Using Dummynet AQM - FreeBSD's CoDel, PIE, FQ-CoDel and FQ-PIE with TEACUP v1.0 testbed

Jonathan Kua, Rasool Al-Saadi, Grenville Armitage
Centre for Advanced Internet Architectures, Technical Report 160708A
Swinburne University of Technology
Melbourne, Australia
jtkua@swin.edu.au, ralsaadi@swin.edu.au, garmitage@swin.edu.au

***Abstract*—Controlled Delay (CoDel) and Proportional Integral Controller Enhanced (PIE) are emerging active queue management (AQM) schemes designed by the IETF AQM working group to counteract the *bufferbloat* phenomenon. FlowQueue-CoDel (FQ-CoDel) is a hybrid scheme that hashes flow into one of multiple queues, applying CoDel onto each queue and utlises a modified Deficit Round Robin scheduling to share link capacity between queues. A recent implementation of CoDel, FQ-CoDel, PIE and FQ-PIE under FreeBSD's ipfw/dummynet framework – Dummynet AQM v0.2.1 has been released and is now part of the FreeBSD HEAD source tree. Its code base is also part of FreeBSD 10.3-STABLE. This technical report documents the extensions to TEACUP in order to support Dummynet AQM. It also describes how TEACUP v1.0 configures ipfw/dummynet to support FreeBSD AQMs.**

***Index Terms*—FreeBSD, AQM, Dummynet, ipfw, CoDel, FQ-CoDel, PIE, FQ-PIE, TEACUP**

I. Introduction

Network routers use buffers to enhance routing performance and increase overall throughput by absorbing packet bursts and reducing packets drop. In recent years, routers have used large buffers due to low memory prices, causing the well-known *bufferbloat* phenomenon– high latency in congested bottlenecks if traditional first-in-first-out (FIFO) queuing is used [1]. In recent years, active queue management (AQM) schemes have been proposed to counteract the effects of bufferbloat. AQM controls queue length by dropping or marking packets from bottleneck buffer when it becomes full or queue delay becomes over a threshold value. Some AQMs, such as RED (Random Early Detection) and its variations, use queue occupancy as an indicator of how much a queue is congested.

CoDel (Controlled Delay) and Proportional Integral controller Enhanced (PIE) are examples of a modern AQM designed to remedy these limitations by using queue delay, instead of queue length, to indicate standing queues. FQ-CoDel is a hybrid scheme – flows are assigned to one of a pool of internal queues, each active queue runs an independent instance of CoDel, and a modified Deficit Round Robin (DRR) scheduler shares outbound link capacity between the active queues. Implementations of CoDel and FQ-CoDel have existed in the Linux kernel since version 3.5 and PIE since version 3.14.

Recently, an independent implementation of CoDel, FQ-CoDel, PIE and FQ-PIE under FreeBSD's ipfw/dummynet framework [2] – Dummynet AQM v0.2.1 has been released and is now part of the FreeBSD HEAD source tree[1].

TCP Experiment Automated Controlled Using Python (TEACUP) [3] is a software for running automated experiments. As of v1.0, TEACUP can only configure FIFO and RED queuing disciplines when using a FreeBSD-based bottleneck router. With the recent release of Dummynet AQM v0.2.1, we have enabled TEACUP to configure CoDel, PIE, FQ-CoDel, FQ-PIE AQMs when using a FreeBSD router, patched with Dummynet AQM v0.2.1.

This technical report is an extension of [4]. Section II describes the TEACUP testbed setup and the test conditions we use for our experiments. In Section III, we document the steps needed to patch TEACUP to run automated experiments with with FreeBSD AQM schemes, and confirm their operation in Section IV. We explain how TEAUCP instantiate Dummynet configurations in Section V.

[1] https://svnweb.freebsd.org/base?view=revision&revision=300779

II. TEACUP TESTBED SETUP AND TEST CONDITIONS

Our experiment trials involve three concurrent TCP connections each pushing data downstream through a single bottleneck queue for 60 seconds. Each flow has different emulated path delay, sharing the same rate-limited bottleneck speed with FIFO or AQM schemes.

In this section, we document the testbed setup, operating systems, TCP Congestion Control algorithms, path conditions and measurement techniques.

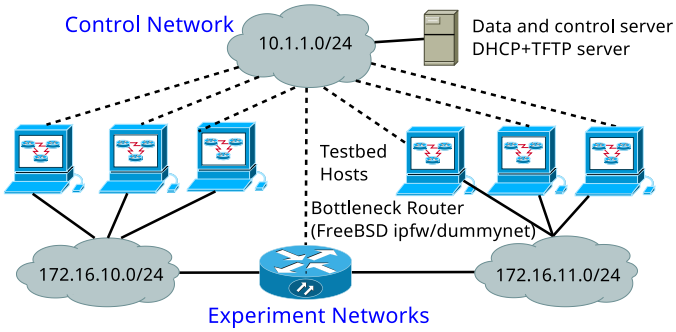


Figure 1: TEACUP testbed

A. Hosts and router

Figure 1 shows our TEACUP-based testbed. The router uses FreeBSD10.1-RELEASE to provide a configurable bottleneck between client(s) on 172.16.10.0/24 and server(s) on 172.16.11.0/24. All end-hosts run 64-bit Linux openSUSE 12.3 (kernel 3.17.4) and TCP CUBIC. The router's Dummynet module provides rate limiting, base RTT emulation and Dummynet AQM v0.2.1 provides AQMs – CoDel, FQ-CoDel, PIE, FQ-PIE. In the experiments described in this report, hosts on 172.16.11.0/24 are sending traffic traffic to hosts on 172.16.10.0/24.

Under FreeBSD router, separate Dummynet pipes are realised, which separately provide rate shaping, delay/loss emulation, and queuing disciplines. ipfw rules are used to redirect packets to the pipes based on the specified source and destination IP parameters. The pipe number is automatically determined by TEACUP.

B. Emulated path and bottleneck conditions

The bottleneck router uses ipfw/dummynet framework to concatenate an emulated path with specific one way delay (OWD) and provide rate shaping to emulate bottleneck with a specific rate.

1) Bottleneck AQM: We repeated each trial using FIFO, Dummynet AQM v0.2.1 implementation of CoDel, FQ-CoDel, PIE, FQ-PIE queuing disciplines in turn to manage the bottleneck queue. We set the total buffer size to either 100 packets (FIFO) or 1000 packets (AQM).

As AQMs are intended to used with minimal operator control or parameters tuning, we use all AQMs at their default settings (except the configurable bottleneck buffer size) as recommended by CoDel, PIE and FQ-CoDel IETF Internet Drafts [5]–[7].

2) Path conditions: The experiment results in this report covers the following emulated path conditions:

- No intrinsic packet loss rate
- One way delay (OWD): Flow 1 - 5ms, Flow 2 - 10ms; Flow 3 - 15ms
- Bottleneck bandwidth: 12Mbps downstream/1Mbps upstream
- Bottleneck buffer sizes: 100 (FIFO) or 1000 (AQM) packets
- Explicit Congestion Notification (ECN) disabled on all hosts

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 Round Trip Time (RTT) is twice the configured OWD.

C. Traffic generator and logging

Each concurrent bulk TCP flow was generated using `iperf`. Each 60-second flow was started 20 seconds apart between the following host pairs:

- Flow 1: Host 1 $\rightarrow$ Host 4 at $t = 0$
- Flow 2: Host 2 $\rightarrow$ Host 5 at $t = 20$
- Flow 3: Host 3 $\rightarrow$ Host 6 at $t = 40$

Packets from all flows traverse the bottleneck router in the same direction. TCP connection statistics were logged using SIFTR [8] under FreeBSD and Web10G [9] under Linux. Packets captured at both hosts with `tcpdump` were used to calculated non-smoothed end to end RTT estimates using CAIA's passive RTT estimator, Synthetic Packet Pairs (SPP) [10].

D. Measuring throughput

Instantaneous' throughput is an approximation derived from the actual bytes transferred during constant 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 from one measurement interval to the next [11].

Throughput calculation of the experiments presented in this report uses a window two seconds wide, sliding forward in steps of 0.5 second.

III. Applying the patch

We provide a patch for TEACUP v1.0 to support Dummynet AQM. This patch allows TEACUP users to configure the FreeBSD-based bottleneck router to use Dummynet AQM in a similar way as configuring Linux router[2] when running automated experiments. Internally, the patched TEACUP will create and configure FreeBSD's ipfw/dummynet objects (pipe, queue and sched) depending on the selected AQM in TEACUP's queues configurations.

A. Patching TEACUP v1.0

The patch modifies `init_dummynet_pipe()` function in `routersetup.py` file to allow TEACUP to configure FreeBSD AQM correctly. Additionally, it modifies `init_dummynet()` function in `hostsetup.py` to increase the limit of Dummynet bottleneck buffer size.

To apply the patch, first clone TEACUP v1.0 source code (revision 21b454[3]) from the Mercurial repository on SourceForge as follows:

```
hg clone -r 21b454 http://hg.code.sf.net/
   p/teacup/code teacup-code
```

Then, download the patch[4] and apply it as follows:

1) Extract the patch file:

```
tar -xvf teacup-dummynet-aqm-patch
   -0.1.tgz -C teacup-code/
```

2) Apply the patch:

```
cd teacup-code
patch -p1 < teacup-dummynet-aqm-patch
   -0.1/teacup-aqm.patch
```

IV. Experiment Results

To illustrate the TEACUP's ability to configure various queue management schemes (FIFO, CoDel, PIE, FQ-CoDel, FQ-PIE) and apply different delays to different flows, we present the experiment results from test cases as described in Section II, where three staggered-start TCP bulk transfers compete through a single rate-limited 12Mbps downstream/1Mbps upstream bottleneck, with 10ms, 20ms, 30ms base RTT respectively.

[2]For more information on how Linux router is configured by TEACUP, see Section IV-B of [3].

[3]This is the latest revision at the time of writing.

[4]The patch can be downloaded from http://caia.swin.edu.au/tools/teacup/downloads/teacup-dummynet-aqm-patch-0.1.tgz

A. Traditional FIFO

Figure 2 shows the throughput, `cwnd` and RTT vs time for FIFO. These figures illustrate the expected high induced queuing delays, and modest capacity sharing of FIFO.

B. Single-queue CoDel and PIE

CoDel and PIE also provide modest capacity sharing, slightly better than FIFO, but significantly improved (lower) RTTs for all three flows. PIE's statistical nature allows slightly higher buildup of queuing delay than CoDel. Figure 3 shows the throughput, `cwnd` and RTT vs time for CoDel and PIE.

Figures 3e and 3f clearly show how each flow gets different emulated base RTT based on TEACUP queues configurations.

Figures 3a and 3b, show that CoDel's throughput becomes slightly lower than PIE throughput when the RTT is 30ms. This is because CoDel emulates a very small queue size while PIE emulates slightly larger queue size.

C. Multi-queue FQ-CoDel and FQ-PIE

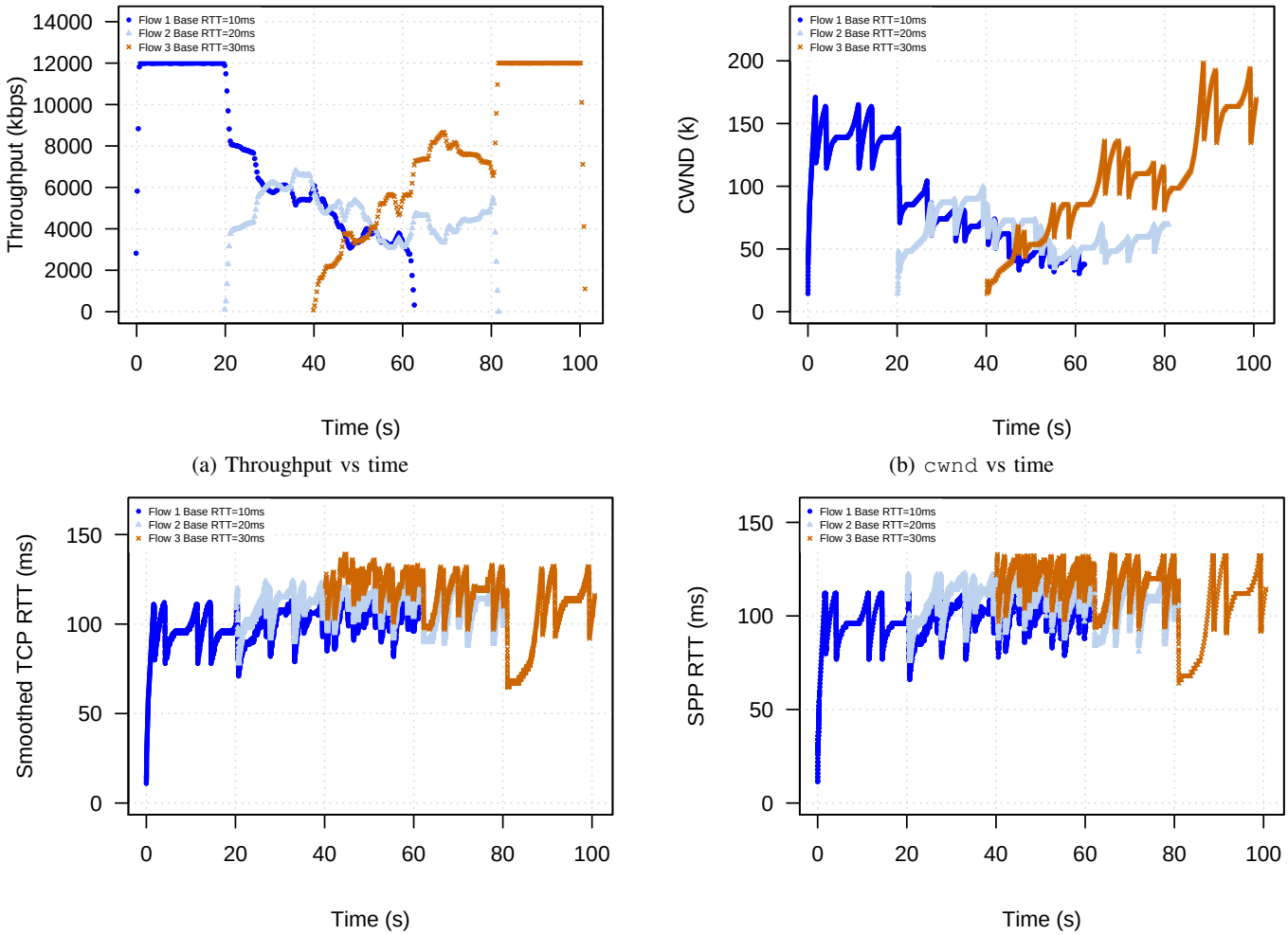
FQ-CoDel and FQ-PIE provide excellent capacity sharing and low queuing delay for all three flows. The excellent capacity sharing of these AQMs is achieved due to the FlowQueue scheduling algorithm that hashes each flow in a separate queue and gives higher priority for new flows (DRR scheduler operation). Significantly lower queuing delays are achieved because CoDel or PIE AQM instance are applied to each sub-queue. Figure 4 shows the throughput, `cwnd` and RTT vs time for FQ-CoDel and FQ-PIE experiments.

V. TEACUP ipfw/dummynet AQM configurations

In multi-flow experiments, TEACUP is capable of configuring same delays for all flows or apply different delays for different flows through a single bottleneck. Here we describe how TEACUP configures ipfw/dummynet for FreeBSD AQM experiments as presented in Section IV.

A. ipfw/dummynet pipes and AQMs

Dummynet has three basic object types (link, queue and scheduler) that are required for constructing a functional pipe. Link object provides traffic shaping, network delay and loss emulation functions. Queue object has a configurable-size buffer that is used to store packets temporarily to absorb network bursts, and managed by



(a) Throughput vs time

(b) `cwnd` vs time

(c) Smoothed RTT vs time

(d) SPP RTT vs time

Figure 2: Three staggered-start Linux CUBIC bulk TCP flows across 100-packet FreeBSD FIFO bottleneck

a queue management scheme (Droptail, RED, CoDel or PIE). Scheduler object is responsible for fetching packets from queues and send them through the connected link. Dummynet provides many scheduling techniques such as FIFO, WF2Q+ and QFQ in addition to the new FQ-CoDel and FQ-PIE AQM/scheduler hybrid schemes. When a new pipe is created (using `ipfw pipe n config` command), Dummynet creates all the three objects and connects them together. Then the pipe can be added to `ipfw` rules. Figure 5 shows a simple Dymmynet pipe with single queue and FIFO scheduler.

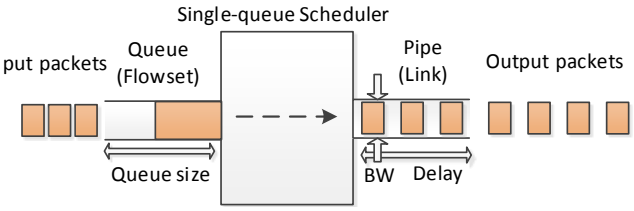
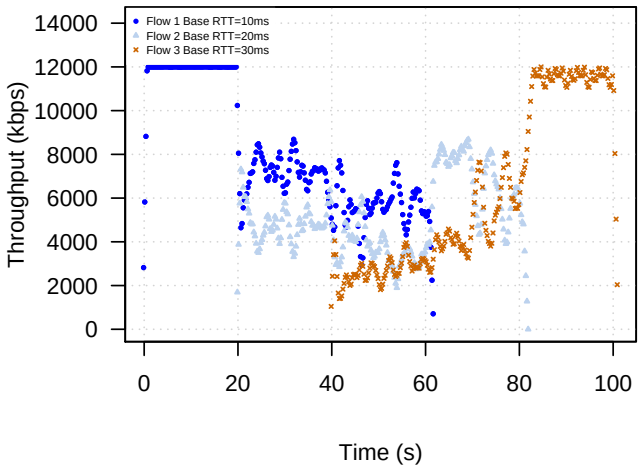
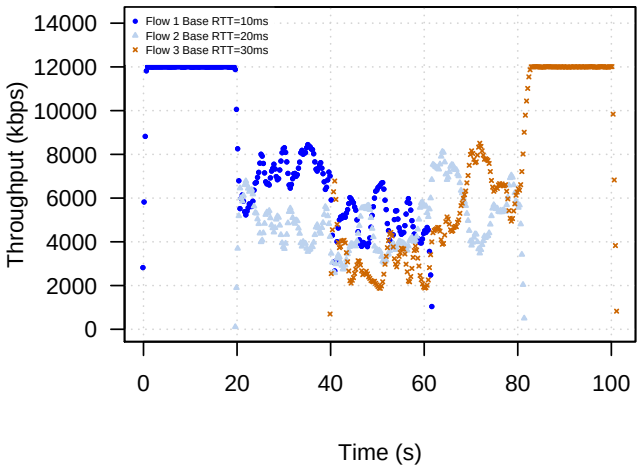


Figure 5: A Simple Dummynet Pipe

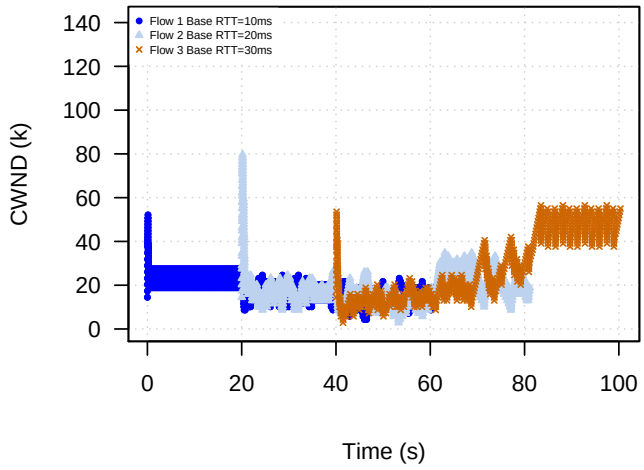
In addition to the single queue configuration, multi-queue scheduler, such as QFQ and RR, can serve multiple queues depending on the scheduling algorithm. Each queue can be configured with different size and managed by different AQM. Figure 6 shows the connections between queues, a multi-queue scheduler and a link.



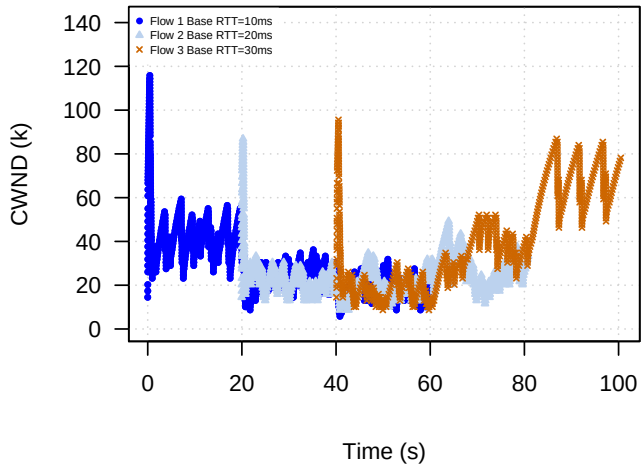
(a) Throughput vs time, CoDel



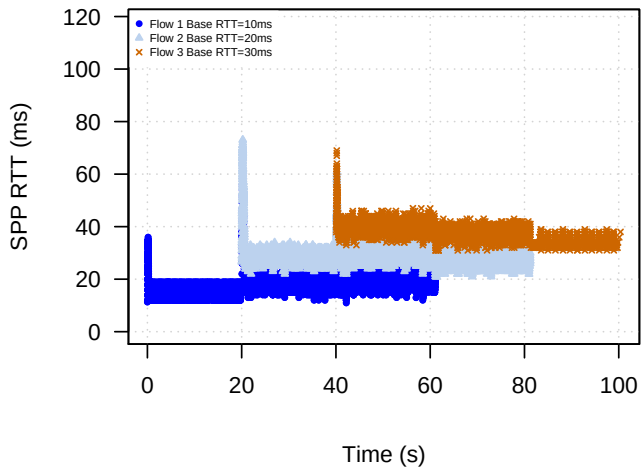
(b) Throughput vs time, PIE



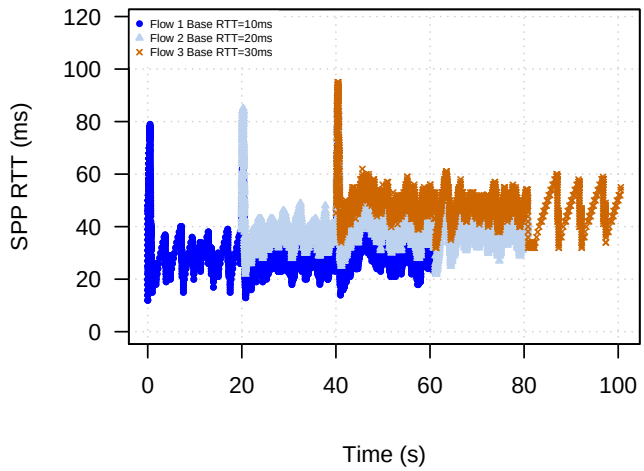
(c) `cwnd` vs time, CoDel



(d) `cwnd` vs time, PIE

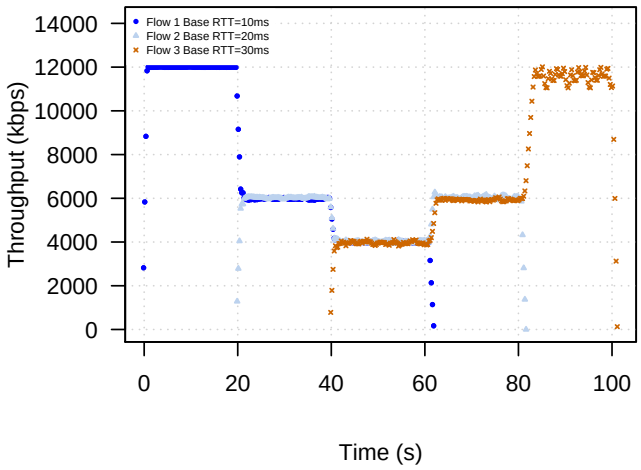


(e) RTT vs time, CoDel

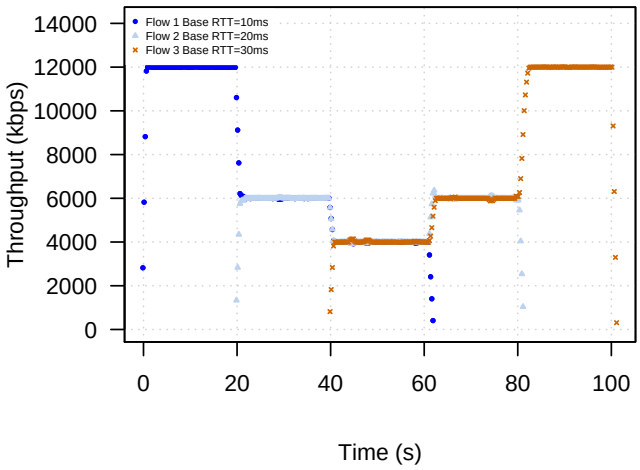


(f) RTT vs time, PIE

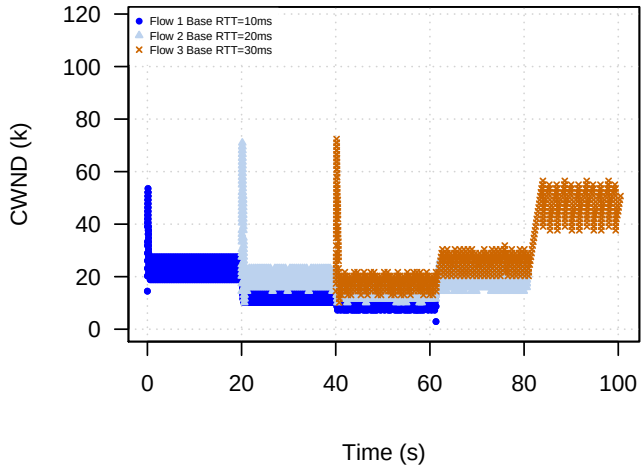
Figure 3: Three staggered-start Linux CUBIC bulk TCP flows across 1000-packet CoDel and PIE bottleneck



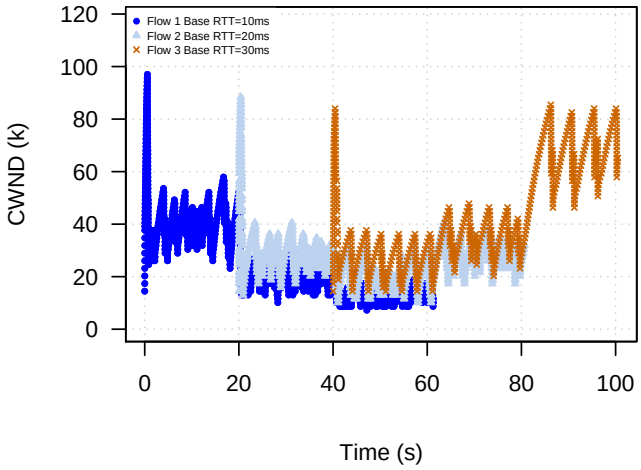
(a) Throughput vs time, FQ-CoDel



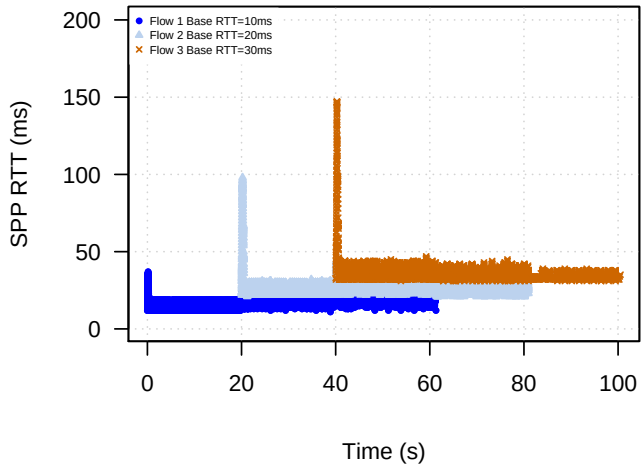
(b) Throughput vs time, FQ-PIE



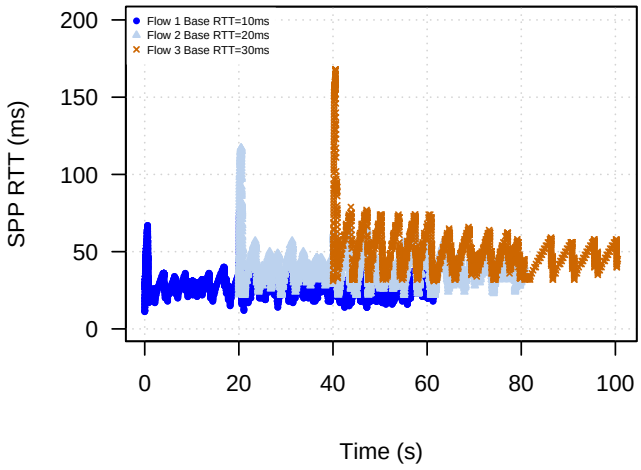
(c) `cwnd` vs time, FQ-CoDel



(d) `cwnd` vs time, FQ-PIE



(e) RTT vs time, FQ-CoDel



(f) RTT vs time, FQ-PIE

Figure 4: Three staggered-start Linux CUBIC bulk TCP flows across 1000-packet FQ-CoDel and FQ-PIE bottleneck

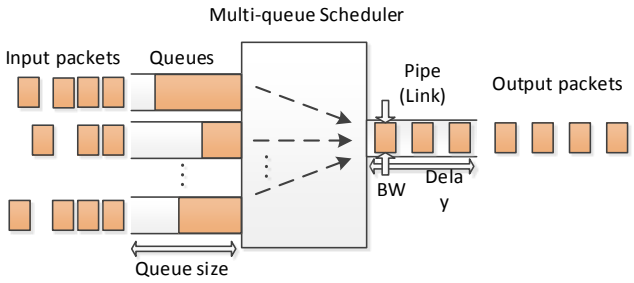


Figure 6: Dummynet pipe with a multi-queue scheduler

In more advanced schemes, as in FQ-CoDel and FQ-PIE AQM/scheduler hybrid scheme, the scheduler implemented as single-queue scheduler with multiple internal sub-queues each managed by AQM (CoDel or PIE) instance. Figure 7 illustrates the relationship between a queue, a FQ-CoDel/FQ-PIE scheduler with its sub-queues and a link. It should be noted that a sub-queue does not have a fixed buffer size but there is a limit for the total number of packets that can be buffered by the whole sub-queues. To configure a pipe with FQ-CoDel/FQ-PIE, a queue, scheduler and link should be created separately and be connected together, then the created queue should be added to `ipfw` rules.

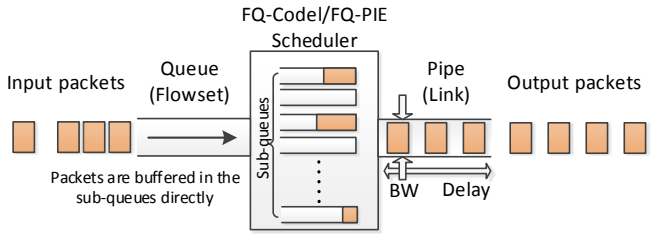


Figure 7: FQ-CoDel/FQ-PIE AQM/Scheduler in Dummynet

B. Configuring same delays for all flows

When a FreeBSD router is used, TEACUP generates and runs `ipfw` commands based on router queues configurations in `config.py`. For each tuple in queues configurations, TEACUP creates a Dummynet pipe with a specific rate limiting, delay and loss emulation. If `queue_disc` argument is a pure queue management name i.e. 'fifo', 'red', 'codel' or 'pie', TEACUP set queue size limit (if any) to the pipe and adds it to `ipfw` rule. If `queue_disc` argument is 'fq_codel' or 'fq_pie', TEACUP creates a queue and fq_codel or fq_pie scheduler, and set queue size limit to the scheduler[5]. After that, the created queue is added to `ipfw` rules.

In the next sub-sections, we show two TEACUP router queues configuration examples to demonstrate how TEACUP configures pure AQM pipes and scheduler/AQM hybrid schemes with the same delays for all flows [6]. In our examples, we used CoDel as pure AQM and FQ-CoDel for the hybrid scheme since same principles are applied to PIE and FQ-PIE.

1) Example of configuring CoDel pipes: In this example, we want to configure two ipfw/dummynet pipes with 10ms emulated delay, no emulated losses, 1000 packets queue size and CoDel AQM. The first pipe has 1Mbit/s traffic shaping and used for traffic passes from '172.16.10.0/24' to '172.16.11.0/24' subnets. The second one has 12Mbit/s traffic shaping and used for traffic passes from '172.16.11.0/24' to '172.16.10.0/24' subnets. Figure 8 shows `TPCONF_router_queues` TEACUP configuration for this example.

As described in section V-B, TEACUP translates the router queues configurations to ipfw/dummynet configurations. Figure 9 (`ipfw pipe show` command) shows `ipfw` rules and Dummynet pipes configured by TEACUP. In this figure, we can see a pipe number 0001 is created with 1Mbit/s rate limit and 10ms emulated delay based on the first tuple of `TPCONF_router_queues`. Dummynet creates a queue (q131073) with 1000 packets queue size limit and CoDel AQM, and connects it to sched 65537 which is a simple FIFO scheduler. For second tuple, a pipe number 0002 is created in the same manner as for the first pipe. Figure 9 (`ipfw show` command) shows ipfw rules for this example. We can see rule number 00100 is created for the first tuple of `TPCONF_router_queues` and rule number 00200 is created for the second tuple.

2) Example of configuring FQ-CoDel pipes: In this example we have configurations similar to the first example except we want to use FQ-CoDel AQM instead of Codel. Figure 10 shows `TPCONF_router_queues` of the example. As described in section V-B, TEACUP interprets 'fq_codel' and 'fq_pie' queues in a different way than in pure AQM. Figure 11 shows `ipfw` rules and Dummynet objects configured by TEACUP. We describe the configuration for the first tuple as the idea is the same for the second tuple.

[5]Queue size limit is for whole sub-queues that created by fq_codel/fq_pie scheduler.

[6]Experiment results generated from these configurations are not presented in this report. For similar experiment results, see [4], [12].

```
TPCONF_router_queues = [

    ('1', " source='172.16.10.0/24', dest='172.16.11.0/24', delay=10, "
     " loss=0, rate=1mbit, queue_disc=codel, queue_size=1000 "),
    ('2', " source='172.16.11.0/24', dest='172.16.10.0/24', delay=10, "
     " loss=0, rate=12mbit, queue_disc=codel, queue_size=1000 "),
]
```

Figure 8: `TPCONF_router_queues` defined in `config.py` – CoDel AQM with same base RTT (20ms) for all flows

```
root@freebsd-router:~ # ipfw pipe show
00001:   1.000 Mbit/s    10 ms burst 0
q131073 1000 sl. 0 flows (1 buckets) sched 65537 weight 0 lmax 0 pri 0
AQM CoDel target 5ms interval 100ms NoECN
 sched 65537 type FIFO flags 0x0 0 buckets 1 active
BKT Prot ___Source IP/port____ ____Dest. IP/port____ Tot_pkt/bytes Pkt/Byte Drp
  0 ip           0.0.0.0/0             0.0.0.0/0       37     1924  0    0   0
00002:  12.000 Mbit/s    10 ms burst 0
q131074 1000 sl. 0 flows (1 buckets) sched 65538 weight 0 lmax 0 pri 0
AQM CoDel target 5ms interval 100ms NoECN
 sched 65538 type FIFO flags 0x0 0 buckets 1 active
  0 ip           0.0.0.0/0             0.0.0.0/0     5183  7770192  3 4500  16
```

```
root@freebsd-router:~ # ipfw show
00100  53226   2832772 pipe 1 ip from 172.16.10.0/24 to 172.16.11.0/24 out
00200 100696 151025260 pipe 2 ip from 172.16.11.0/24 to 172.16.10.0/24 out
65534 314278 315884813 allow ip from any to any
65535      0         0 deny ip from any to any
```

Figure 9: ipfw/dummynet configurations for CoDel

Similar to example 1, in `ipfw pipe show` command output of figure 11, we can see a pipe number 0001 is created with 1Mbit/s rate limit and 10ms emulated delay based on the first tuple of `TPCONF_router_queues` but with droptail queue management and default queue size limit. This because FQ-CoDel is implemented as a scheduler and has its own queue size limit. So, just the link object of the created pipe is used (to be connected to the scheduler) in such configuration.

In `ipfw sched pipe show` command output of figure 11, a scheduler (sched 1) of type FQ_CODEL is created with 1000 packets total queues size limit and other default parameters, and connected to pipe 1.

In `ipfw queue pipe show` command output of figure 11, a queue (q00001) is created with default parameters is created and connected to sched 1. As we mention before, this queue does not buffer packets but instead passes them directly to FQ-CoDel to be hashed and buffered in the sub-queues. Then the queue is added to `ipfw` rules as shown in `ipfw show` command output.

C. Configuring delays per-flow

In order to configure different base RTTs for different flows, we need to setup multiple Dummynet pipes. Based on the fact that `ipfw` passes (if not blocked) packets from a rule to next rule by default, we can emulate different delay based on packet's source and destination IP address.

First, normal TEACUP queues is set up but without delay parameter. Then, additional queues are added for each source/destination IP addresses (or subnets) with FIFO queue discipline and a desired delay. This makes packets with specific IP address to pass to a specific pipe that emulates the desired delay. For example, in the experiments described in this report, we use the `TPCONF_router_queues` in `config.py` as shown in Figure 12.

Here we used CoDel as an illustrative example; the same configuration principles apply to FQ-CoDel, PIE

```
TPCONF_router_queues = [

    ('1', " source='172.16.10.0/24', dest='172.16.11.0/24', delay=10, "
     " loss=0, rate=1mbit, queue_disc=fq_codel, queue_size=1000 "),
    ('2', " source='172.16.11.0/24', dest='172.16.10.0/24', delay=10, "
     " loss=0, rate=21mbit, queue_disc=fq_codel, queue_size=1000 "),
]
```

Figure 10: `TPCONF_router_queues` defined in `config.py` – FQ-CoDel AQM with same base RTT (20ms) for all flows

```
root@freebsd-router:~ # ipfw pipe show
00001:   1.000 Mbit/s    10 ms burst 0
q131073  50 sl. 0 flows (1 buckets) sched 65537 weight 0 lmax 0 pri 0 droptail
 sched 65537 type FIFO flags 0x0 0 buckets 0 active
00002:  12.000 Mbit/s    10 ms burst 0
q131074  50 sl. 0 flows (1 buckets) sched 65538 weight 0 lmax 0 pri 0 droptail
 sched 65538 type FIFO flags 0x0 0 buckets 0 active
```

```
root@freebsd-router:~ # ipfw sched show
00001:   1.000 Mbit/s    10 ms burst 0
q65537  50 sl. 0 flows (1 buckets) sched 1 weight 0 lmax 0 pri 0 droptail
 sched 1 type FQ_CODEL flags 0x0 0 buckets 1 active
 FQ_CODEL target 5ms interval 100ms quantum 1514 limit 1000 flows 1024 ECN
   Children flowsets: 1
BKT Prot ___Source IP/port____ ____Dest. IP/port____ Tot_pkt/bytes Pkt/Byte Drp
  0 ip           0.0.0.0/0            0.0.0.0/0       13      676  0    0   0
00002:  12.000 Mbit/s    10 ms burst 0
q65538  50 sl. 0 flows (1 buckets) sched 2 weight 0 lmax 0 pri 0 droptail
 sched 2 type FQ_CODEL flags 0x0 0 buckets 1 active
 FQ_CODEL target 5ms interval 100ms quantum 1514 limit 1000 flows 1024 ECN
   Children flowsets: 2
  0 ip           0.0.0.0/0            0.0.0.0/0     17765 26643192  5 7500  49
```

```
root@freebsd-router:~ # ipfw queue show
q00001  50 sl. 0 flows (1 buckets) sched 1 weight 0 lmax 0 pri 0 droptail
q00002  50 sl. 0 flows (1 buckets) sched 2 weight 0 lmax 0 pri 0 droptail
```

```
root@freebsd-router:~ # ipfw show
00100  53226   2832772 queue 1 ip from 172.16.10.0/24 to 172.16.11.0/24 out
00200 100696 151025260 queue 2 ip from 172.16.11.0/24 to 172.16.10.0/24 out
65534 314278 315884813 allow ip from any to any
65535      0         0 deny ip from any to any
```

Figure 11: ipfw/dummynet configurations for FQ-CoDel

and FQ-PIE. As shown in Figure 12, we setup eight queues. Queue 1 and 2 setups bidirectional rate limits, queuing disciplines and bottleneck buffer size between '172.16.10.0/24' and '172.16.11.0/24' network (experiment networks). No delays are being set in the first two queues. After being rate-shaped with the desired queuing discipline, constant delay is emulated with subsequent queues for specific hosts (hostname or IP address) in one direction. The first flow - between 'host1A' and 'host1B' has a OWD of 5ms; the second flow - between 'host2A' and 'host2B' has a OWD of 10ms; and the third flow - between 'host3A' and 'host3B' has a OWD of 15ms.

Figure 13 and 14 shows the corresponding ipfw rules ipfw/dummynet configurations. Experiment results with OWD delays configured on a per-flow basis for are presented in Section IV.

```
TPCONF_router_queues = [

    ('1', " source='172.16.10.0/24', dest='172.16.11.0/24', delay=0, "
     " loss=V_loss, rate=V_up_rate, queue_disc=V_aqm, queue_size=V_bsize "),
    ('2', " source='172.16.11.0/24', dest='172.16.10.0/24', delay=0, "
     " loss=V_loss, rate=V_down_rate, queue_disc=V_aqm, queue_size=V_bsize "),

    ('3', " source='host1A', dest='host1B', delay=5, "
     " queue_disc='fifo', queue_size='100' "),
    ('4', " source='host1B', dest='host1A', delay=5, "
     " queue_disc='fifo', queue_size='100' "),

    ('5', " source='host2A', dest='host2B', delay=10, "
     " queue_disc='fifo', queue_size='100' "),
    ('6', " source='host2B', dest='host2A', delay=10, "
     " queue_disc='fifo', queue_size='100' "),

    ('7', " source='host3A', dest='host3B', delay=15, "
     " queue_disc='fifo', queue_size='100' "),
    ('8', " source='host3B', dest='host3A', delay=15, "
     " queue_disc='fifo', queue_size='100' ")
]
```

Figure 12: `TPCONF_router_queues` defined in `config.py` – different base RTTs {10, 20, 30}ms for multiple flows

```
root@freebsd-router:~ # ipfw show
00100  53226   2832772 pipe 1 ip from 172.16.10.0/24 to 172.16.11.0/24 out
00200 100696 151025260 pipe 2 ip from 172.16.11.0/24 to 172.16.10.0/24 out
00300  20213   1077304 pipe 3 ip from 172.16.10.11 to 172.16.11.14 out
00400  38113  57163224 pipe 4 ip from 172.16.11.14 to 172.16.10.11 out
00500  12651    671468 pipe 5 ip from 172.16.10.12 to 172.16.11.17 out
00600  23811  35710144 pipe 6 ip from 172.16.11.17 to 172.16.10.12 out
00700  20363   1084052 pipe 7 ip from 172.16.10.13 to 172.16.11.18 out
00800  38656  57977892 pipe 8 ip from 172.16.11.18 to 172.16.10.13 out
65534 314278 315884813 allow ip from any to any
65535      0         0 deny ip from any to any
```

Figure 13: `ipfw` rules for flows with different base RTTs

```
root@freebsd-router:~ # ipfw pipe show
00001:   1.000 Mbit/s    0 ms burst 0
q131073 1000 sl. 0 flows (1 buckets) sched 65537 weight 0 lmax 0 pri 0
AQM CoDel target 5ms interval 100ms NoECN
 sched 65537 type FIFO flags 0x0 0 buckets 1 active
BKT Prot ___Source IP/port____ ____Dest. IP/port____ Tot_pkt/bytes Pkt/Byte Drp
  0 ip           0.0.0.0/0             0.0.0.0/0       37     1924  0    0   0
00002:  12.000 Mbit/s    0 ms burst 0
q131074 1000 sl. 0 flows (1 buckets) sched 65538 weight 0 lmax 0 pri 0
AQM CoDel target 5ms interval 100ms NoECN
 sched 65538 type FIFO flags 0x0 0 buckets 1 active
  0 ip           0.0.0.0/0             0.0.0.0/0     5183  7770192  3 4500  16
00003: unlimited         5 ms burst 0
q131075 100 sl. 0 flows (1 buckets) sched 65539 weight 0 lmax 0 pri 0 droptail
 sched 65539 type FIFO flags 0x0 0 buckets 1 active
  0 ip           0.0.0.0/0             0.0.0.0/0     2990   159640  0    0   0
00004: unlimited         5 ms burst 0
q131076 100 sl. 0 flows (1 buckets) sched 65540 weight 0 lmax 0 pri 0 droptail
 sched 65540 type FIFO flags 0x0 0 buckets 1 active
  0 ip           0.0.0.0/0             0.0.0.0/0     5163  7740192  0    0   0
00005: unlimited        10 ms burst 0
q131077 100 sl. 0 flows (1 buckets) sched 65541 weight 0 lmax 0 pri 0 droptail
 sched 65541 type FIFO flags 0x0 0 buckets 0 active
00006: unlimited        10 ms burst 0
q131078 100 sl. 0 flows (1 buckets) sched 65542 weight 0 lmax 0 pri 0 droptail
 sched 65542 type FIFO flags 0x0 0 buckets 0 active
00007: unlimited        15 ms burst 0
q131079 100 sl. 0 flows (1 buckets) sched 65543 weight 0 lmax 0 pri 0 droptail
 sched 65543 type FIFO flags 0x0 0 buckets 0 active
00008: unlimited        15 ms burst 0
q131080 100 sl. 0 flows (1 buckets) sched 65544 weight 0 lmax 0 pri 0 droptail
 sched 65544 type FIFO flags 0x0 0 buckets 0 active
```

Figure 14: ipfw/dummynet configurations for CoDel – different base RTTs

VI. Conclusions

This technical report describes the extension to TEACUP v1.0 in order to support automated configuration of bottleneck router using Dummynet AQM – FreeBSD implementation of CoDel, FQ-CoDel, PIE and FQ-PIE in the ipfw/dummynet framework. We explain how Dummynet configurations are instantiated by TEACUP. We also provide patching instructions, illustrative ipfw/dummynet configurations and present preliminary experimental results to illustrate the plausible operation of our testbed when base delays are configured on a per-flow basis in multi-flow experiments.

VII. Acknowledgements

Extensions to TEACUP described in this report were made possible in part by a gift from the Comcast Innovation Fund.

References

[1] J. Gettys and K. Nichols, "Bufferbloat: Dark Buffers in the Internet," *Queue*, vol. 9, pp. 40:40–40:54, Nov 2011.

[2] R. Al-Saadi and G. Armitage, "Implementing AQM in FreeBSD." http://caia.swin.edu.au/freebsd/aqm/, 2015.

[3] S. Zander and G. Armitage, "TEACUP v1.0 - A System for Automated TCP Testbed Experiments," Tech. Rep. 150529A, Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, 29 May 2015.

[4] R. Al-Saadi and G. Armitage, "Dummynet AQM v0.2 – CoDel, FQ-CoDel, PIE and FQ-PIE for FreeBSD's ipfw/dummynet framework," Tech. Rep. 160418A, Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, 18 April 2016.

[5] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar, "Controlled Delay Active Queue Management." IETF Draft, Mar 2016.

[6] R. Pan, P. Natarajan, F. Baker, G. White, B. VerSteeg, M. Prabhu, C. Piglione, and V. Subramanian, "PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem." IETF Draft, draft-ietf-aqm-pie-06, April 2016.

[7] T. Høeiland-Jørgensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet, "FlowQueue-Codel." IETF Draft, draft-ietf-aqm-fq-codel-06, March 2016.

[8] L. Stewart, "SIFTR - Statistical Information for TCP Research." http://caia.swin.edu.au/urp/newtcp/tools.html, 2007.

[9] "The Web10G Project." http://web10g.org.

[10] 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)*, pp. 264 – 267, IEEE, Oct 2013.

[11] S. Zander and G. Armitage, "TEACUP v1.0 - Data Analysis Functions," Tech. Rep. 150529B, Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, 29 May 2015.

[12] R. Al-Saadi and G. Armitage, "Dummynet AQM v0.1 - CoDel and FQ-CoDel for FreeBSD's ipfw/dummynet framework," Tech. Rep. 160226A, Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, 26 February 2016.