Baseline two-flow and three-flow TCP results over PIE and fq_codel for TEACUP v0.4 testbed

Grenville Armitage
Centre for Advanced Internet Architectures, Technical Report 140516A
Swinburne University of Technology
Melbourne, Australia
garmitage@swin.edu.au

***Abstract*—This technical report summarises a basic set of two-flow and three-flow TCP experiments designed to illustrate the operation of our TEACUP testbed under teacup-v0.4.8 using pfifo (tail-drop), PIE and fq_codel queue management schemes at the bottleneck router. We primarily observe that both PIE and fq_codel cause competing loss-based TCP flows to induce substantially less bottleneck queuing delays than pfifo, without materially impacting on throughput. We test FreeBSD's NewReno, Linux's CUBIC and Windows 7's default TCP. This report does *not* attempt to make any meaningful comparisons between the tested TCP algorithms, nor rigorously evaluate the consequences of using PIE or fq_codel.**

***Index Terms*—TCP, fq_codel, PIE, pfifo, experiments, testbed, TEACUP**

I. Introduction[1]

CAIA has developed TEACUP[2] [1] to support a comprehensive experimental evaluation of different TCP congestion control algorithms. Our actual testbed is described in [2]. This report summarises a basic set of two-flow and three-flow TCP experiments that illustrate the impact of pfifo (tail-drop), PIE [3] and fq_codel [4] (based on codel [5]) queue management schemes at the bottleneck router.

The trials use four or six hosts to create two or three competing flows respectively. The bottleneck in each case is a Linux-based router using `netem` and `tc` to provide independently configurable bottleneck rate limits and artificial one-way delay (OWD). The trials run over a small range of emulated path conditions using FreeBSD NewReno, Linux CUBIC and (for the two-flow case) Windows 7's default TCP.

[1]Erratum: Online copy updated July 13th 2014 to clarify use of Windows 7's default TCP.

[2]TCP Experiment Automation Controlled Using Python.

We primarily observe that both PIE and fq_codel cause competing loss-based TCP flows to induce substantially less bottleneck queuing delays than pfifo, without materially impacting throughput. This report does not attempt to make any meaningful comparisons between the tested TCP algorithms, nor do we explore the differences between PIE and fq_codel in any detail.

The rest of the report is organised as follows. Section II summarises the the testbed topology and physical configuration for these trials. We then summarise the impact of base RTT, buffer size and AQM on throughput and induced RTT for two flows in Section III and three flows in Section IV. Section V concludes and outlines future work.

II. Testbed topology and test conditions

Trials involved either two or three concurrent TCP connections pushing data through a single bottleneck for 60 or 90 seconds. The path has different emulated delays, bottleneck speeds and AQM algorithms. Here we document the testbed topology, 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 [6]), 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).

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

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 hosts on 172.16.10/24 as the data sources and hosts on 172.16.11/24 as the data sinks. See [2] for more technical details of how the router and each host was configured.

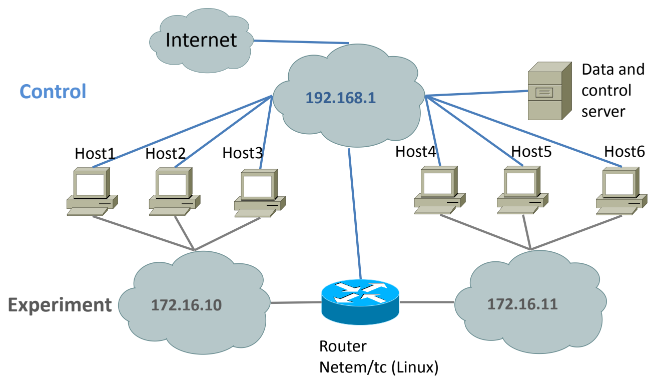


Figure 1: Testbed overview

B. Host operating system and TCP combinations

The two-flow trials used used three different operating systems, and four different TCP algorithms.

- FreeBSD: Newreno[4] and CDG[5] (v0.1)
- Linux: CUBIC
- Windows 7: default TCP

The three-flow trials used used two different operating systems, and three different TCP algorithms.[6]

- FreeBSD: Newreno and CDG (v0.1)
- Linux: CUBIC

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

C. Emulated path and bottleneck 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. Packets sit in a 1000-packet buffer while being delayed (to provide the artificial OWD), then sit in a smaller "bottleneck buffer" of configurable size (in packets) while being rate-shaped to the bottleneck bandwidth.

1) Bottleneck AQM: We repeated each trial using the Linux 3.10.18 kernel's implementations of pfifo, PIE and fq_codel algorithms in turn to manage the bottleneck buffer (queue) occupancy.

As noted in Section V.D of [2], we compiled PIE into the 3.10.18 kernel from source [7] dated July 2nd 2013,[7] and included a suitably patched iproute2 (v3.9.0) [8].

See Appendix D for more details on the bottleneck router's AQM configuration. Since they are largely intended to require minimal operator control or tuning, we used PIE and fq_codel at their default settings. As PIE and fq_codel are under active development, future experiments will likely requiring re-patching and updating of the router's kernel.

2) Path conditions: We emulated the following path and bottleneck conditions:

- 0% intrinsic loss rate[8]
- One way delay: 0, 10, 40 and 100ms
- Bottleneck bandwidths: 2, 6 and 10Mbps[9]
- Bottleneck buffer sizes: 50, 90 and 180 pkts

These conditions were applied bidirectionally, using separate delay and rate shaping stages in the router for traffic in each direction. Consequently, the path's intrinsic (base) RTT is always twice the configured OWD. The 180-packet bottleneck buffer was greater than the path's intrinsic BDP (bandwidth delay product) for all combinations of bandwidth and OWD.

D. Traffic generator and logging

Each concurrent TCP flow was generated using iperf 2.0.5 [9] on all three OSes, patched to enable correct control of the send and receive buffer sizes [10]. For each flow, iperf requests 600Kbyte socket buffers to ensure `cwnd` growth was not artificially limited by the maximum receive window.

The two-flow trials launched concurrent 90-second flows between the following host pairs:

[4]http://www.freebsd.org/cgi/man.cgi?query=cc_newreno
[5]http://www.freebsd.org/cgi/man.cgi?query=cc_cdg
[6]Due to temporary instability in the testbed we have no Windows 7 results for the three-flow trials.
[7]MODULE_INFO(srcversion, "1F54383BFCB1F4F3D4C7CE6");
[8]I.e. no additional packet loss beyond that induced by congestion of the bottleneck's buffer.
[9]The three-flow trials only used the 10Mbps rate limit.

- Host1→Host4
- Host2→Host5

The three-flow trials launched concurrent 60-second flows between the following host pairs (Figure 1):

- Host1→Host4
- Host2→Host5
- Host3→Host6

Data packets from all flows traverse the bottleneck router in the same direction.

TCP connection statistics were logged using SIFTR [11] under FreeBSD, Web10g [6] under Linux and TCP EStats under Windows.

Packets captured at both hosts with tcpdump were used to calculate non-smoothed end to end RTT estimates using CAIA's passive RTT estimator, SPP [12], [13].

E. Measuring throughput

'Instantaneous' throughput is an approximation derived from the actual bytes transferred during constant (but essentially arbitrary) windows of time. Long windows smooth out the effect of transient bursts or gaps in packet arrivals. Short windows can result in calculated throughput that swings wildly (but not necessarily meaningfully) from one measurement interval to the next.

For these experiments we use a two-second wide window sliding forward in steps of 0.5 second.

III. Impact of AQM on RTT and Throughput with Two concurrent flows

This section summarises the impact of varying the AQM and path parameters – base OWD, bottleneck buffer size and speed – on achievable throughput and overall RTT when two flows share the bottleneck. We ran two NewReno flows (section III-A), two CUBIC flows (section III-B), two Windows 7 default TCP flows (section III-C) and one NewReno with one CDG v0.1 flows flows (section III-D).

A. Two NewReno flows

Two competing NewReno flows ran for 90 seconds through the bottleneck between FreeBSD hosts.

1) RTT and throughput at 10Mbps: Figure 2a shows the RTT experienced by two competing FreeBSD NewReno flows through a 10Mbps bottleneck. As expected, the use of pfifo results in both flows experiencing extremely high queueing delays. However, using either PIE or fq_codel effectively drops the experienced RTT down almost to the path's intrinsic RTT.

Figure 2b shows that the throughput achieved by each flow is broadly the same using all three AQMs. Nevertheless, pfifo still creates a more unpredictable sharing of bandwidth, whilst fq_codel appears to provide exceedingly tight bandwidth sharing at 0ms and 10ms OWD.

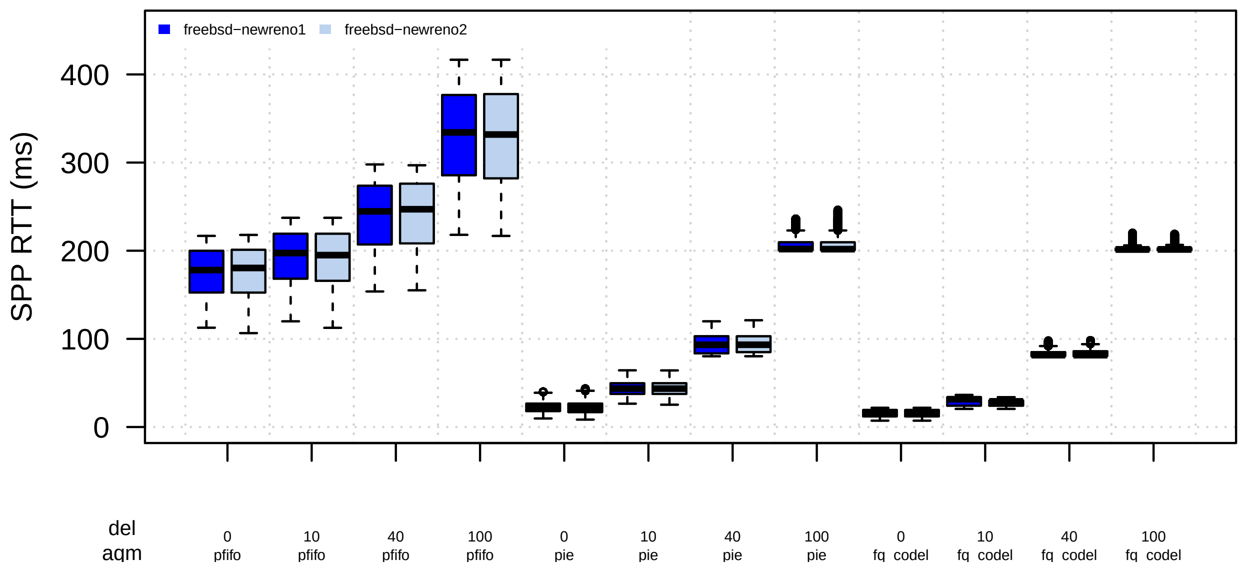
2) RTT and throughput at 2, 6 and 10Mbps and 20ms RTT: Figure 3a shows the variation of path RTT versus AQM with 2Mbps, 6Mbps and 10Mbps bottleneck rate limits at 10ms OWD (20ms RTT). The dramatic benefit of either PIE or fq_codel at lower bottleneck rates is quite clear. Figure 3b shows the variation in achieved throughput under the same circumstances. Both PIE and fq_codel achieve their RTT improvement with no meaningful impact on performance.[10]

3) Throughput, `cwnd` and RTT versus time – two runs with pfifo: Boxplots do not provide insight into how characteristics like throughput and RTT actually vary over the lifetime of an individual flow. Figure 4 illustrates how NewReno behaves during two runs of the same experimental conditions – 20ms RTT (10ms OWD), 10Mbps bottleneck rate limit, pfifo queue management of a 180-packet bottleneck buffer. In real testbeds no two trial runs will be identical.

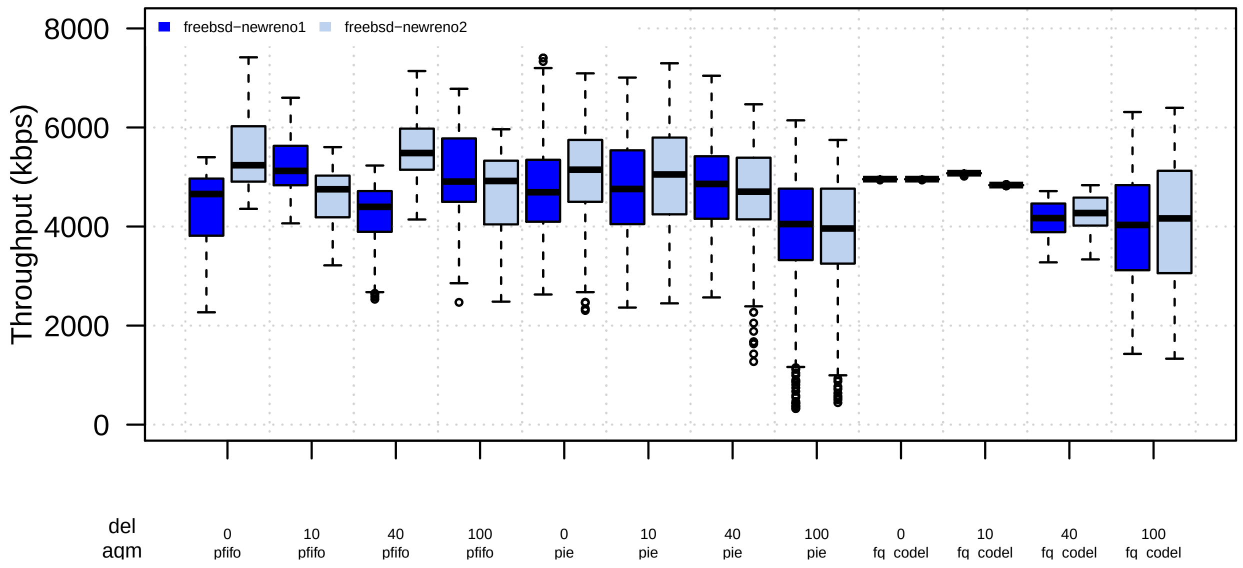
Figures 4a (throughput vs time) and 4c (`cwnd` vs time) show the first run experiences multiple periods where the two flows diverge and re-converge on half of the available path capacity. Figures 4b and 4d show this divergence occurring only once in the second run. Nevertheless, both runs show a similar spread of throughput over time. Given that queuing delays are imposed on all traffic sharing the dominant congested bottleneck, Figures 4e and 4f show the RTT fluctuations of both flows (around a median RTT just under 200ms) track closely over time within each run.

4) Throughput, `cwnd` and RTT versus time – one run with PIE and fq_codel: Next we look briefly at the

[10]Future work will evaluate why, for low OWDs, throughput appears to fluctuate more over time when the bottleneck is managed by PIE rather than fq_codel.



(a) RTT vs OWD and AQM with 10Mbps rate limit



(b) Throughput vs OWD and AQM with 10Mbps rate limit

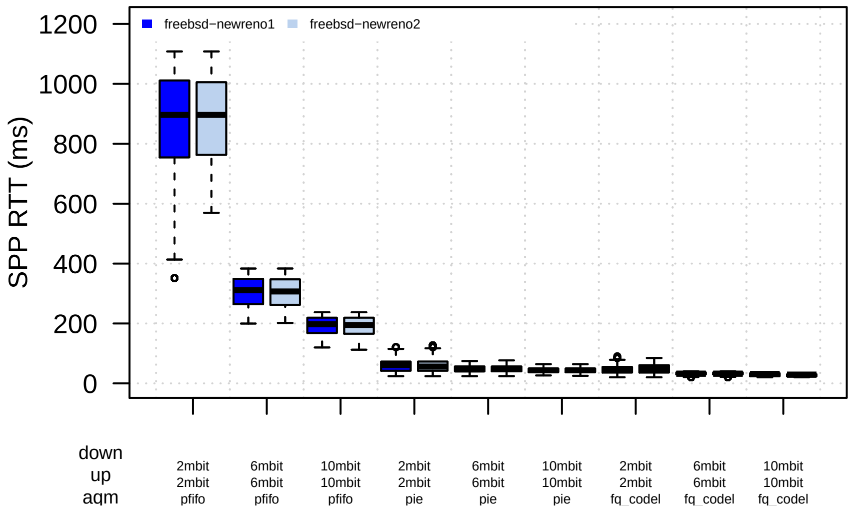
Figure 2: Two FreeBSD NewReno flows – 180 packet bottleneck buffer [*del*: OWD in ms, *aqm*: AQM scheme]

impact of PIE and fq_codel on flow behaviour over time in Figure 5.[11] Again, the path has a base 20ms RTT (10ms OWD), 10Mbps bottleneck rate limit and 180-packet bottleneck buffer.

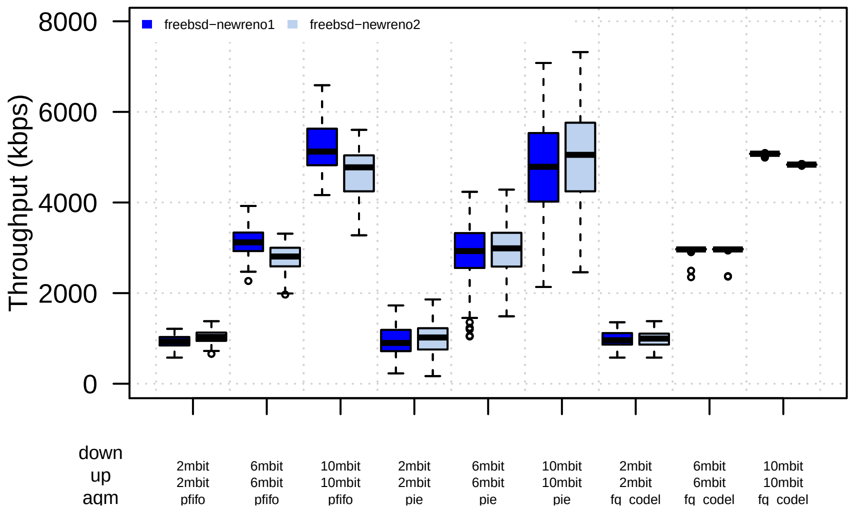
Figure 5a show each flow's throughput through the PIE bottleneck fluctuating relatively evenly over time, while Figure 5b shows each flow's throughput being equal and constant through the fq_codel bottleneck. This is consistent with the aggregate results in Figure 2b. At the same time, Figures 5c and 5d show `cwnd` cycling relatively rapidly for PIE, and bouncing very rapidly over a small range with fq_codel.

The nett effect on RTT is shown in Figures 5c and 5d – RTT cycles relatively rapidly between 20ms to 60ms for PIE, and bounces very rapidly between 20ms and 40ms with fq_codel. In both cases the reduction relative to RTT over a pfifo bottleneck (Figures 4e and 4f) is dramatic.

[11]Keeping in mind that clarity around the different impacts of PIE and fq_codel are a matter for future work.

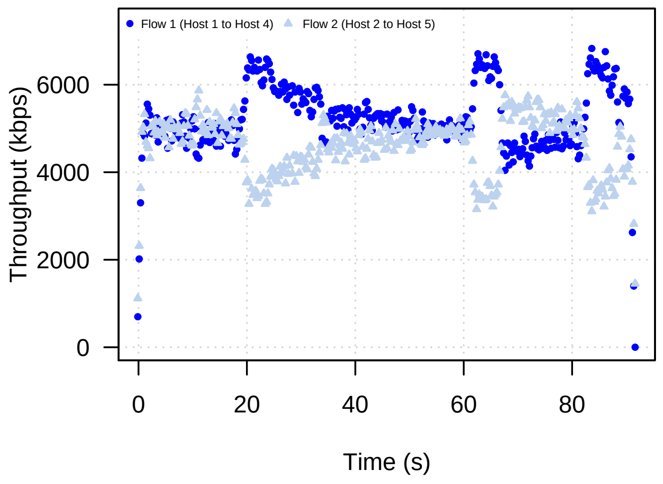


(a) RTT vs rate limit and AQM over 20ms RTT path

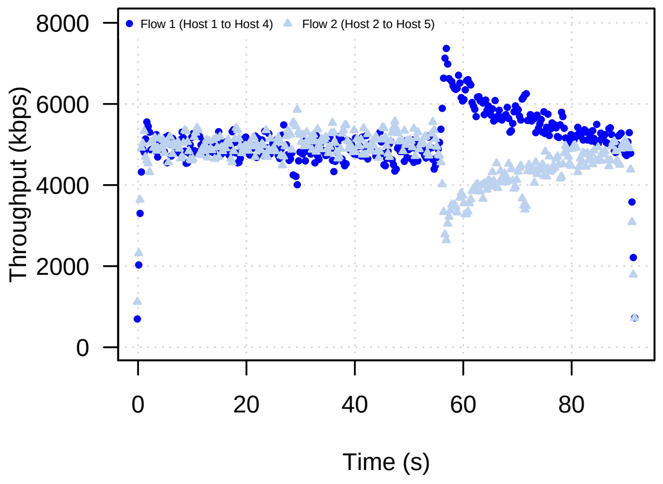


(b) Throughput vs rate limit and AQM over 20ms RTT path

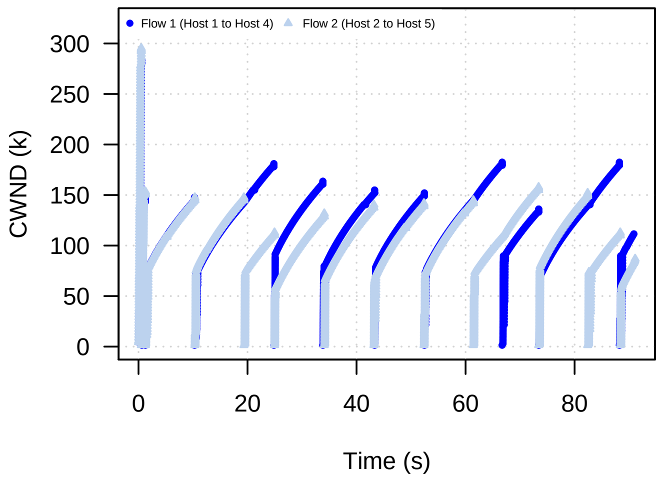
Figure 3: Two FreeBSD NewReno flows – 180 packet bottleneck buffer, symmetric ‘up’ and ‘down’ speeds



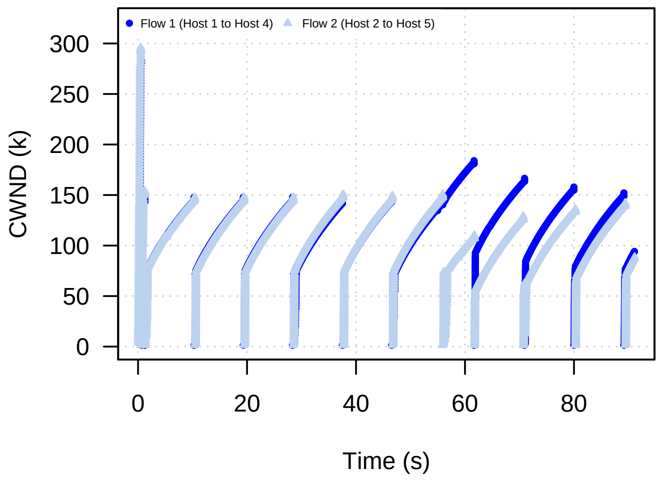
(a) Throughput vs time – first run



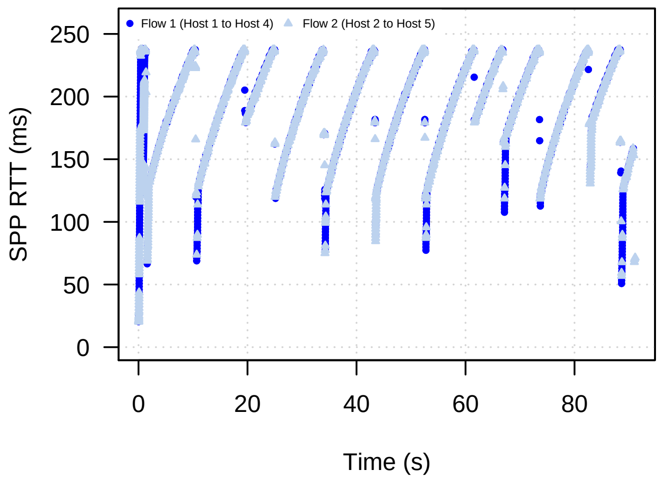
(b) Throughput vs time – second run



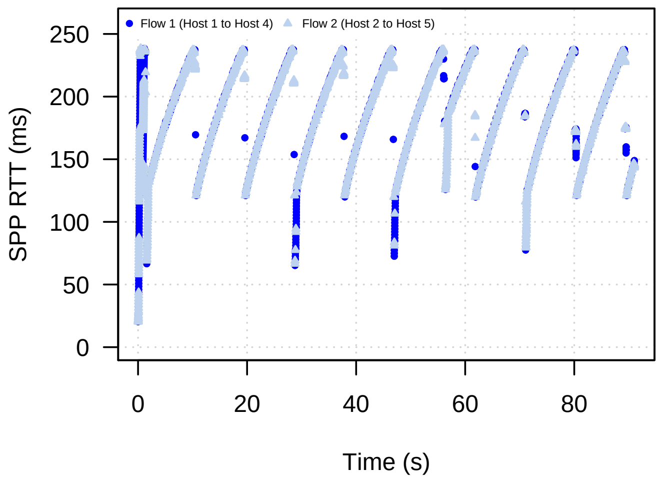
(c) `cwnd` vs time – first run



(d) `cwnd` vs time – second run

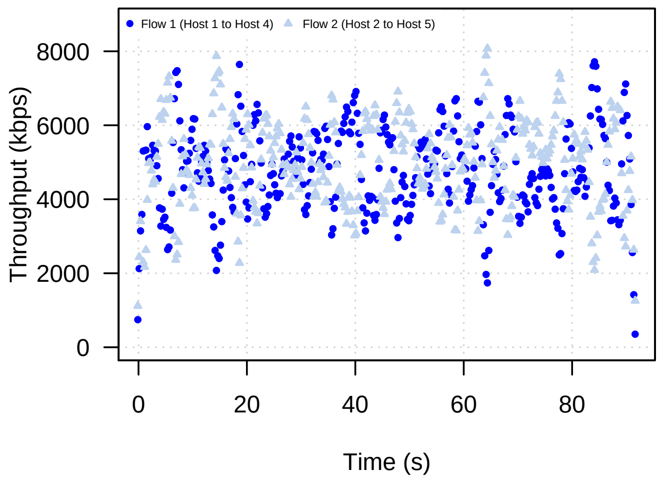


(e) Total RTT vs time – first run

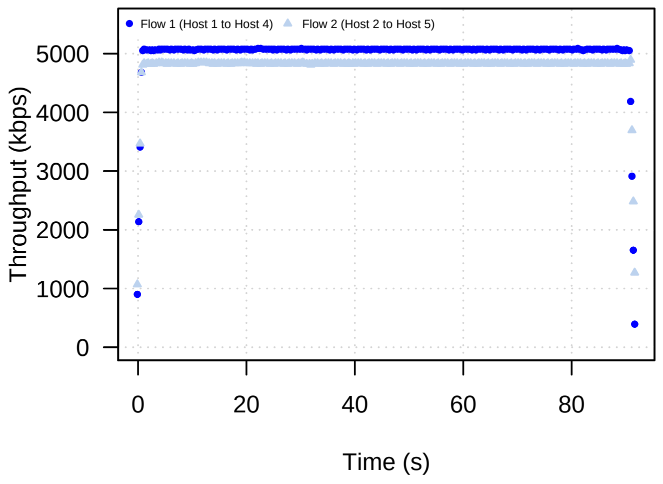


(f) Total RTT vs time – second run

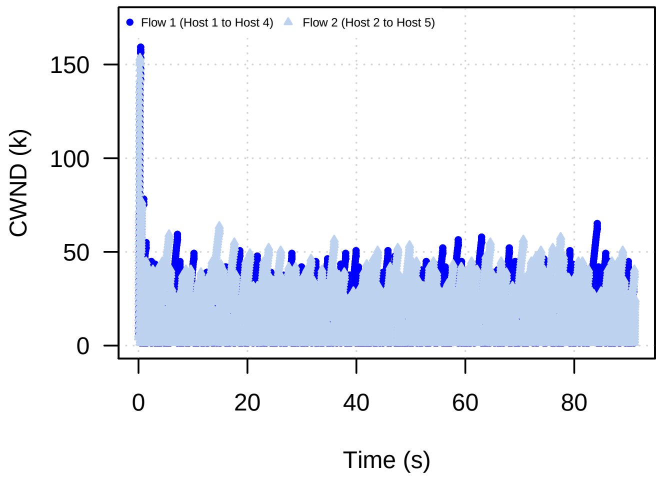
Figure 4: Two separate runs of two concurrent FreeBSD NewReno flows over a 20ms RTT path with 10Mbps rate limit and 180-packet *pfifo* bottleneck buffer



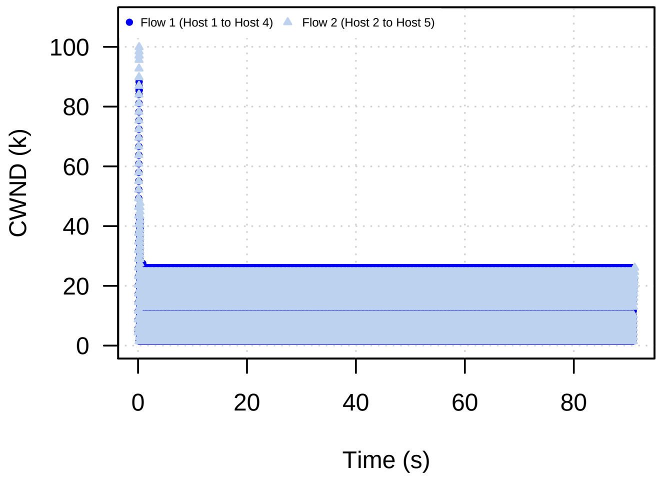
(a) Throughput vs time – PIE



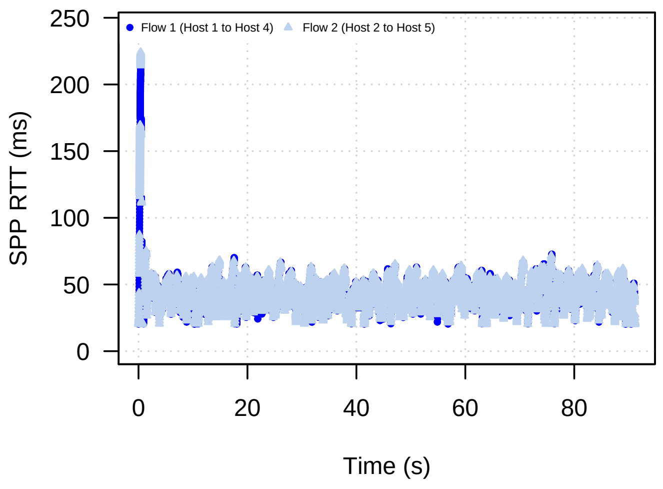
(b) Throughput vs time – fq_codel



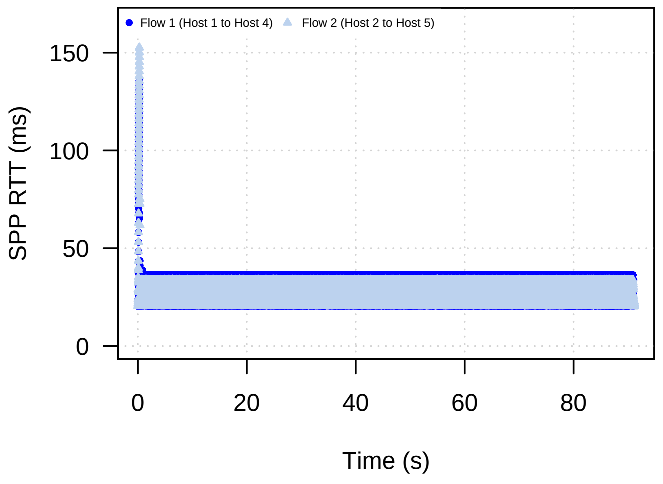
(c) `cwnd` vs time – PIE



(d) `cwnd` vs time – fq_codel



(e) Total RTT vs time – PIE



(f) Total RTT vs time – fq_codel

Figure 5: Two concurrent FreeBSD NewReno flows over a 20ms RTT path with 10Mbps rate limit and 180-packet bottleneck buffer managed by *PIE* or *fq_codel*

B. Two CUBIC flows

Two competing CUBIC flows ran for 90 seconds through the bottleneck between Linux hosts.

1) RTT and throughput at 10Mbps: Figure 6a shows the RTT experienced by two competing Linux CUBIC flows through a 10Mbps bottleneck. As expected, the use of pfifo results in both flows experiencing extremely high queueing delays. However, using either PIE or fq_codel effectively drops the RTT down almost to the path's intrinsic RTT.

Figure 6b shows that the throughput achieved by each flow is broadly the same using all three AQMs. As with the NewReno case, pfifo still creates a more unpredictable sharing of bandwidth, whilst fq_codel appears to provide exceedingly tight bandwidth sharing at 0ms and 10ms OWD.

2) RTT and throughput at 2, 6 and 10Mbps and 20ms RTT: Figure 7a shows the variation of path RTT versus AQM with 2Mbps, 6Mbps and 10Mbps bottleneck rate limits at 10ms OWD (20ms RTT). The dramatic benefit of either PIE or fq_codel at lower bottleneck rates is quite clear. Figure 7b shows the variation in achieved throughput under the same circumstances. Both PIE and fq_codel achieve their RTT improvement with no meaningful impact on performance.

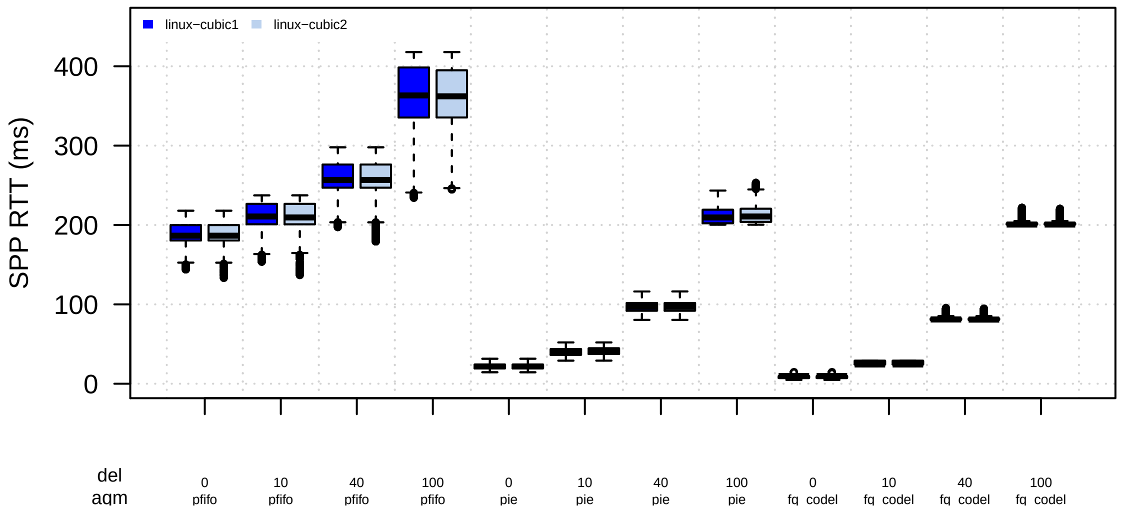
3) Throughput, `cwnd` and RTT versus time – two runs with pfifo: As we did for the NewReno experiments, we illustrate in Figure 8 how CUBIC behaves during two runs of the same experimental conditions – 20ms RTT (10ms OWD), 10Mbps bottleneck rate limit, pfifo queue management of a 180-packet bottleneck buffer.

Figures 8a (throughput vs time) and 8c (`cwnd` vs time) show the two concurrent flows experiencing slightly unbalance sharing of available path capacity over time. In the second run (Figures 8b and 8d) this happens again, but plays out somewhat differently over time. Given that queuing delays are imposed on all traffic sharing the dominant congested bottleneck, the RTT fluctuations (Figures 8e and 8f) of both flows track closely over time within each run.

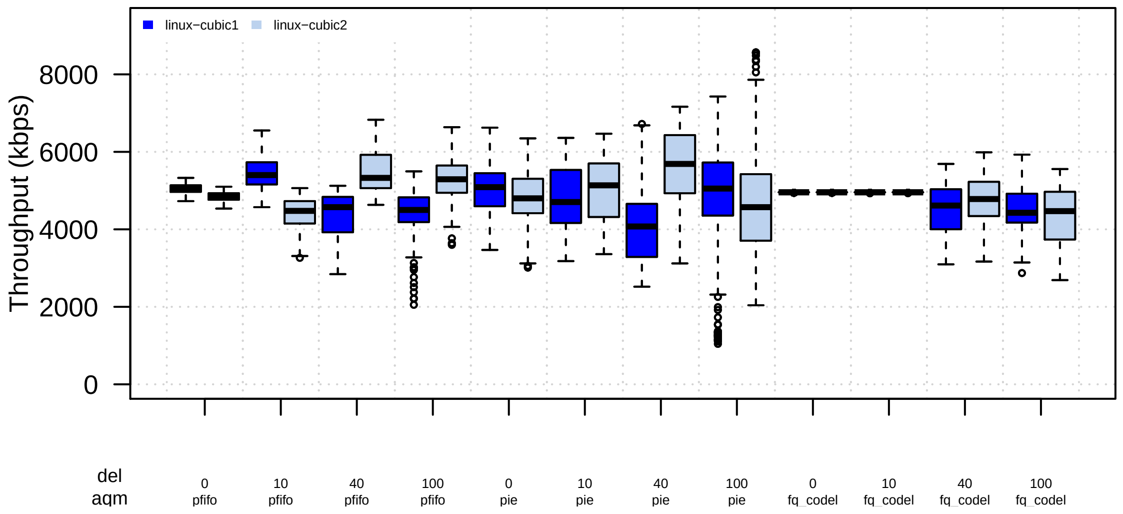
4) Throughput, `cwnd` and RTT versus time – one run with PIE and fq_codel: Next we look briefly at the impact of PIE and fq_codel on flow behaviour over time in Figure 9. Again, the path has a base 20ms RTT (10ms OWD), 10Mbps bottleneck rate limit and 180-packet bottleneck buffer.

Figure 9a show each flow's throughput through the PIE bottleneck fluctuating relatively evenly over time, while Figure 9b shows each flow's throughput being equal and constant through the fq_codel bottleneck. This is consistent with the aggregate results in Figure 6b. At the same time, Figures 9c and 9d show `cwnd` cycling relatively rapidly for PIE, and bouncing around over a very small range with fq_codel.

The nett effect on RTT is shown in Figures 9c and 9d – RTT cycles relatively rapidly around 25ms to 55ms for PIE, and bounces between 20ms and 30ms with fq_codel. In both cases the reduction relative to RTT over a pfifo bottleneck (Figures 4e and 4f) is dramatic.

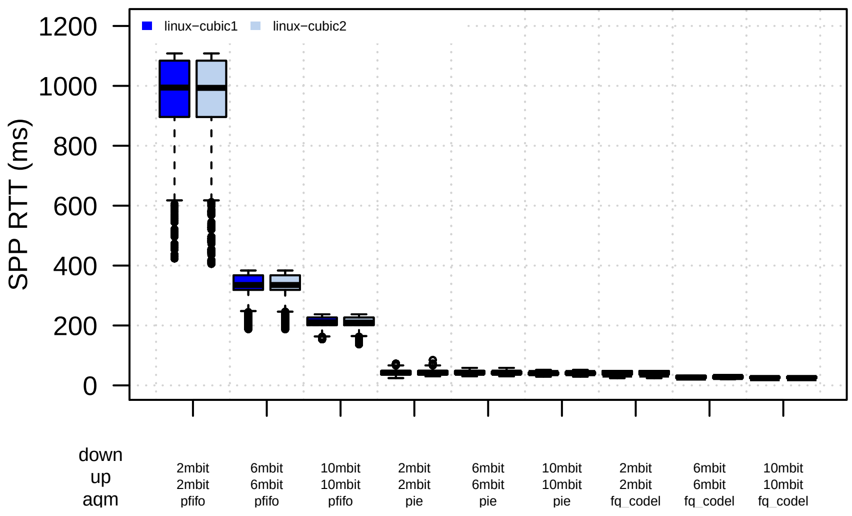


(a) RTT vs OWD and AQM with 10Mbps rate limit

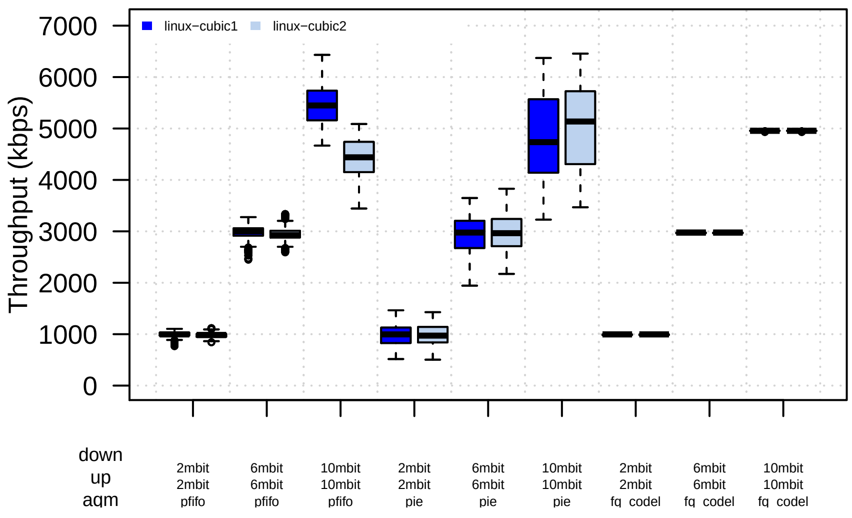


(b) Throughput vs OWD and AQM with 10Mbps rate limit

Figure 6: Two Linux CUBIC flows – 180 packet bottleneck buffer [*del*: OWD in ms, *aqm*: AQM scheme]

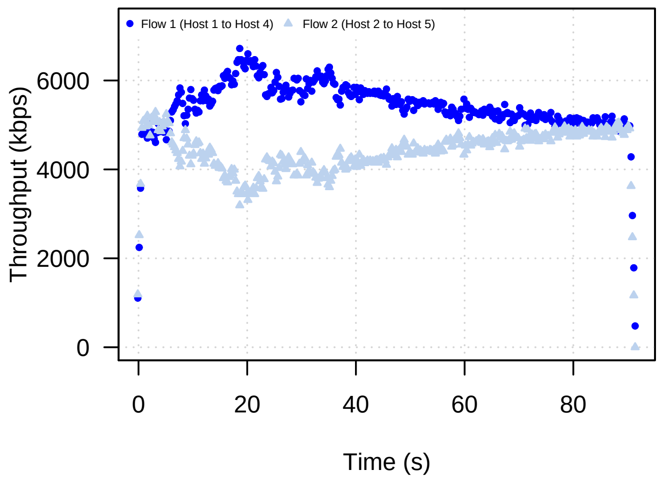


(a) RTT vs rate limit and AQM over 20ms RTT path

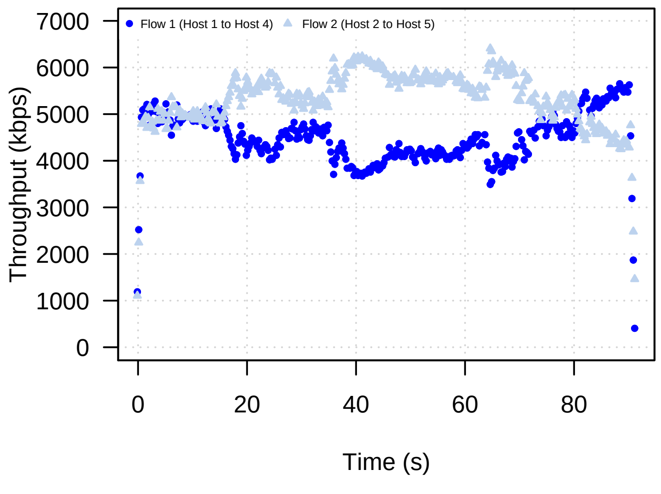


(b) Throughput vs rate limit and AQM over 20ms RTT path

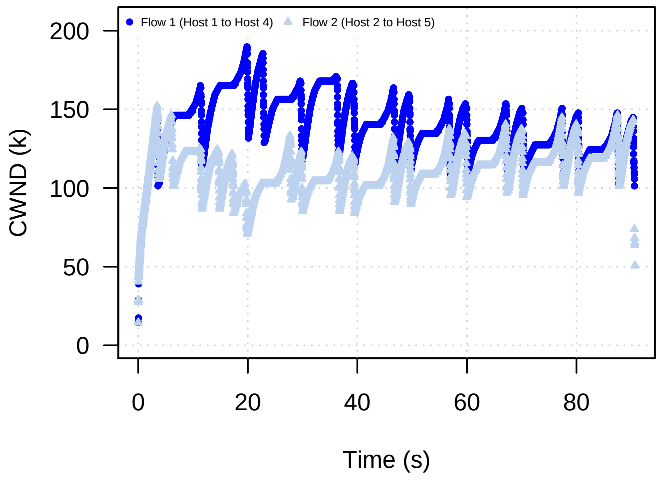
Figure 7: Two Linux CUBIC flows – 180 packet bottleneck buffer, symmetric ‘up’ and ‘down’ speeds



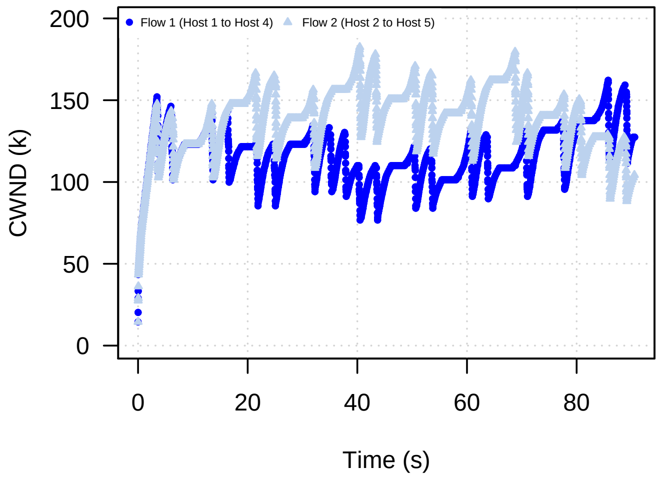
(a) Throughput vs time – first run



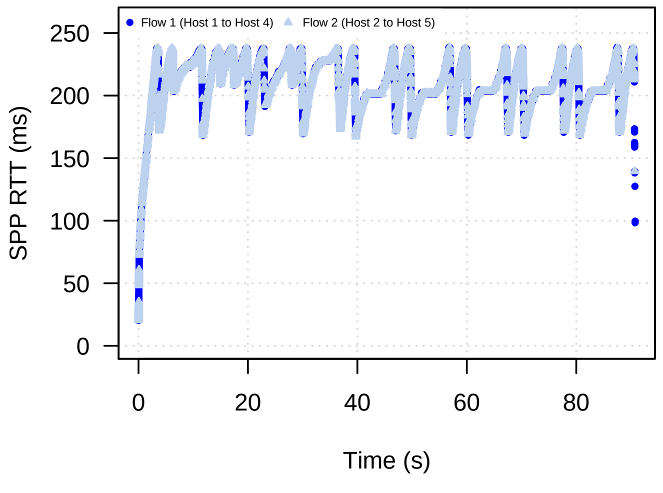
(b) Throughput vs time – second run



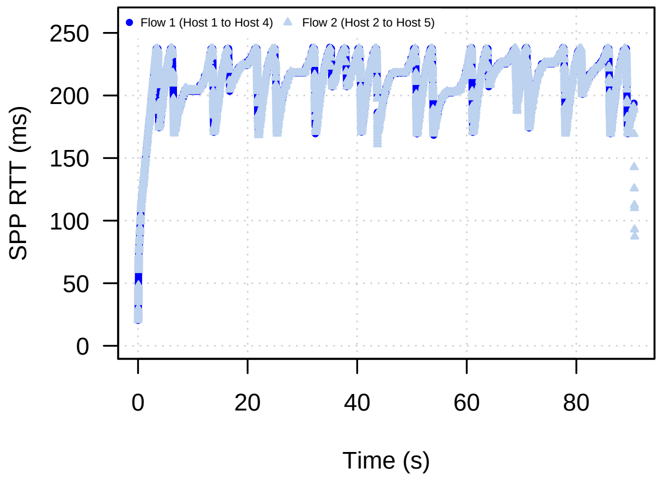
(c) `cwnd` vs time – first run



(d) `cwnd` vs time – second run

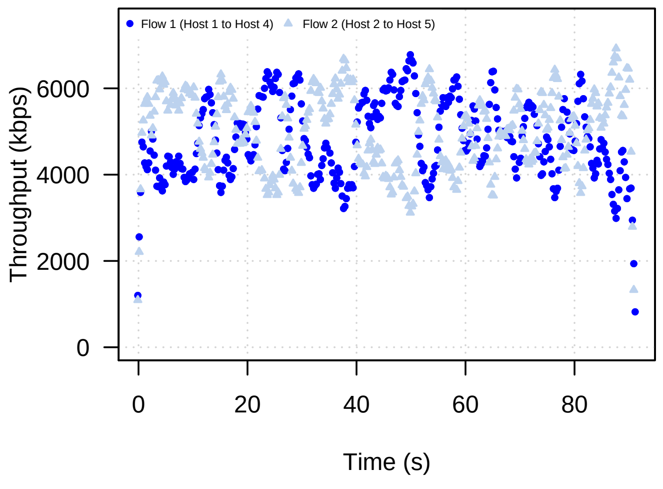


(e) Total RTT vs time – first run

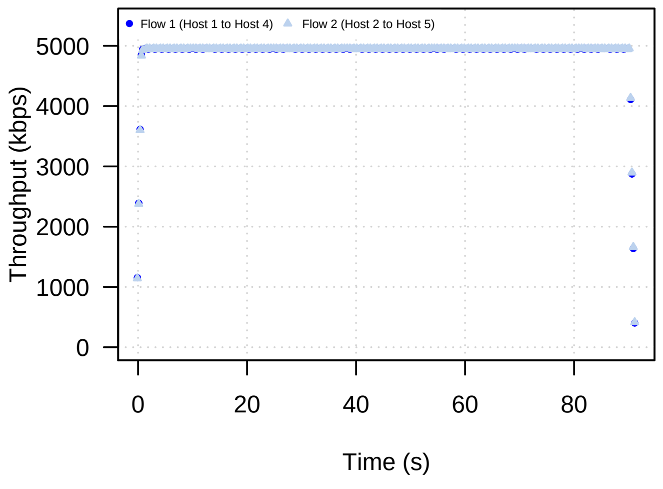


(f) Total RTT vs time – second run

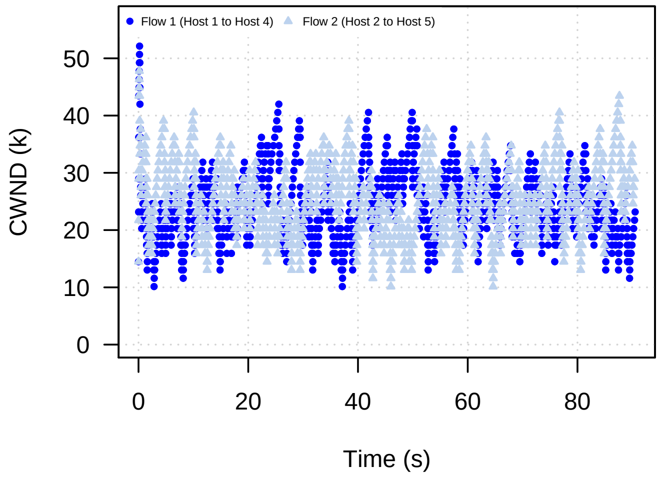
Figure 8: Two separate runs of two concurrent Linux CUBIC flows over a 20ms RTT path with 10Mbps rate limit and 180-packet *pfifo* bottleneck buffer



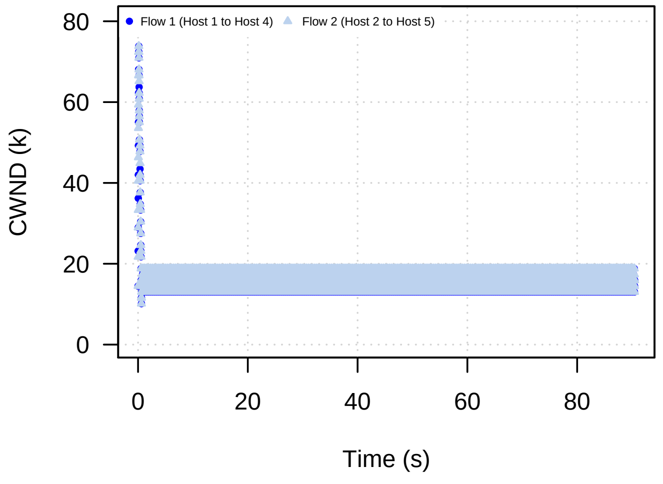
(a) Throughput vs time – PIE



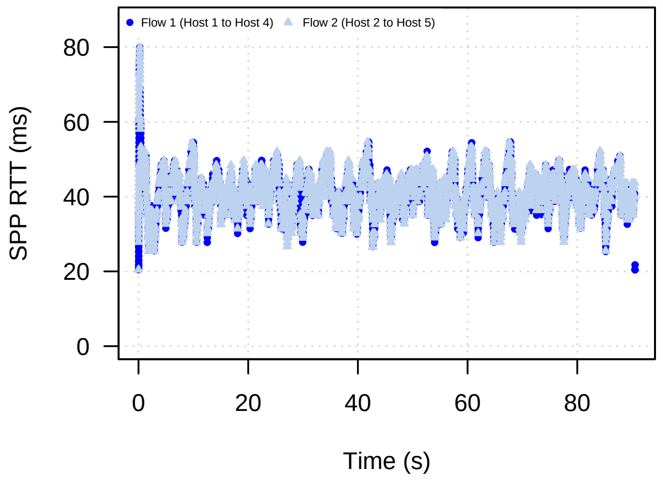
(b) Throughput vs time – fq_codel



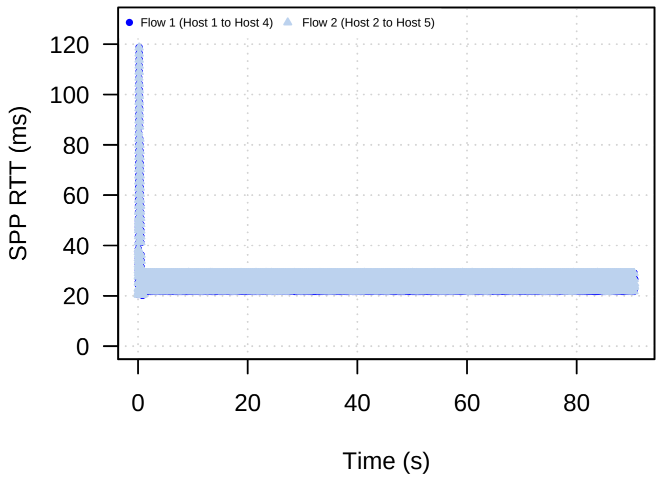
(c) `cwnd` vs time – PIE



(d) `cwnd` vs time – fq_codel



(e) Total RTT vs time – PIE



(f) Total RTT vs time – fq_codel

Figure 9: Two concurrent Linux CUBIC flows over a 20ms RTT path with 10Mbps rate limit and 180-packet bottleneck buffer managed by *PIE* or *fq_codel*

C. Two Win7 default TCP flows

Two competing default TCP flows ran for 90 seconds through the bottleneck between Windows 7 hosts.

Figure 10a shows the RTT experienced by two competing flows through a 10Mbps bottleneck. As expected, the use of pfifo results in both flows experiencing extremely high queueing delays. However, using either PIE or fq_codel effectively drops the RTT down almost to the path's intrinsic RTT.

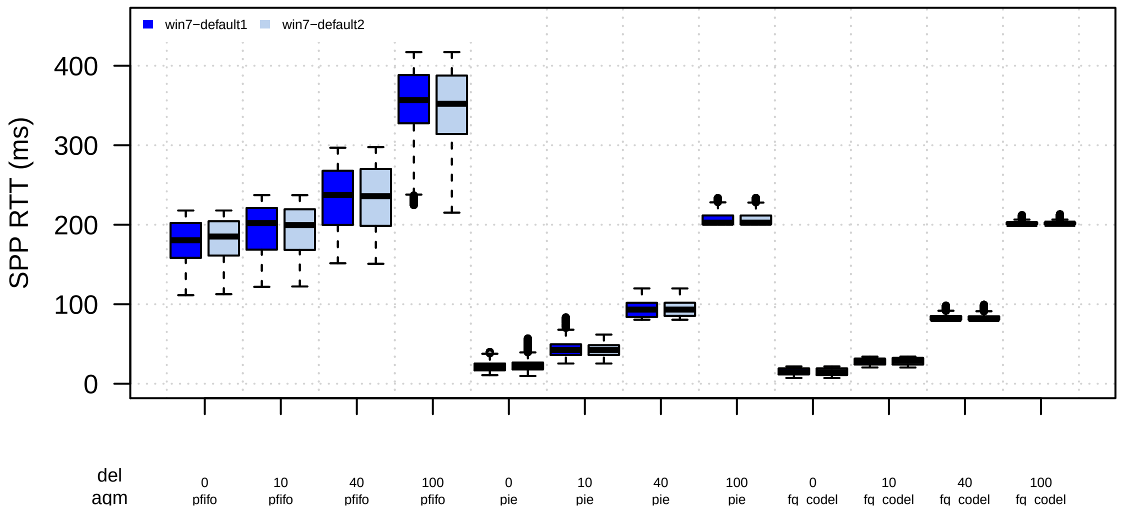
Figure 10b shows that the throughput achieved by each flow is broadly the same using all three AQMs. As with the NewReno case, pfifo still creates a more unpredictable sharing of bandwidth, whilst fq_codel appears to provide exceedingly tight bandwidth sharing at 0ms and 10ms OWD.

D. One NewReno and one CDG flow

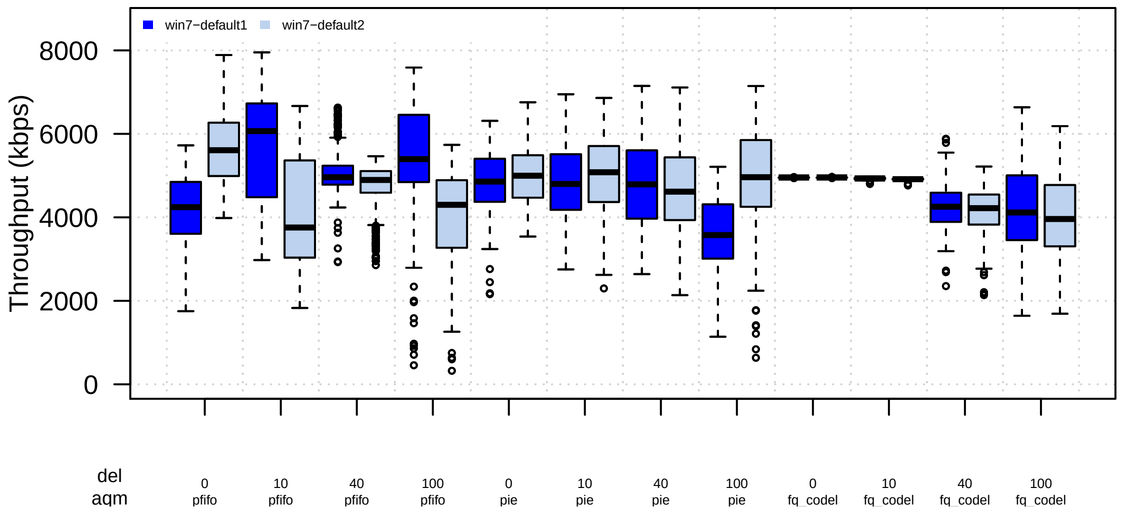
Figure 11a shows the RTT experienced by a FreeBSD NewReno and FreeBSD CDG v0.1 flow competing with each other through a 10Mbps bottleneck. The loss-based TCP flow effectively over-rides CDG's attempts to minimise the shared bottleneck queue's length, so with pfifo queue management both flows experience extremely high queueing delays. However, as seen with the NewReno and CUBIC cases, using either PIE or fq_codel queue management effectively drops the experienced RTT down almost to the path's intrinsic RTT.

Figure 11b shows that choice of AQM also has a noticeable impact on the sharing of capacity between the CDG and NewReno flows. With pfifo, CDG v0.1 is dominated by the NewReno flow. With PIE and fq_codel the CDG v0.1 flow achieves a modest share of the available bandwidth at low OWDs. CDG's poor performance at higher OWDs is consistent with the poor single-flow CDG behaviour also seen in [14] at higher OWDs.[12]

[12]Clarity around the behaviour of CDG v0.1 at higher OWDs is a matter for future work.

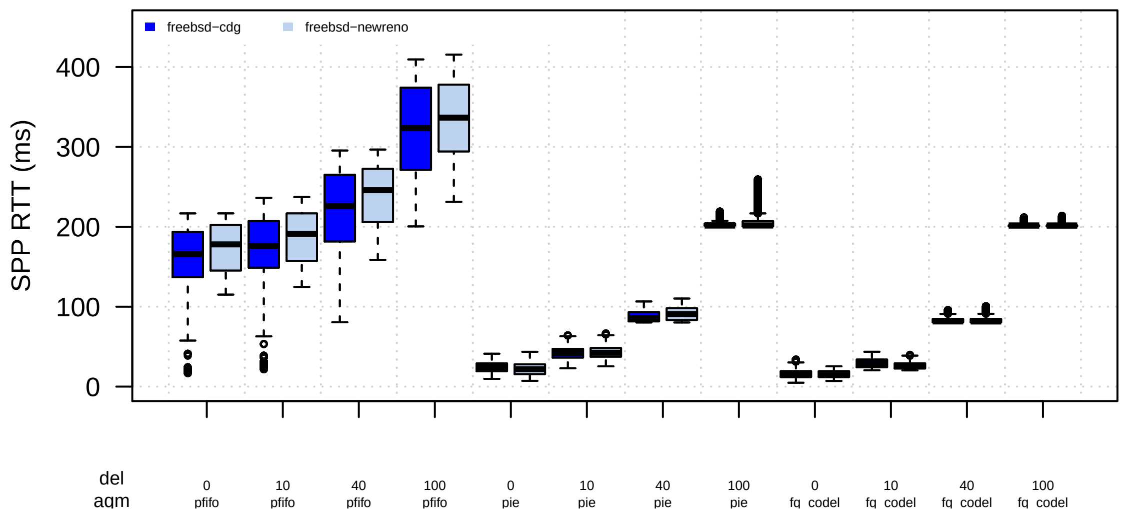


(a) RTT vs OWD and AQM with 10Mbps rate limit

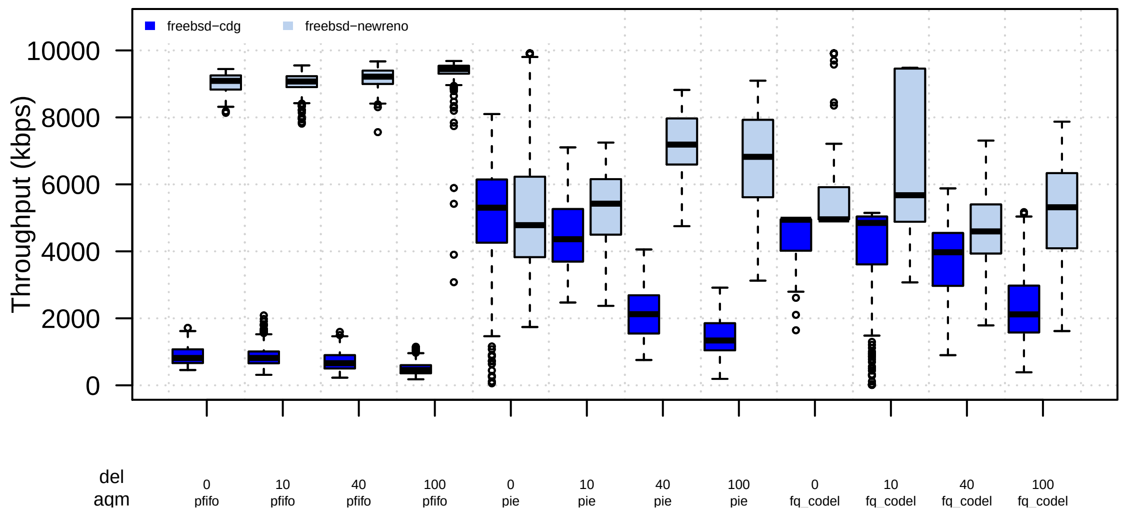


(b) Throughput vs OWD and AQM with 10Mbps rate limit

Figure 10: Two Win7 default TCP flows – 180 packet bottleneck buffer [*del*: OWD in ms, *aqm*: AQM scheme]



(a) RTT vs OWD and AQM with 10Mbps rate limit



(b) Throughput vs OWD and AQM with 10Mbps rate limit

Figure 11: NewReno and CDG v0.1 flow – 180 packet bottleneck buffer [*del*: OWD in ms, *aqm*: AQM scheme]

IV. Impact of AQM on RTT and Throughput with Three concurrent flows

This section summarises the impact of varying the AQM and path parameters – base OWD, bottleneck buffer size and speed – on achievable throughput and overall RTT when three flows share the bottleneck for 60 seconds. We ran three FreeBSD NewReno (section IV-A), three Linux CUBIC (section IV-B) and a mix of FreeBSD NewReno and CDG v0.1 flows (section IV-C).

A. Three NewReno flows

Figure 12a shows the RTT experienced by three competing FreeBSD NewReno flows through a 10Mbps bottleneck. As expected, the use of pfifo results in all three flows experiencing extremely high queueing delays. However, using either PIE or fq_codel effectively drops the experienced RTT down almost to the path's intrinsic RTT.

Figure 12b shows that the throughput achieved by each flow is broadly the same using all three AQMs. Nevertheless, pfifo still creates a more unpredictable sharing of bandwidth, whilst fq_codel appears to provide exceedingly tight bandwidth sharing at 0ms and 10ms OWD.

B. Three CUBIC flows

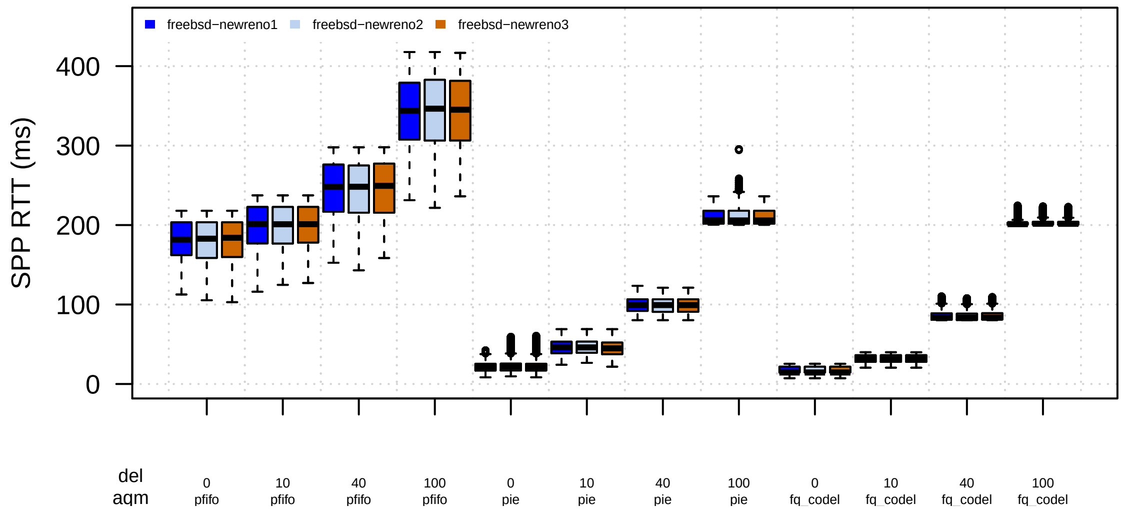
Figure 13a shows the RTT experienced by three competing Linux CUBIC flows through a 10Mbps bottleneck. As expected, the use of pfifo results in all three flows experiencing extremely high queueing delays. However, using either PIE or fq_codel effectively drops the RTT down almost to the path's intrinsic RTT.

Figure 13b shows that the throughput achieved by each flow is broadly the same using all three AQMs. As with the NewReno case, pfifo still creates a more unpredictable sharing of bandwidth, whilst fq_codel appears to provide exceedingly tight bandwidth sharing at 0ms and 10ms OWD.

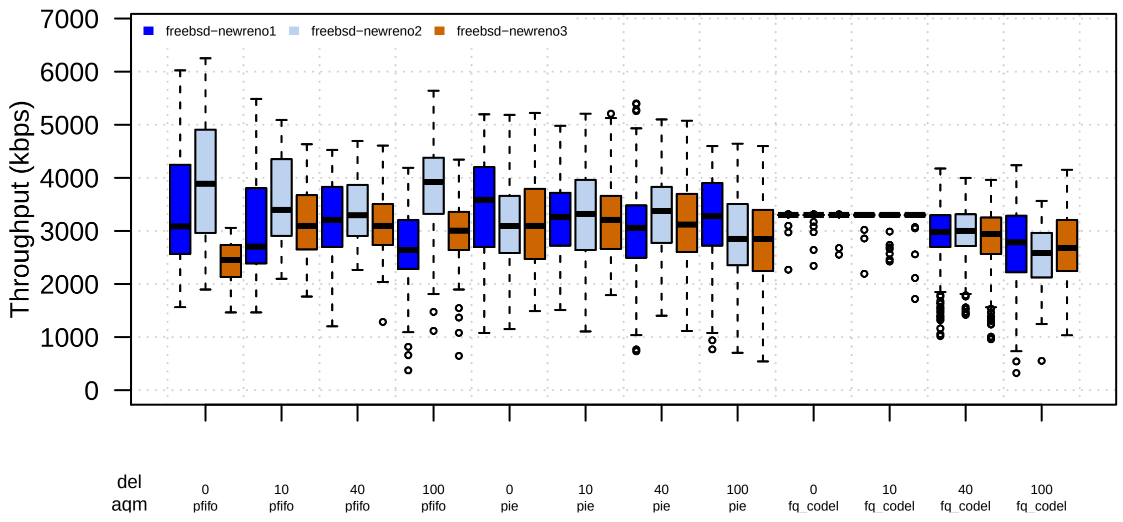
C. One CDG and two NewReno flows

Figure 14a shows the RTT experienced by one FreeBSD CDG v0.1 flow competing with two FreeBSD NewReno flows through a 10Mbps bottleneck. As would be expected, the use of pfifo results in all three flows experiencing extremely high queueing delays. And as seen with the NewReno and CUBIC cases, using either PIE or fq_codel effectively drops the RTT down almost to the path's intrinsic RTT.

Figure 14b shows that choice of AQM has a noticeable impact on the sharing of capacity between the CDG and NewReno flows. With pfifo, CDG v0.1 is dominated by the two NewReno flows. But with PIE and fq_codel, the CDG flow achieves respectible share of the available bandwidth at low OWDs. CDG's poor performance at higher OWDs is consistent with the poor single-flow behaviour also seen in [14] at higher OWDs.

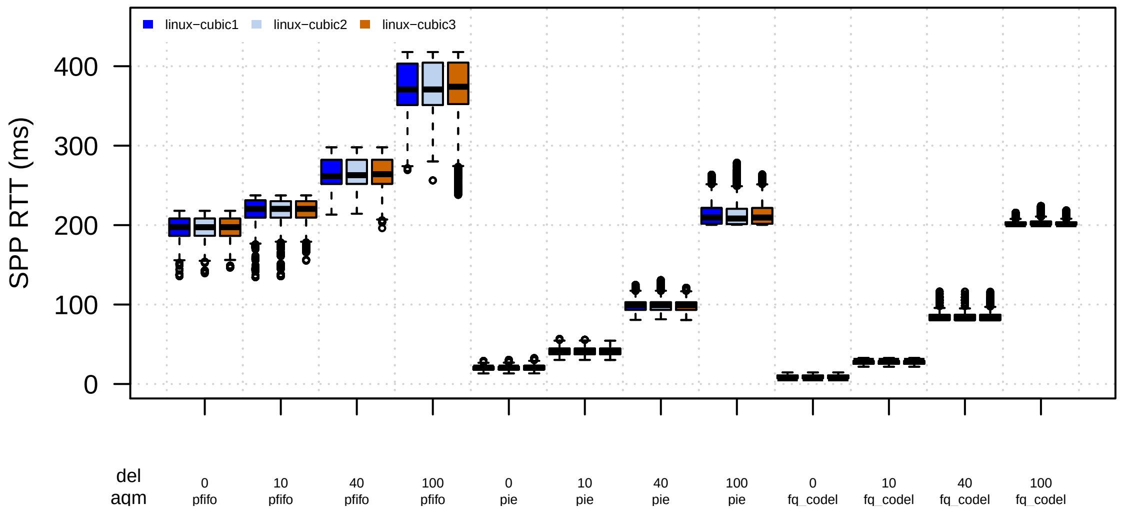


(a) RTT vs OWD and AQM

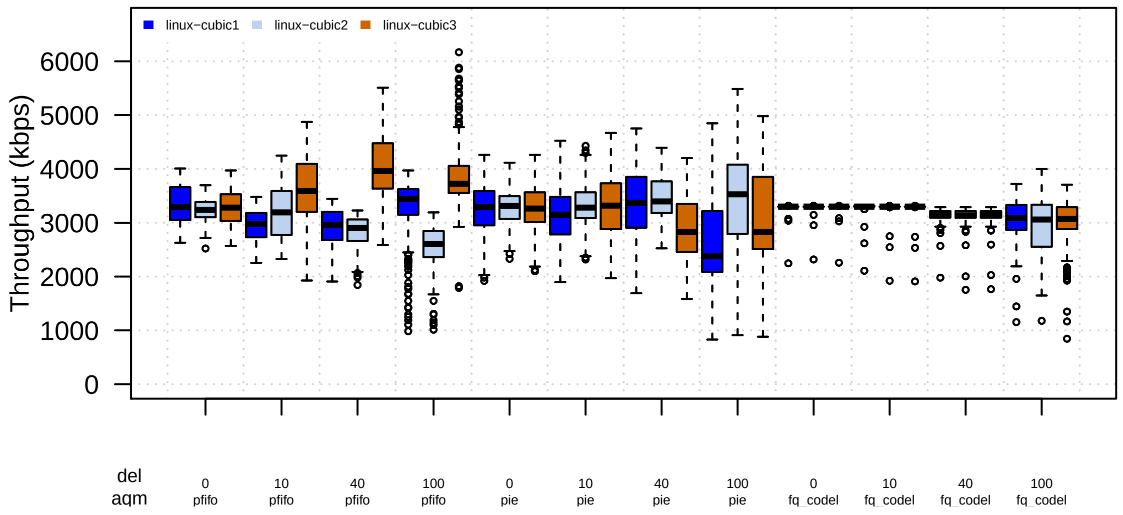


(b) Throughput vs OWD and AQM

Figure 12: Three FreeBSD NewReno flows @ 10Mbps and 180 pkt buffer [OWD (del): 0ms, 10ms, 40ms and 100ms. AQM (aqm): pfifo, PIE and fq_codel]

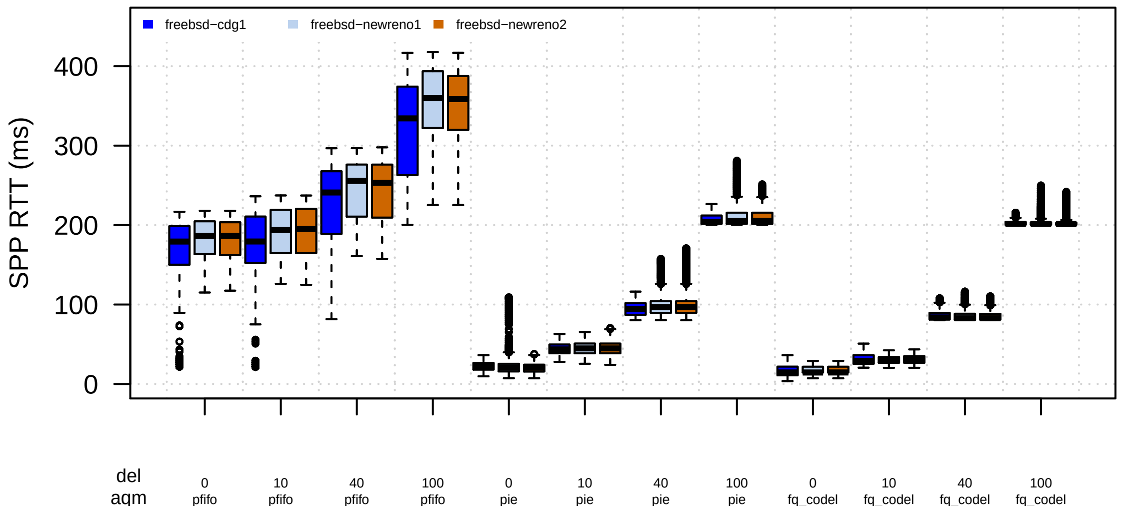


(a) RTT vs OWD and AQM

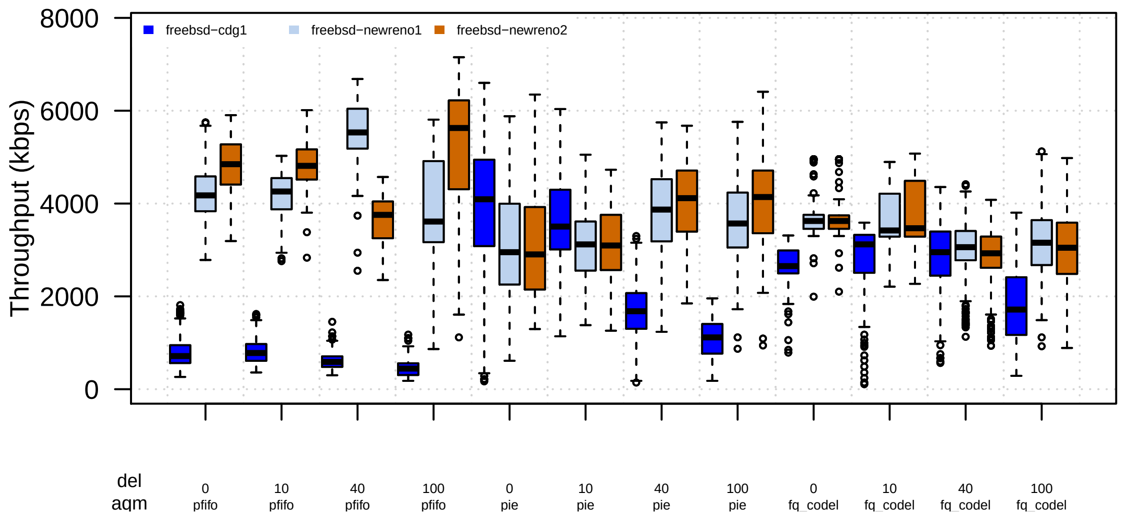


(b) Throughput vs OWD and AQM

Figure 13: Three Linux CUBIC flows @ 10Mbps and 180 pkt buffer [OWD (del): 0ms, 10ms, 40ms and 100ms. AQM (aqm): pfifo, PIE and fq_codel]



(a) RTT vs OWD and AQM



(b) Throughput vs OWD and AQM

Figure 14: Two NewReno flows and one CDG flow @ 10Mbps and 180 pkt buffer [OWD (del): 0ms, 10ms, 40ms and 100ms. AQM (aqm): pfifo, PIE and fq_codel]

V. Conclusions and Future Work

This report describes some simple multi-flow TCP tests run on the CAIA TCP testbed using TEACUP v0.4.8. We provide preliminary observations as to the consequences of two or three TCP flows concurrently sharing a 2Mbps, 6Mbps and 10Mbps bottleneck in the same direction. The bottleneck uses pfifo (tail-drop), PIE or fq_codel queue management schemes with default settings.

As expected, FreeBSD NewReno, Linux CUBIC and Windows 7 default TCP all induced significant queuing delays when sharing a pfifo-based bottleneck. Using PIE and fq_codel at their default settings resulted in observed RTT dropping significantly (almost to the path's intrinsic RTT) without significant impact on achieved throughput.

This report is *only* intended to illustrate the plausibly correct behaviour of our testbed when two and three concurrent flows share a symmetric path. Detailed analysis of why we see these specific results is a subject for future work. Future work will also include varying any PIE and fq_codel parameters that may plausibly be tuned, and explore both asymmetric path latencies and asymmetric path bottleneck bandwidths with concurrent (competing) TCP flows. Future work may also attempt to draw some conclusions about which of the tested TCP and AQM algorithms are 'better' by various metrics.

Acknowledgements

TEACUP v0.4.8 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 HOST TCP STACK CONFIGURATION

For the NewReno 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
- kern.version: 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

net.inet.tcp information from sysctl

- net.inet.tcp.rfc1323: 1
- net.inet.tcp.mssdflt: 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.v6mssdflt: 1220
- net.inet.tcp.cc.available: newreno, cdg
- net.inet.tcp.cc.algorithm: cdg (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
- 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.overflows: 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
- 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 HOST TCP STACK CONFIGURATION

For the Cubic 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 = cubic
- 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`
- `net.ipv4.tcp_window_scaling = 1`
- `net.ipv4.tcp_wmem = 4096 65535 4194304`
- `net.ipv4.tcp_workaround_signed_windows = 0`

tcp_cubic information from /sys/module

- `/sys/module/tcp_cubic/parameters/beta:717`
- `/sys/module/tcp_cubic/parameters/hystart_low_window:16`
- `/sys/module/tcp_cubic/parameters/fast_convergence:1`
- `/sys/module/tcp_cubic/parameters/initial_ssthresh:0`
- `/sys/module/tcp_cubic/parameters/hystart_detect:3`
- `/sys/module/tcp_cubic/parameters/bic_scale:41`
- `/sys/module/tcp_cubic/parameters/tcp_friendliness:1`
- `/sys/module/tcp_cubic/parameters/hystart_ack_delta:2`
- `/sys/module/tcp_cubic/parameters/hystart:1`
- `/sys/module/tcp_cubic/version:2.3`

Appendix C
Windows 7 Host TCP stack configuration

For the default 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
```

APPENDIX D
LINUX ROUTER CONFIGURATION

The bottleneck router is an 8-core machine, patched (as noted in Section V.D of [2]) to tick at 10000Hz for high precision packet scheduling behaviour.

uname

```
Linux newtcp5.caia.swin.edu.au
3.10.18-vanilla-10000hz #1 SMP PREEMPT Fri
Nov 8 20:10:47 EST 2013 x86_64 x86_64 x86_64
GNU/Linux
```

System information from sysctl

- `kernel.osrelease = 3.10.18-vanilla-10000hz`
- `kernel.ostype = Linux`
- `kernel.version = #1 SMP PREEMPT Fri Nov 8 20:10:47 EST 2013`

Bottleneck / AQM configuration

As noted in Section III.H of [1], we use separate stages to apply artificial delay and rate shaping respectively. The selected AQM (pfifo, PIE or fq_codel) is applied on ingress to the rate shaping section.

qdisc when using PIE:

```
qdisc pie 1002: dev ifb0 parent 1:2 limit 90p target
20 tupdate 30 alpha 2 beta 20
```

qdisc when using fq_codel:

```
qdisc fq_codel 1002: dev ifb0 parent 1:2 limit
90p flows 1024 quantum 1514 target 5.0ms interval
100.0ms ecn
```

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] R. Pan, P. Natarajan, C. Piglione, M. Prabhu, V. Subramanian, F. Baker, and B. V. Steeg, "PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem," February 2014. [Online]. Available: http://tools.ietf.org/html/draft-pan-aqm-pie-01

[4] T. Høiland-Jørgensen, "Technical description of FQ CoDel," February 2014. [Online]. Available: https://www.bufferbloat.net/projects/codel/wiki/Technical_description_of_FQ_CoDel

[5] K. Nichols and V. Jacobson, "Controlled Delay Active Queue Management," March 2014. [Online]. Available: http://tools.ietf.org/html/draft-nichols-tsvwg-codel-02

[6] "The Web10G Project." [Online]. Available: http://web10g.org

[7] "PIE AQM source code." [Online]. Available: ftp://ftpeng.cisco.com/pie/linux_code/pie_code

[8] "iproute2 source code." [Online]. Available: http://www.kernel.org/pub/linux/utils/net/iproute2

[9] "iperf Web Page." [Online]. Available: http://iperf.fr/

[10] "NEWTCP Project Tools." [Online]. Available: http://caia.swin.edu.au/urp/newtcp/tools.html

[11] L. Stewart, "SIFTR – Statistical Information For TCP Research." [Online]. Available: http://www.freebsd.org/cgi/man.cgi?query=siftr

[12] 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.

[13] A. Heyde, "SPP Implementation," August 2013. [Online]. Available: http://caia.swin.edu.au/tools/spp/downloads.html

[14] G. Armitage, "Baseline single-flow TCP results for TEACUP v0.4.5 testbed," Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, Tech. Rep. 140502A, 02 May 2014. [Online]. Available: http://caia.swin.edu.au/reports/140502A/CAIA-TR-140502A.pdf