so the RTT doesn’t spread quite as high above the path’s intrinsic (base) RTT.

3) *Running at 10Mbps*: Running the path at 10Mbps gives results broadly as expected. Figure 24 shows that the 10Mbps path is saturated by all three TCPs for all buffer sizes at low to moderate OWDs. At 100ms OWD we see the impact of bottleneck buffers being smaller than a path’s bandwidth-delay product – we see a modest reduction in median throughput with 50 and 90 packet bottleneck buffers.

Figure 25 shows the RTT distributions across the same range of OWD and bottleneck buffers when the path runs at 10Mbps. As with the 2Mbps case, median, peak and spread of RTTs increase as the bottleneck buffer rises from 50 to 180 packets, and the OWD has a clear impact on minimum observed RTT. However, at high OWD each TCP spends proportionally less time congesting the bottleneck buffer itself, so the RTT doesn’t spread quite as high above the path’s intrinsic (base) RTT.

B. Comparing FreeBSD’s CDG to NewReno

Figure 26 shows the RTT induced by single FreeBSD CDG and NewReno flows over a path with 10ms and 100ms intrinsic OWD, all three bottleneck rate limits and 90 and 180 packet bottleneck buffer sizes. Figure 27 shows the throughput achieved by the single CDG and NewReno flows under these same conditions.

Two aspects stand out:

- CDG introduces very little to the path’s intrinsic RTT, regardless of link speed and buffer size
- CDG achieves close to 95% or better of NewReno’s throughput when the path’s intrinsic RTT is low, but suffers noticeable performance degradation (and variability) when the path’s intrinsic RTT is high

These results illustrate that while CDG v0.1 may be useful in low-RTT environments, is not ready for more general use in the wider internet.¹²

VI. CONCLUSIONS AND FUTURE WORK

This report describes a range of simple single-flow TCP tests run on the CAIA TCP testbed using TEACUP v0.4.5. We have observed reasonably correct behaviours of NewReno and CUBIC under FreeBSD and Linux, and Compound TCP under and Windows 7. The bottleneck router appears to produce the desired path characteristics for simple cases noted here.

Future work will include more advanced AQMs (such as PIE and fq_codel), asymmetric path latencies, asymmetric path bottleneck bandwidths and concurrent (competing) TCP flows in various configurations. Future work may also attempt to draw some conclusions about which of the tested TCP and AQM algorithms are ‘better’ by various metrics.

¹²FreeBSD 9.2 includes CDG v0.1 as a selectable CC algorithm for research and experimentation, and certainly does not recommend it for normal use.

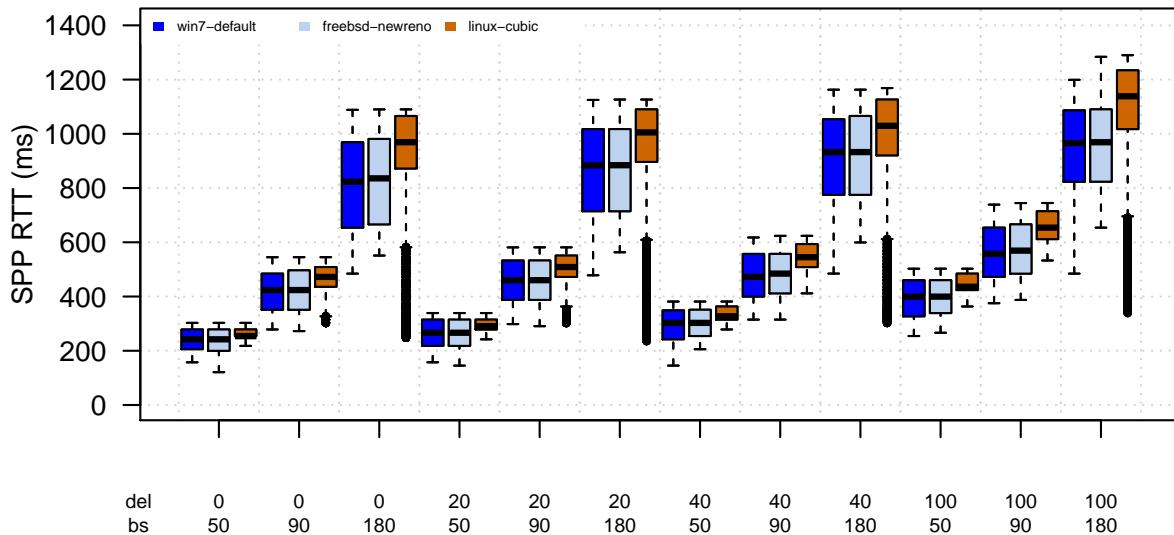


Figure 20. RTT vs OWD and Buffer size: Windows 7 default TCP, FreeBSD NewReno and Linux CUBIC @ 2Mbps
[OWD (del): 0ms, 20ms, 40ms and 100ms. Buffer size (bs): 50, 90 and 180 packets]

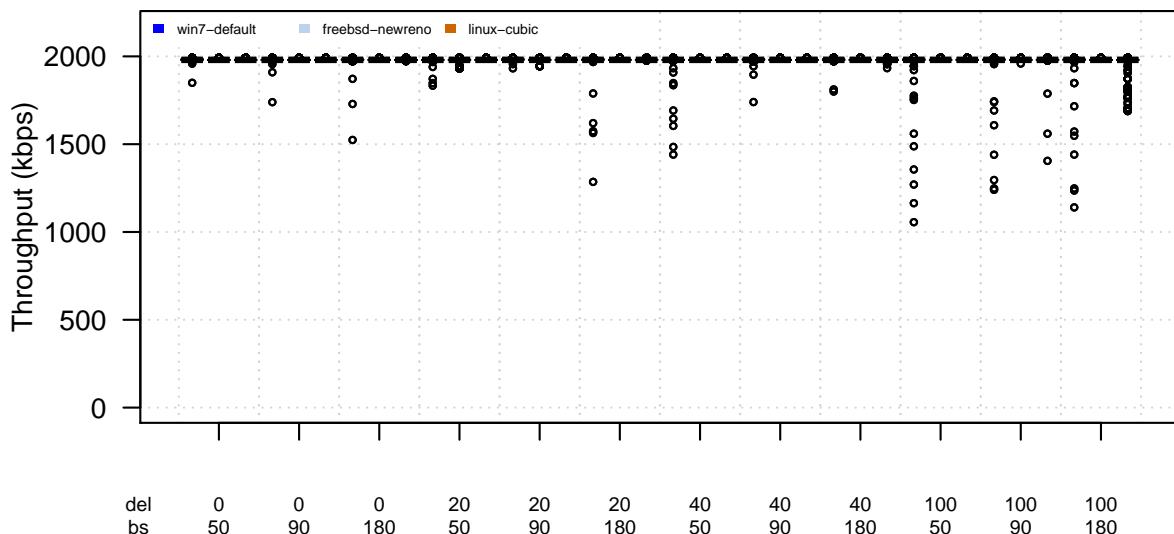


Figure 21. Throughput vs OWD and Buffer size: Windows 7 default TCP, FreeBSD NewReno and Linux CUBIC @ 2Mbps
[OWD (del): 0ms, 20ms, 40ms and 100ms. Buffer size (bs): 50, 90 and 180 packets]

ACKNOWLEDGEMENTS

TEACUP v0.4.5 was developed at CAIA by Sebastian Zander, as part of a project funded by Cisco Systems and titled “Study in TCP Congestion Control Performance In A Data Centre”. This is a collaborative effort between CAIA and Mr Fred Baker of Cisco Systems.

APPENDIX A FREEBSD TCP STACK CONFIGURATION

For the NewReno, CUBIC and CDG trials:

uname

```
FreeBSD newtcp3.caia.swin.edu.au 9.2-RELEASE FreeBSD
9.2-RELEASE #0 r255898: Thu Sep 26 22:50:31 UTC 2013
root@bake.isc.freebsd.org:/usr/obj/usr/src/sys/GENERIC
amd64
```

System information from sysctl

- kern.ostype: FreeBSD
- kern.osrelease: 9.2-RELEASE
- kern.osrevision: 199506

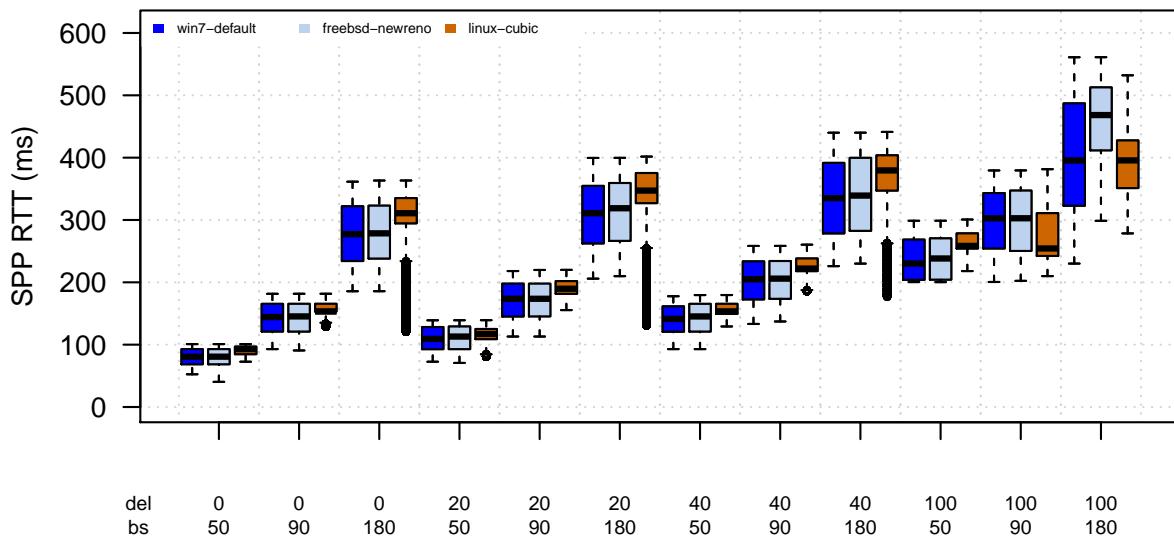


Figure 22. RTT vs OWD and Buffer size: Windows 7 default TCP, FreeBSD NewReno and Linux CUBIC @ 6Mbps
[OWD (del): 0ms, 20ms, 40ms and 100ms. Buffer size (bs): 50, 90 and 180 packets]

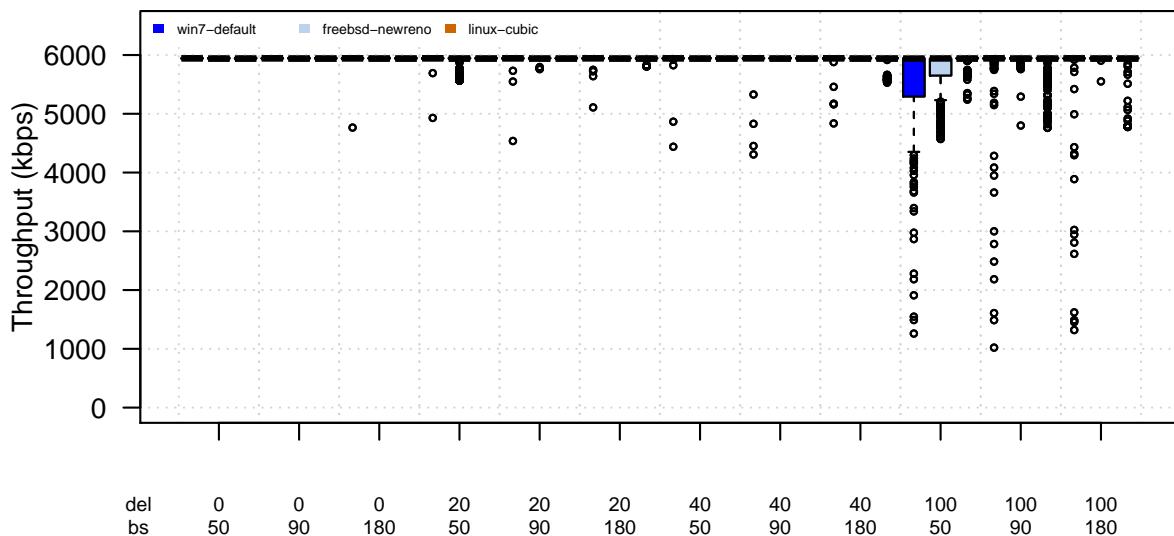


Figure 23. Throughput vs OWD and Buffer size: Windows 7 default TCP, FreeBSD NewReno and Linux CUBIC @ 6Mbps
[OWD (del): 0ms, 20ms, 40ms and 100ms. Buffer size (bs): 50, 90 and 180 packets]

net.inet.tcp information from sysctl

- net.inet.tcp.rfc1323: 1
- net.inet.tcp.mssdfilt: 536
- net.inet.tcp.keepidle: 7200000
- net.inet.tcp.keepintvl: 75000
- net.inet.tcp.sendspace: 32768
- net.inet.tcp.recvspace: 65536
- net.inet.tcp.keepinit: 75000
- net.inet.tcp.delacktime: 100
- net.inet.tcp.v6mssdfilt: 1220
- net.inet.tcp.cc.available: newreno, cdg¹³
- net.inet.tcp.cc.algorithm: cdg

¹³Or cubic or newreno

- net.inet.tcp.cc.cdg.loss_compete_hold_backoff: 5
- net.inet.tcp.cc.cdg.loss_compete_consec_cong: 5
- net.inet.tcp.cc.cdg.smoothing_factor: 8
- net.inet.tcp.cc.cdg.exp_backoff_scale: 3
- net.inet.tcp.cc.cdg.beta_loss: 50
- net.inet.tcp.cc.cdg.beta_delay: 70
- net.inet.tcp.cc.cdg.alpha_inc: 0
- net.inet.tcp.cc.cdg.version: 0.1
- net.inet.tcp.hostcache.purge: 0
- net.inet.tcp.hostcache.prune: 5
- net.inet.tcp.hostcache.expire: 1
- net.inet.tcp.hostcache.count: 0
- net.inet.tcp.hostcache.bucketlimit: 30
- net.inet.tcp.hostcache.hashsize: 512
- net.inet.tcp.hostcache.cachelimit: 15360

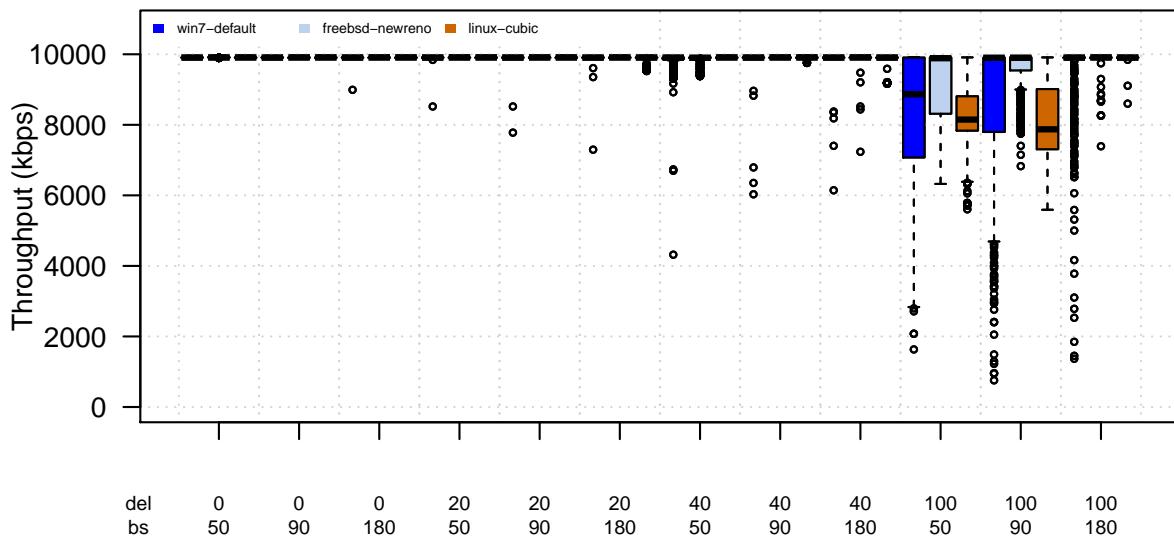


Figure 24. Throughput vs OWD and Buffer size: Windows 7 default TCP, FreeBSD NewReno and Linux CUBIC @ 10Mbps
[OWD (del): 0ms, 20ms, 40ms and 100ms. Buffer size (bs): 50, 90 and 180 packets]

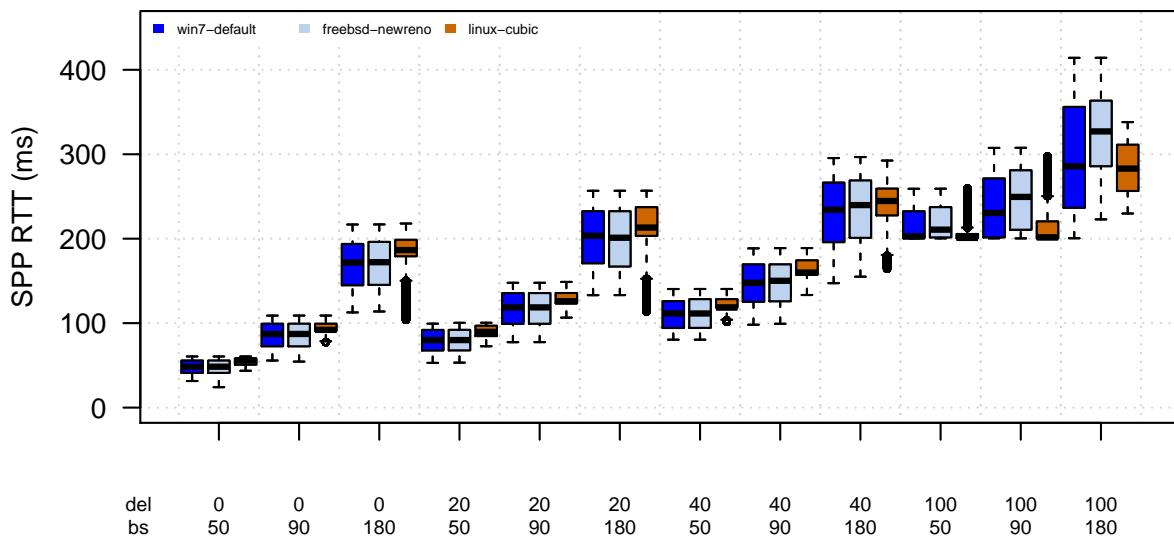


Figure 25. RTT vs OWD and Buffer size: Windows 7 default TCP, FreeBSD NewReno and Linux CUBIC @ 10Mbps
[OWD (del): 0ms, 20ms, 40ms and 100ms. Buffer size (bs): 50, 90 and 180 packets]

- net.inet.tcp.recvbuf_max: 2097152
- net.inet.tcp.recvbuf_inc: 16384
- net.inet.tcp.recvbuf_auto: 1
- net.inet.tcp.insecure_rst: 0
- net.inet.tcp.ecn.maxretries: 1
- net.inet.tcp.ecn.enable: 0
- net.inet.tcp.abc_l_var: 2
- net.inet.tcp.rfc3465: 1
- net.inet.tcp.experimental.initcwnd10: 0
- net.inet.tcp.rfc3390: 1
- net.inet.tcp.rfc3042: 1
- net.inet.tcp.drop_synfin: 0
- net.inet.tcp.delayed_ack: 1
- net.inet.tcp.blackhole: 0
- net.inet.tcp.log_in_vain: 0
- net.inet.tcp.sendbuf_max: 2097152
- net.inet.tcp.sendbuf_inc: 8192
- net.inet.tcp.sendbuf_auto: 1
- net.inet.tcp.tso: 0
- net.inet.tcp.path_mtu_discovery: 1
- net.inet.tcp.reass.overflow: 0
- net.inet.tcp.reass.cursegments: 0
- net.inet.tcp.reass.maxsegments: 1680
- net.inet.tcp.sack.globalholes: 0
- net.inet.tcp.sack.globalmaxholes: 65536
- net.inet.tcp.sack.maxholes: 128
- net.inet.tcp.sack.enable: 1
- net.inet.tcp.soreceive_stream: 0
- net.inet.tcp.isn_reseed_interval: 0
- net.inet.tcp.icmp_may_RST: 1
- net.inet.tcp.pcbcount: 6
- net.inet.tcp.do_tcpdrain: 1

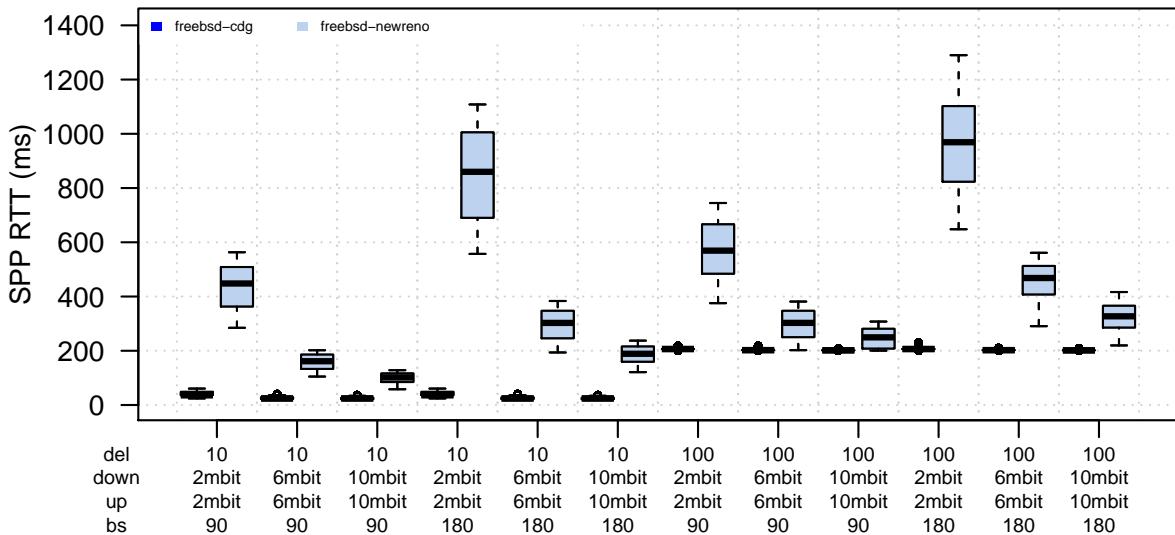


Figure 26. RTT vs OWD, Buffer size and rate limit: FreeBSD CDGv0.1 and FreeBSD NewReno
[OWD (del): 10ms and 100ms. Speeds (down and up): 2, 6 and 10Mbps. Buffer size (bs): 90 and 180 packets]

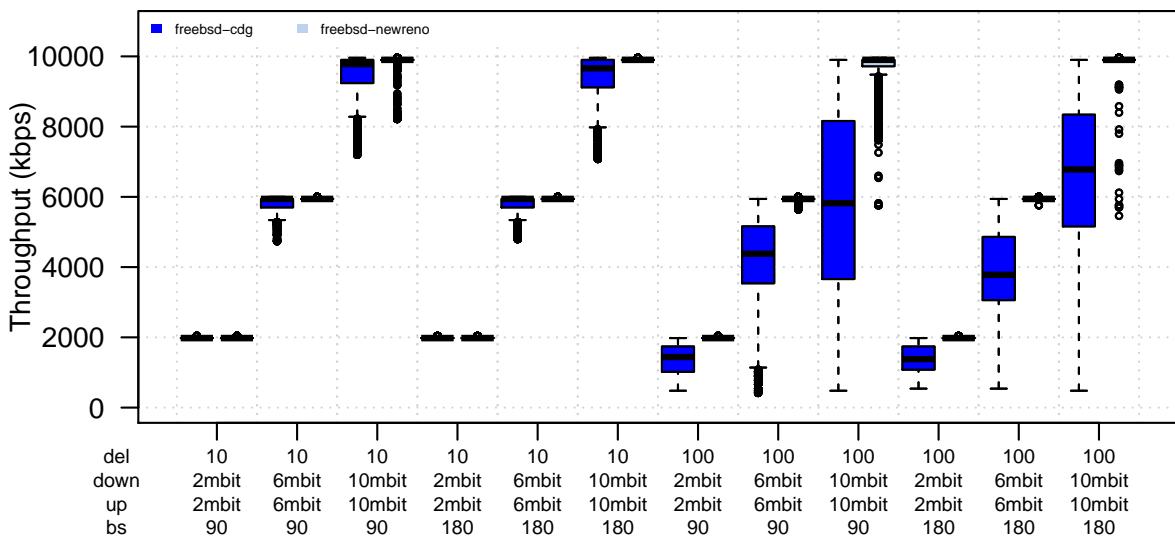


Figure 27. Throughput vs OWD, Buffer size and rate limit: FreeBSD CDGv0.1 and FreeBSD NewReno
[OWD (del): 10ms and 100ms. Speeds (down and up): 2, 6 and 10Mbps. Buffer size (bs): 90 and 180 packets]

- net.inet.tcp.tcbhashsize: 512
- net.inet.tcp.log_debug: 0
- net.inet.tcp.minmss: 216
- net.inet.tcp.syncache.rst_on_sock_fail: 1
- net.inet.tcp.syncache.rexmtlimit: 3
- net.inet.tcp.syncache.hashsize: 512
- net.inet.tcp.syncache.count: 0
- net.inet.tcp.syncache.cachelimit: 15375
- net.inet.tcp.syncache.bucketlimit: 30
- net.inet.tcp.syncookies_only: 0
- net.inet.tcp.syncookies: 1
- net.inet.tcp.timer_race: 0
- net.inet.tcp.per_cpu_timers: 0
- net.inet.tcp.rexmit_drop_options: 1
- net.inet.tcp.keepcnt: 8
- net.inet.tcp.finwait2_timeout: 60000

- net.inet.tcp.fast_finwait2_recycle: 0
- net.inet.tcp.always_keepalive: 1
- net.inet.tcp.rexmit_slop: 200
- net.inet.tcp.rexmit_min: 30
- net.inet.tcp.msl: 30000
- net.inet.tcp.nolocaltimewait: 0
- net.inet.tcp.maxtcptw: 5120

APPENDIX B LINUX TCP STACK CONFIGURATION

For the CUBIC and NewReno trials:

uname

```
Linux newtcp3.caia.swin.edu.au 3.9.8-desktop-web10g
#1 SMP PREEMPT Wed Jan 8 20:20:07 EST 2014 x86_64
x86_64 x86_64 GNU/Linux
```

System information from sysctl

- kernel.osrelease = 3.9.8-desktop-web10g
- kernel.ostype = Linux
- kernel.version = #1 SMP PREEMPT Wed Jan 8 20:20:07 EST 2014

net.ipv4.tcp information from sysctl

- net.ipv4.tcp_abort_on_overflow = 0
- net.ipv4.tcp_adv_win_scale = 1
- net.ipv4.tcp_allowed_congestion_control = cubic reno
- net.ipv4.tcp_app_win = 31
- net.ipv4.tcp_available_congestion_control = cubic reno
- net.ipv4.tcp_base_mss = 512
- net.ipv4.tcp_challenge_ack_limit = 100
- net.ipv4.tcp_congestion_control = cubic¹⁴
- net.ipv4.tcp_cookie_size = 0
- net.ipv4.tcp_dma_copybreak = 4096
- net.ipv4.tcp_dsack = 1
- net.ipv4.tcp_early_retrans = 2
- net.ipv4.tcp_ecn = 0
- net.ipv4.tcp_fack = 1
- net.ipv4.tcp_fastopen = 0
- net.ipv4.tcp_fastopen_key = e8a015b2-e29720c6-4ce4eff7-83c84664
- net.ipv4.tcp_fin_timeout = 60
- net.ipv4.tcp_frto = 2
- net.ipv4.tcp_frto_response = 0
- net.ipv4.tcp_keepalive_intvl = 75
- net.ipv4.tcp_keepalive_probes = 9
- net.ipv4.tcp_keepalive_time = 7200
- net.ipv4.tcp_limit_output_bytes = 131072
- net.ipv4.tcp_low_latency = 0
- net.ipv4.tcp_max_orphans = 16384
- net.ipv4.tcp_max_ssthresh = 0
- net.ipv4.tcp_max_syn_backlog = 128
- net.ipv4.tcp_max_tw_buckets = 16384
- net.ipv4.tcp_mem = 89955 119943 179910
- net.ipv4.tcp_moderate_rcvbuf = 0
- net.ipv4.tcp_mtu_probing = 0
- net.ipv4.tcp_no_metrics_save = 1
- net.ipv4.tcp_orphan_retries = 0
- net.ipv4.tcp_reordering = 3
- net.ipv4.tcp_retrans_collapse = 1
- net.ipv4.tcp_retries1 = 3
- net.ipv4.tcp_retries2 = 15
- net.ipv4.tcp_rfc1337 = 0
- net.ipv4.tcp_rmem = 4096 87380 6291456
- net.ipv4.tcp_sack = 1
- net.ipv4.tcp_slow_start_after_idle = 1
- net.ipv4.tcp_stdurg = 0
- net.ipv4.tcp_syn_retries = 6
- net.ipv4.tcp_synack_retries = 5
- net.ipv4.tcp_syncookies = 1
- net.ipv4.tcp_thin_dupack = 0
- net.ipv4.tcp_thin_linear_timeouts = 0
- net.ipv4.tcp_timestamps = 1
- net.ipv4.tcp_tso_win_divisor = 3
- net.ipv4.tcp_tw_recycle = 0
- net.ipv4.tcp_tw_reuse = 0

¹⁴Or newreno

- net.ipv4.tcp_window_scaling = 1
- net.ipv4.tcp_wmem = 4096 65535 4194304
- net.ipv4.tcp_workaround_signed_windows = 0

APPENDIX C WINDOWS 7 TCP STACK CONFIGURATION

For the compound TCP trials:

Cygwin uname

```
CYGWIN_NT-6.1 newtcp3 1.7.25(0.270/5/3) 2013-08-31
20:37 x86_64 Cygwin
```

netsh int show int

Admin State	State	Type	Interface Name
Enabled	Connected	Dedicated	Local Area Connection 2
Enabled	Connected	Dedicated	Local Area Connection

netsh int tcp show global

Querying active state...	
TCP Global Parameters	

Receive-Side Scaling State : enabled	
Chimney Offload State : disabled	
NetDMA State : enabled	
Direct Cache Acess (DCA) : disabled	
Receive Window Auto-Tuning Level : normal	
Add-On Congestion Control Provider : none	
ECN Capability : disabled	
RFC 1323 Timestamps : disabled	

netsh int tcp show heuristics

TCP Window Scaling heuristics Parameters	

Window Scaling heuristics : enabled	
Qualifying Destination Threshold : 3	
Profile type unknown : normal	
Profile type public : normal	
Profile type private : normal	
Profile type domain : normal	
netsh int tcp show security	

Querying active state...	

```
Memory Pressure Protection : disabled
```

```
Profiles : enabled
```

netsh int tcp show chimneystats

```
Your System Administrator has disabled TCP  
Chimney.
```

```
netsh int ip show offload
```

```
Interface 1: Loopback Pseudo-Interface 1
```

```
Interface 12: Local Area Connection
```

```
Interface 14: Local Area Connection 2
```

netsh int ip show global

```
Querying active state...
```

```
General Global Parameters
```

```
-----  
Default Hop Limit : 128 hops
```

```
Neighbor Cache Limit : 256 entries per  
interface
```

```
Route Cache Limit : 128 entries per  
compartment
```

```
Reassembly Limit : 32893088 bytes
```

```
ICMP Redirects : enabled
```

```
Source Routing Behavior : dontforward
```

```
Task Offload : disabled
```

```
Dhcp Media Sense : enabled
```

```
Media Sense Logging : disabled
```

```
MLD Level : all
```

```
MLD Version : version3
```

```
Multicast Forwarding : disabled
```

```
Group Forwarded Fragments : disabled
```

```
Randomize Identifiers : enabled
```

```
Address Mask Reply : disabled
```

```
Current Global Statistics
```

```
-----  
Number of Compartments : 1
```

```
Number of NL clients : 7
```

```
Number of FL providers : 4
```

REFERENCES

- [1] S. Zander, G. Armitage, “TEACUP v0.4 – A System for Automated TCP Testbed Experiments,” Centre for Advanced Internet Architectures, Swinburne University of Technology, Tech. Rep. 140314A, March 2014. [Online]. Available: <http://caia.swin.edu.au/reports/140314A/CAIA-TR-140314A.pdf>
- [2] ——, “CAIA Testbed for TCP Experiments,” Centre for Advanced Internet Architectures, Swinburne University of Technology, Tech. Rep. 140314B, March 2014. [Online]. Available: <http://caia.swin.edu.au/reports/140314B/CAIA-TR-140314B.pdf>
- [3] “The Web10G Project.” [Online]. Available: <http://web10g.org>
- [4] “iperf Web Page.” [Online]. Available: <http://iperf.fr/>
- [5] “NEWTCP Project Tools.” [Online]. Available: <http://caia.swin.edu.au/urp/newtcp/tools.html>
- [6] L. Stewart, “SIFTR – Statistical Information For TCP Research.” [Online]. Available: <http://www.freebsd.org/cgi/man.cgi?query=siftr>
- [7] S. Zander and G. Armitage, “Minimally-Intrusive Frequent Round Trip Time Measurements Using Synthetic Packet-Pairs,” in *The 38th IEEE Conference on Local Computer Networks (LCN 2013)*, 21-24 October 2013.
- [8] A. Heyde, “SPP Implementation,” August 2013. [Online]. Available: <http://caia.swin.edu.au/tools/spp/downloads.html>