Dummynet AQM v0.1 – CoDel and FQ-CoDel for FreeBSD's ipfw/dummynet framework

Rasool Al-Saadi, Grenville Armitage
Centre for Advanced Internet Architectures, Technical Report 160226A
Swinburne University of Technology
Melbourne, Australia
ralsaadi@swin.edu.au, garmitage@swin.edu.au

***Abstract*—Controlled delay (CoDel) is an active queue management (AQM) scheme designed to control bottleneck queueing delay. FlowQueue-CoDel (FQ-CoDel) is hybrid scheme that hashes flows into one of N queues, applies CoDel AQM on a per-queue basis and utilises modified Deficit Round Robin (DRR) scheduling to share link capacity between queues. Although implementations of CoDel and FQ-CoDel have existed in the Linux kernel since version 3.5, no well-documented implementation has existed for FreeBSD to date. Here we document an independent implementation of CoDel and FQ-CoDel in FreeBSD's ipfw/dummynet framework – adding useful functionality to FreeBSD and supporting the IETF's AQM working group by confirming the current CoDel and FQ-Codel Internet Drafts. We confirm the functionality of our implementation by comparing with Linux CoDel/FQ-CoDel. Patches have been prepared for FreeBSD11-CURRENT-r295345 and FreeBSD-10.x-RELEASE.**

***Index Terms*—AQM, Scheduler, CoDel, FQ-CoDel, FreeBSD, Dummynet, IPFW**

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 oversized buffer due to low memory prices and this causes high latency in congested bottlenecks if traditional tail drop is used. Researchers have been attracted to find solutions to this problem by using active queue management (AQM) to manage bottlenecks buffers. AQM controls queue length by dropping/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) [1] and its variations, use queue occupancy as an indicator of how much a queue is congested. However, these types of AQMs are hard to configure and perform badly in certain scenarios [2].

CoDel (Controlled delay) [3] is an example of a modern AQM designed to remedy these limitations by using queue delay, instead of queue length, to indicate standing queues. FQ-CoDel [4] 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 [5]. However, FreeBSD has only had an undocumented, Linux-code based CoDel implementing in pf/ALTQ framework. Using current IETF Internet Drafts [3], [4] we have independently implemented CoDel and FQ-Codel for FreeBSD's ipfw/dummynet framework [6]. Our work helps usefully enhance FreeBSD and demonstrate that [3] and [4] are clear enough for implementation.

Our v0.1 patchset (CoDel and FQ-CoDel) can be applied to FreeBSD11-CURRENT r295345 and FreeBSD-10.{0,1,2}-RELEASE. We confirm the functionality of our v0.1 implementation by comparing our results with Linux CoDel and FQ-CoDel implementation.

The rest of this technical report is organised as follow: Section II includes basic information about ipfw/dummynet. Sections III and IV contain detailed information about configuring CoDel and FQ-CoDel within ipfw. Section V gives instructions of how to apply, test and install our patch. Section VI compares experimentally derived results from the FreeBSD and Linux CoDel and FQ-CoDel implementations, and Section VII includes our conclusion and future work.

II. ipfw/Dummynet

FreeBSD's IPFW firewall supports both IPv4 and IPv6, and allows filtering, redirecting, NAT, forwarding, and other operations on IP packets passing through network interfaces [7]. IPFW is tightly integrated with

dummynet, providing traffic shaping, delay emulation, packet scheduling and queue management functionality.

Originally a tool to run experiments in an emulated network environment, Dummynet has been improved over time to support emulation of complex network configurations [8]. The latest version of Dummynet implements three queue management schemes (Drop Tail, RED and GRED) and supports dynamically loadable packet schedulers with many schedulers implementation including First In First Out (FIFO), Worst-case Weighted Fair Queueing (WF2Q+), Quick Fair Queueing (QFQ), and others.

Dummynet provides users with three objects:

1) **Pipe**: represents a link that supports traffic shaping and delay emulation.
2) **Queue**: represents a queue of packets managed by a queuing management scheme and connected to a packet scheduler.
3) **Scheduler**: represents a packet scheduler connected to a link and has one or more queues.

The user-space `ipfw` command is used to create, delete, configure and show these objects. For simplicity and compatibility reasons, dummynet creates additional objects when a certain objects is created. For example, when new pipe is created, queue and scheduler objects are created as well. For more details see the ipfw(8) FreeBSD Man Pages [9].

We implemented CoDel and FQ-CoDel in Dummynet due to its popularity and flexibility. To make the process of implementing new AQMs for Dummynet easier, we also added support for dynamically loadable AQM kernel module similar Dummynet schedulers.

An AQM module must define two mandatory functions (dequeue and enqueue) and could have a structure for state variables (for each queue) to store AQM internal variables and another for configuration parameters (for flowset) to store AQM configurations. In our implementation, CoDel is implemented as an AQM module to be configured for queues (or for implicit queues created when pipes are created) while FQ-CoDel is implemented as scheduler module that includes both FQ scheduler code and CoDel AQM code.

III. CoDel

CoDel drops or marks packets depending on packet sojourn time in the queue, and aims to achieve high throughput while controlling queue delay and to be nearly insensitive to flow RTT. It was designed to be parameterless and work properly on the Internet without changing its default configurations. However, the defaults in [3] are not always suitable (for example, when path RTT is high) and can degrade TCP throughput. Thus, our implementation provides options to change CoDel parameters for each pipe/queue individually as well as changing the defaults.

A. CoDel Parameters

CoDel has two primary parameters (`target` and `interval`) and one option (to enable Explicit Congestion Notification, ECN). `target` is the minimum acceptable persistent queue delay that CoDel allows. CoDel does not drop packets directly after packets sojourn time becomes higher than `target` but waits for `interval` before dropping. `interval` should be set to maximum RTT for all expected connection. `ecn` controls whether, for ECN-enabled TCP flows, CoDel marks or drops packets when queue delay become high.

Table I shows our CoDel configuration parameters, default values and `sysctl` control variables to change the default value.

Table I: CoDel configuration parameters

Parameter	Default	sysctl variable
`target`	5 ms	`net.inet.ip.dummynet.codel.target`
`interval`	100 ms	`net.inet.ip.dummynet.codel.interval`
`ecn`	no ECN	-

B. CoDel Synopsis

CoDel is used with dummynet '`pipe`' or '`queue`' and can be configured through `ipfw` [9] interface. CoDel has the following synopsis:

```
ipfw pipe/queue x config [...] codel [
    target t] [interval t] [ecn]
```

where

`t` is time in millisecond.

Note: any token after '`codel`' is considered a CoDel parameter, so ensure all pipe/queue configuration options are written before '`codel`'.

C. Examples of using CoDel

This subsection includes some examples of using CoDel with ipfw/dummynet. It should be noted that `ipfw` passes packets that match a classification rule of dummynet pipe/queue to next rule by default. Thus, a rule with '`allow`' action should be added in some point after pipe/queue rule.

1) One pipe controlled by CoDel AQM (default configuration) and rate limit to 1 Mbits/s.

```
ipfw pipe 1 config bw 1mbits/s codel
ipfw add 100 pipe 1 form any to any
```

2) Two queues controlled by CoDel AQM using different CoDel configurations parameters. The pipe that queue 1 and 2 use has rate limit to 10 Mbits/s and 20ms emulated delay. In more details, queue 1 and 2 connected to an implicit WF2Q+ scheduler that use pipe 1 for traffic shaping and adding emulated delay.

```
ipfw pipe 1 config bw 10mbits/s delay
    20ms
ipfw queue 1 config pipe 1 codel
   target 7 ecn
ipfw queue 2 config pipe 1 codel
   target 8 interval 160 ecn
ipfw add 100 queue 1 form
   192.168.0.0/16 to 192.168.0.0/16
ipfw add 200 queue 2 from
   172.16.0.0/16 to 172.16.0.0/16
```

3) Two queues - queue 1 controlled by CoDel AQM and queue 2 uses droptail. Both queues are connected to QFQ scheduler that uses pipe 1 for rate limit to 5 Mbits/s.

```
ipfw pipe 1 config bw 5mbits/s
ipfw sched 1 config pipe 1 type qfq
ipfw queue 1 config sched 1 codel
ipfw queue 2 config sched 1
ipfw add 100 queue 1 form
   192.168.0.0/16 to 192.168.0.0/16
ipfw add 200 queue 2 from
   172.16.0.0/16 to 172.16.0.0/16
```

IV. FQ-CoDel

As noted earlier, FQ-CoDel aims to control queuing delays while sharing bottleneck capacity relatively evenly among competing flows. FQ-CoDel's modified DRR (Deficit Round Robin) scheduler manages two lists of queues – old queues and new queues – to provide brief periods of priority to lightweight or short burst flows. FQ-CoDel's internal, dynamically created queues are controlled by separate instances of CoDel AQM (including separate state variables per queue).

The default parameters for FQ-CoDel in [4] are chosen to be generally useful. However, they are not always suitable so our implementation provides options to change FQ-CoDel parameters for each scheduler individually as well as changing the defaults.

A. FQ-CoDel Parameters

FQ-CoDel has five primary parameters (`target`, `interval`, `quantum`, `limit` and `flows`) and one option (to enable Explicit Congestion Notification, ECN). `target`,`interval` and `ecn` are CoDel parameters described in section III-A. `quantum` is number of bytes a queue can be served before being moved to the tail of old queues list. `limit` is the hard size limit of all queues managed by an instance of the fq_codel scheduler. `flows` is number of flow queues that fq_codel creates and manages.

Table II shows our FQ-CoDel configuration parameters, default values and `sysctl` control variables to change the default value.

Table II: FQ-CoDel configuration parameters

Parameter	Default	sysctl variable
`target`	5 ms	`net.inet.ip.dummynet.fqcodel.target`
`interval`	100 ms	`net.inet.ip.dummynet.fqcodel.interval`
`ecn`	no ECN	-
`quantum`	1514	`net.inet.ip.dummynet.fqcodel.quantum`
`limit`	10240	`net.inet.ip.dummynet.fqcodel.limit`
`flows`	1024	`net.inet.ip.dummynet.fqcodel.flows`

B. FQ-CoDel synopsis

fq_codel is used with dummynet scheduler object ('schd') and can be configured through `ipfw` interface. fq_codel has the following synopsis:

```
ipfw sched x config [...] type fq_codel
     [target t] [interval t] [ecn] [
    quantum n] [limit n] [flows n]
```

where

`t` is time in millisecond.

`n` is integer number

Note: any token after '`fq_codel`' is considered an FQ-CoDel parameter, so ensure all other scheduler configuration options come before '`fq_codel`'.

C. Examples of using FQ-CoDel

This subsection includes some examples of using fq_codel with ipfw/dummynet. Note that `ipfw` passes packets that match a classification rule of dummynet pipe/queue to next rule by default. Thus, a rule with '`allow`' action should be added in some point after pipe/queue rule.

1) One scheduler with one queue, 2048 fq_codel sub-queues, target 7ms and quantum 2000 bytes

```
ipfw pipe 1 config bw 10mbits/s
ipfw sched 1 config pipe 1 type
   fq_codel target 7 quantum 2000
   flows 2048
ipfw queue 1 config sched 1
ipfw add 100 queue 1 form
   192.168.0.0/16 to 192.168.0.0/16
```

2) One scheduler with two queues (1024 fq_codel sub-queues by default), interval 150ms, ECN enabled

```
ipfw pipe 1 config delay 10ms
ipfw sched 1 config pipe 1 type
   fq_codel interval 150 ecn
ipfw queue 1 config sched 1
ipfw queue 2 config sched 1
ipfw add 100 queue 1 form
   192.168.0.0/16 to 192.168.0.0/16
ipfw add 200 queue 2 from
   172.16.0.0/16 to 172.16.0.0/16
```

V. APPLYING THE PATCH/TESTING/INSTALLATION

We have tested our FreeBSD11-based v0.1 patch [6] against FreeBSD11-CURRENT r295345, although it may also work with earlier or later builds. To build r295345 from source you'll need to checkout FreeBSD11-CURRENT r295345 source tree using:

```
svn checkout -r 295345 svn://svn.freebsd.
   org/base/head/ /usr/src/
```

We also provide a version for FreeBSD 10.x-RELEASE (10.0, 10.1, 10.2) as a separate patch file that combines our CoDel/FQ-CoDel code with ECN marking code backported from FreeBSD11-CURRENT r266941.

Once the patch is applied, you only need to (re)build the `dummynet.ko` kernel module and `ipfw` userland command (rather than rebuild a complete kernel and world from source). As `root` user do the following steps:

A. Applying the patch

Apply the patch as follows:

1) Extract the patch file

```
tar -xvf dummynet-aqm-patch-0.1.tgz -
   C /usr/src/ cd /usr/src
```

2) Apply the patch
   a) For FreeBSD11-CURRENT

```
patch -p1 < dummynet-aqm-patch
   -0.1/freebsd11-r295345.patch
```

   b) For FreeBSD 10.x-RELEASE

```
patch -p1 < dummynet-aqm-patch
   -0.1/freebsd10.x.patch
```

3) Copy `ip_dummynet.h` to /usr/include/netinet.

```
cp /usr/src/sys/netinet/ip_dummynet.h
    /usr/include/netinet/
```

4) Build ipfw userland.

```
cd /usr/src/sbin/ipfw
make
```

5) Build dummynet kernel module

```
cd /usr/src/sys/modules/dummynet/
make
```

B. Testing the patched ipfw/dummynet

Use the following steps to check whether the patch applied cleanly and built both userland ipfw and dummynet kernel module:

1) Check if old dummynet is already loaded

```
loaded kldstat | grep dummynet
```

2) If dummynet is already loaded, unload it.

```
kldunload dummynet
```

3) Load the patched dummynet.ko into FreeBSD kernel

```
kldload /usr/src/sys/modules/dummynet
   /dummynet.ko
```

4) Check debug messages using '`dmesg`' command:

```
dmesg | grep CODEL
```

The Output should be something like:

```
load_dn_aqm dn_aqm CODEL loaded
load_dn_sched dn_sched FQ_CODEL
   loaded
```

5) Use the patched ipfw interface (specify a full pathname when use ipfw):

```
/usr/src/sbin/ipfw/ipfw pipe 1 config
    codel
/usr/src/sbin/ipfw/ipfw pipe 1 show
```

The Output should be something like:

```
00001: unlimited 0 ms burst 0
q131073 50 sl. 0 flows (1 buckets)
   sched 65537 weight 0 lmax 0 pri 0
   AQM CoDel target 5 interval 100
sched 65537 type FIFO flags 0x0 0
   buckets 0 active
```

C. Installing the patched ipfw and dummynet.ko

To use the patched ipfw/dummynet by default, install them as follow:

1) Install ipfw interface, ignore any error if appears.

```
cd /usr/src/sbin/ipfw
make install
```

2) Install dummynet kernel module

```
cp /usr/src/sys/modules/dummynet/
    dummynet.ko /boot/kernel/
```

3) (Optional) To avoid the warning "KLD '/boot/kernel/dummynet.ko' is newer than the linker.hints file", regenerate kernel loader hints with:

```
kldxref /boot/kernel
```

VI. Experimental comparisons of CoDel/FQ-CoDel in Linux and FreeBSD

We used a TEACUP-based [10] testbed[1] to compare the behaviour of our implementation with a Linux implementation. Figure 1 shows our testbed's network topology, with three hosts[2], one bottleneck router[3] and a control host. Experiment network links are 1Gbps Ethernet, while the Control network used separate 100Mbps Ethernet connections to each machine.

Each end host could be booted into FreeBSD10.1-RELEASE or Linux 4.2 while the control host ran FreeBSD 10.1. The bottleneck router could be booted into FreeBSD10.1-RELEASE, FreeBSD11-CURRENT-r295345 or Linux 3.17.4 as required. (Note that because FreeBSD11-CURRENT is the development branch with lots of debugging code enabled, the FreeBSD CoDel and FQ-CoDel tests in this section were run using FreeBSD 10.1-RELEASE.) When running CoDel and FQ-CoDel under Linux, TEACUP uses the `netem` and `tc` modules to emulate our target RTT and bottleneck bandwidths.[4]

A. Linux CoDel vs FreeBSD CoDel

In all our CoDel comparison experiments we used `iperf` under FreeBSD or Linux to generate one TCP flow running for 60 seconds between Hosts 2 and 3 (Figure 1). The router applied independent instances of CoDel to 10Mbit/sec bottlenecks in each direction, with each CoDel buffer set to 1000 packets. The router also emulated either 20ms or 40ms of underlying path RTT.

[1]TEACUP source is at https://sourceforge.net/projects/teacup/

[2]Intel Core 2 Duo @ 3GHz, 4GiB RAM, 1Gbps NICs

[3]Intel Core 2 Duo @ 2.33GHz, 1Gbps NICs

[4]See section IV-B of [10] for details on how TEACUP appropriately configures `netem` and `tc` for this purpose.

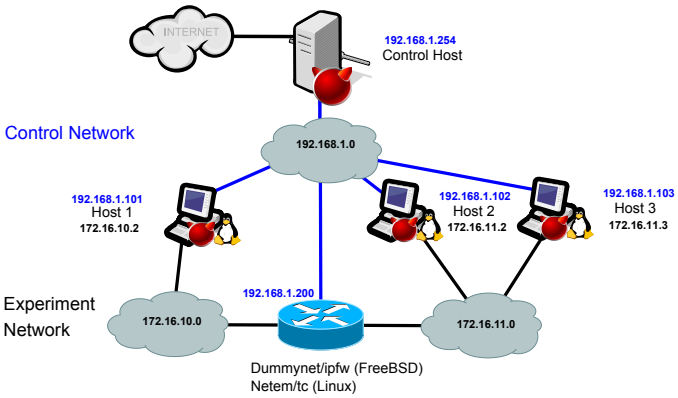


Figure 1: TEACUP Testbed topology

Scenario 1: NewReno over CoDel @ 20ms RTT

In scenario 1 the end hosts booted into FreeBSD and ran TCP NewReno over a 20ms RTT path. CoDel was configured with target 5 ms, interval 100 ms and no ECN. Figure 2 shows throughput[5], CWND and smoothed TCP RTT versus time for both FreeBSD and Linux implementations of CoDel. Our implementation behaves very similarly to CoDel under Linux.

Scenario 2: NewReno over CoDel @ 40ms RTT

Scenario 2 is the same as scenario 1 except that the emulated path RTT is 40 ms. Figure 3 shows throughput, CWND and smoothed TCP RTT versus time, and again the behaviour of our implementation is very similar to CoDel under Linux.

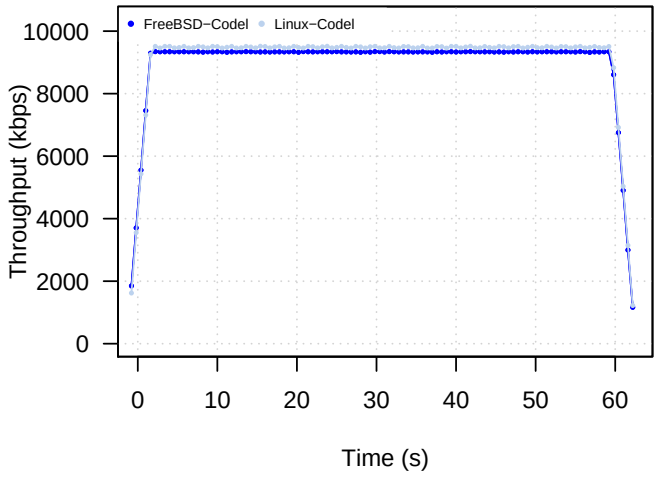
Scenario 3: CUBIC over CoDel @ 20ms RTT

In scenario 3 the end hosts booted into Linux and ran TCP CUBIC over a 20ms RTT path. CoDel was configured with target 10ms, interval 100ms and ECN enabled. We checked number of dropped packet and confirmed that ECN is functional with nothing dropped during the experiment. Figure 4 shows throughput, CWND, smoothed TCP RTT versus time, and again the behaviour of our implementation is very similar to CoDel under Linux.

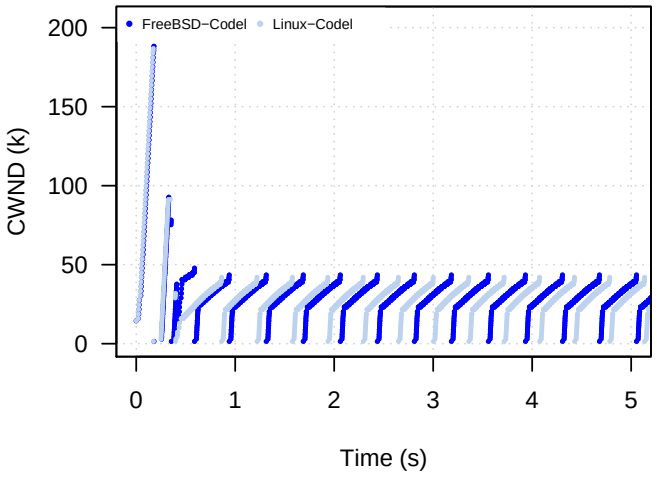
B. Linux FQ-CoDel vs FreeBSD FQ-CoDel

Similar to the CoDel experiments, we compared our FQ-CoDel implementation results with Linux FQ-CoDel to illustrate their similarities. In this case we used multiple instances of `iperf` under Linux to generate four TCP CUBIC flows with staggered start and end

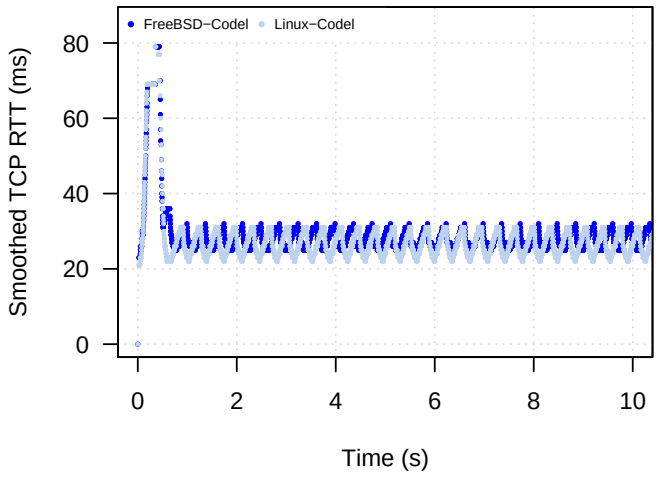
[5]All CoDel experiments throughputs were calculated using 3 seconds window moving forward in steps of 0.6 sec



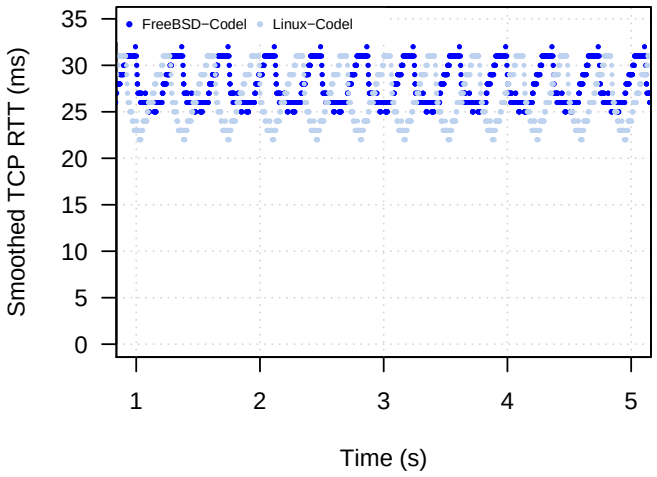
(a) Throughput vs. Time



(b) CWND vs. Time from t=0 to t=5



(c) smoothed TCP RTT vs. Time for first 10 seconds



(d) smoothed TCP RTT vs. Time from t=1 to t=5

Figure 2: One FreeBSD NewReno flow, 20ms RTT path, 10Mbps rate limit and 1000-packet bottleneck buffer. Linux and FreeBSD CoDel configured for target 5ms, interval 100ms, ECN disabled (Scenario 1)

times (starting at t=0, 10, 20, 30 seconds and each flow lasting for 60 seconds). Each instance of fq-codel was configured for target 5ms, interval 100ms, no ECN, quantum 1514 bytes and with 10240 packets of bottleneck buffering shared by 1024 fq_codel sub-queues.

Scenario 4: 10Mbit/sec bottleneck

In scenario 4 the bottleneck was configured for 10Mbit/sec. Figure 5 shows throughput[6], CWND and smoothed TCP RTT versus time for the experiment. As with CoDel, our FQ-CoDel implementation behaves similarly to the Linux implementation.

Scenario 5: 1Mbit/sec bottleneck

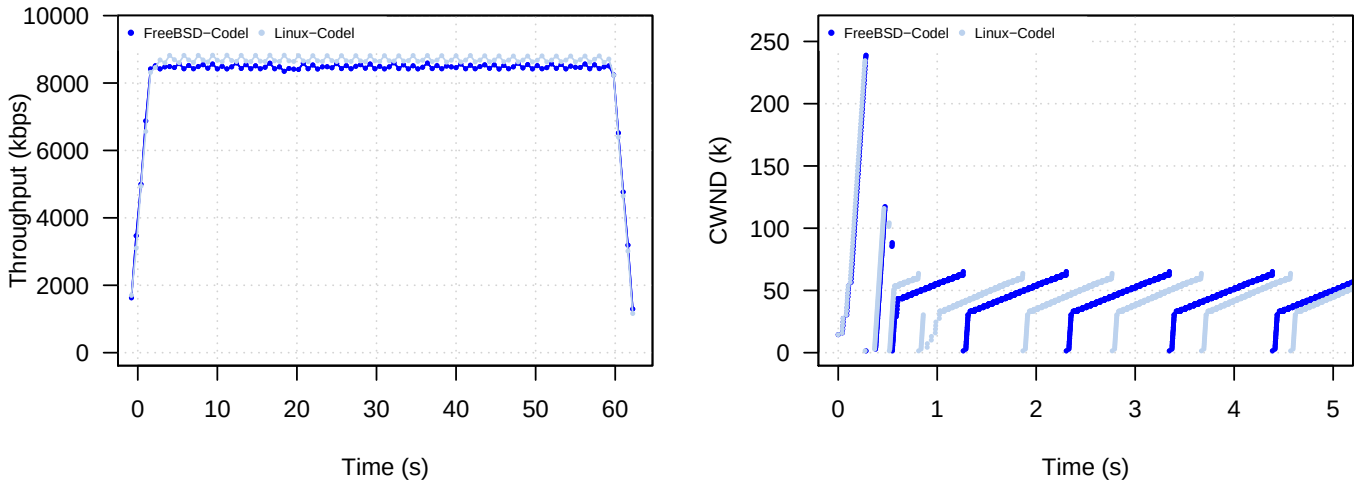
Scenario 5 repeats scenario 4 but with a 1Mbit/sec bottleneck. Figure 6 shows throughput, CWND and smoothed TCP RTT versus time for the experiment. For reasons we have yet to determine, our FQ-CoDel implementation seems to actually share the limited 1Mbps among multiple flows more consistently than the Linux implementation.[7]

VII. Conclusions and Future Work

This report focuses on Dummynet AQM v0.1 – our FreeBSD implementation of CoDel and FQ-CoDel in

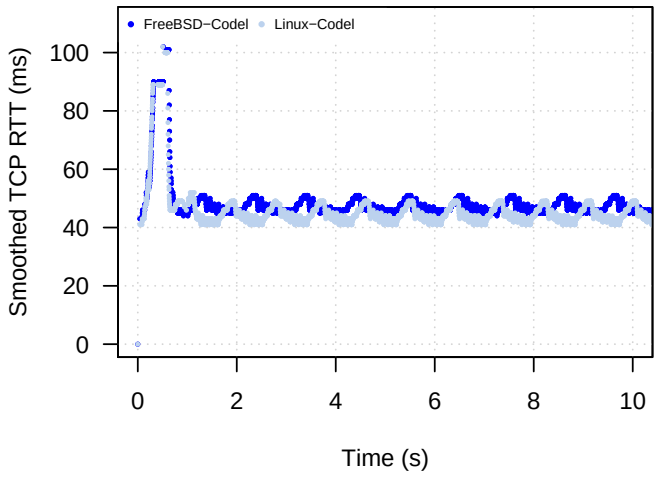
[6] All FQ-CoDel experiments throughputs were calculated using a 1.5 sec window moving forward in steps of 0.3 sec

[7] Whether our implementation's behaviour is more *correct* is a separate question

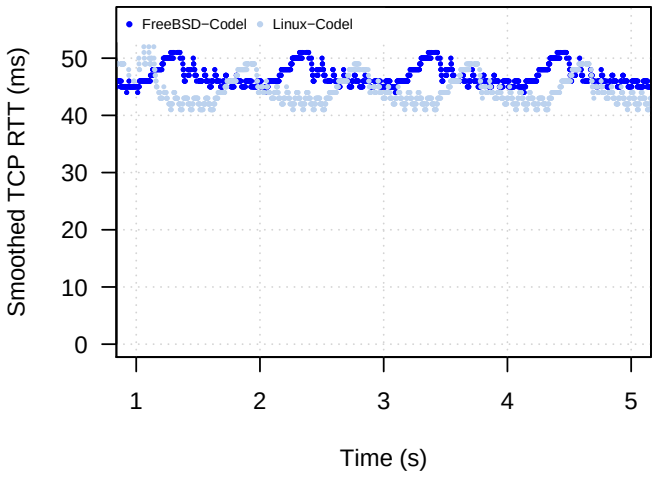


(a) Throughput vs. Time

(b) CWND vs. Time from t=0 to t=5



(c) smoothed TCP RTT vs. Time for first 10 seconds



(d) smoothed TCP RTT vs. Time from t=1 to t=5

Figure 3: One FreeBSD NewReno flow, 40ms RTT path, 10Mbps rate limit and 1000-packet bottleneck buffer. Linux and FreeBSD CoDel configured for target 5ms, interval 100ms, ECN disabled (Scenario 2)

ipfw/dummynet framework. We summarise the AQM and Dummynet context, provide instructions for applying the patch, and present preliminary experimental results suggesting that our independent implementation of [3] and [4] behaves very similarly to the Linux kernel 3.17.4 implementations.

Our experimental testing has been limited, aiming primarily to confirm plausible similarity in behaviours and (indirectly) confirm the clarity of the current Internet Drafts. Future work will include more detailed study of the small (yet unexpected) divergence in behaviour between Linux FQ-CoDel and our FreeBSD implementation when faced with low (1Mbps) bottleneck rates. In the near future we also hope to release an independent implementation of PIE AQM (based on the available Internet Draft [11]) and FQ-PIE (based on plausible design choices).

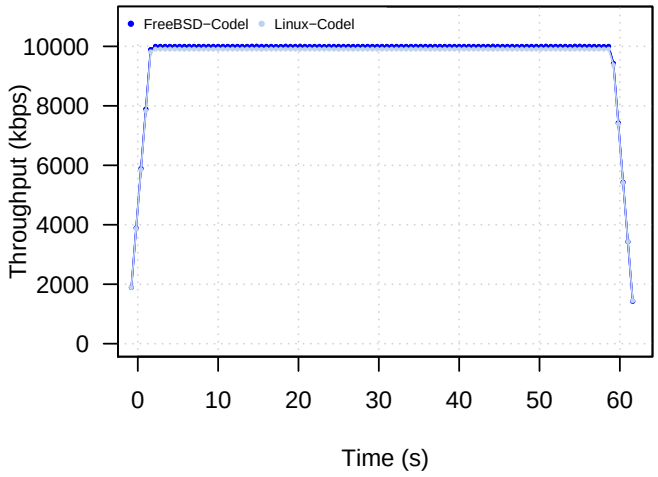
VIII. Acknowledgements

This project has been made possible in part by a gift from the Comcast Innovation Fund.

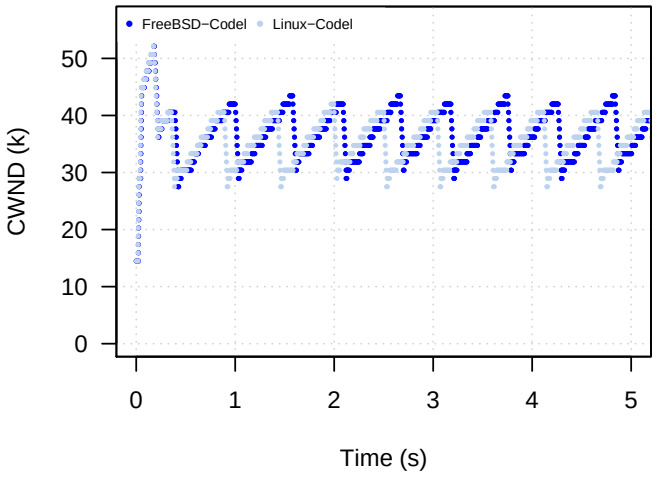
References

[1] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *Networking, IEEE/ACM Transactions on*, vol. 1, no. 4, pp. 397–413, Aug 1993.

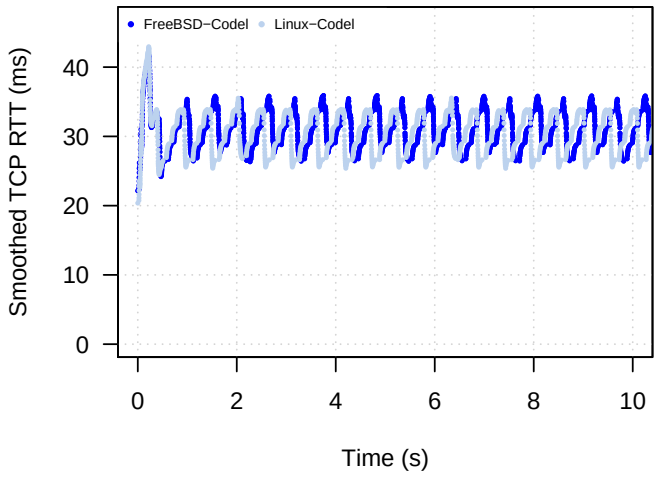
[2] K. Nichols and V. Jacobson, "A modern aqm is just one piece of the solution to bufferbloat," *ACM Queue Networks*, vol. 10, no. 5, 2012. [Online]. Available: http://queue.acm.org/detail.cfm?id=2209336



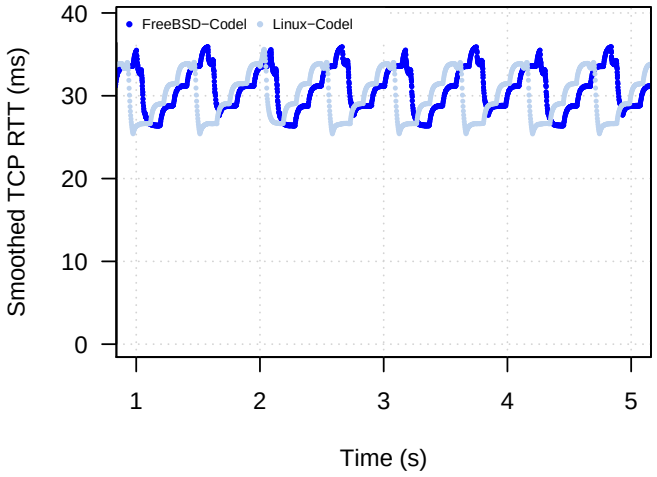
(a) Throughput vs. Time



(b) CWND vs. Time from t=0 to t=5



(c) smoothed TCP RTT vs. Time for first 10 seconds



(d) smoothed TCP RTT vs. Time from t=1 to t=5

Figure 4: One Linux CUBIC flow, 20ms RTT path, 10Mbps rate limit and 1000-packet bottleneck buffer. Linux and FreeBSD CoDel configured for target 10ms, interval 100ms, ECN enabled (Scenario 3)

[3] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar, "Controlled Delay Active Queue Management," IETF Draft, December 2015. [Online]. Available: https://tools.ietf.org/html/draft-ietf-aqm-codel-02

[4] T. Hoeiland-Joergensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet, "FlowQueue-Codel," IETF Draft, February 2016. [Online]. Available: https://tools.ietf.org/html/draft-ietf-aqm-fq-codel-04

[5] *Bufferbloat Wiki*. [Online]. Available: www.bufferbloat.net/projects/codel/wiki

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

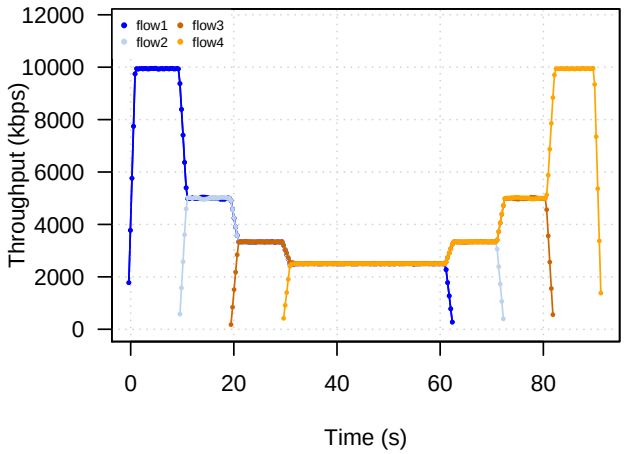
[7] *IPFW - FreeBSD Handbook*. The FreeBSD Documentation Project, 2015. [Online]. Available: https://www.freebsd.org/doc/handbook/firewalls-ipfw.html

[8] M. Carbone and L. Rizzo, "Dummynet revisited," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 2, pp. 12–20, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1764873.1764876

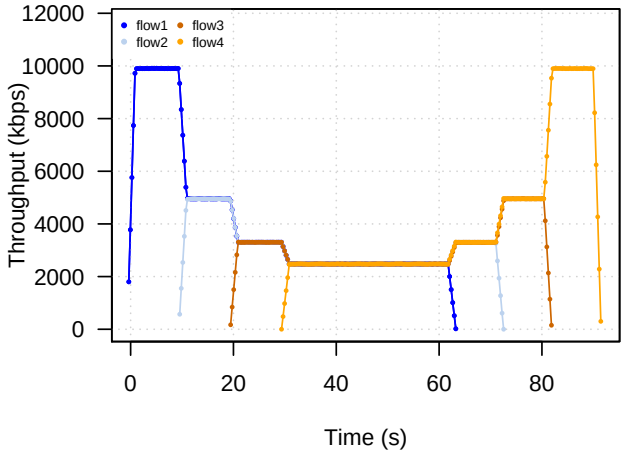
[9] *IPFW(8)*, FreeBSD System Manager's Manual. [Online]. Available: https://www.freebsd.org/cgi/man.cgi?ipfw(8)

[10] S. Zander and G. Armitage, "TEACUP v1.0 - A System for Automated TCP Testbed Experiments," Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, Tech. Rep. 150529A, 2015. [Online]. Available: http://caia.swin.edu.au/reports/150529A/CAIA-TR-150529A.pdf

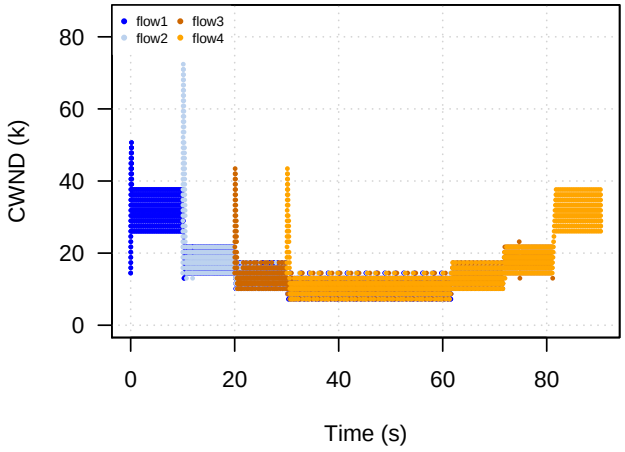
[11] R. Pan, P. Natarajan, F. Baker, G. White, B. VerSteeg, M. Prabhu, C. Piglione, and V. Subramanian, "PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem," IETF Draft, https://tools.ietf.org/html/draft-ietf-aqm-pie-03, November 2015. [Online]. Available: https://tools.ietf.org/html/draft-ietf-aqm-pie-03



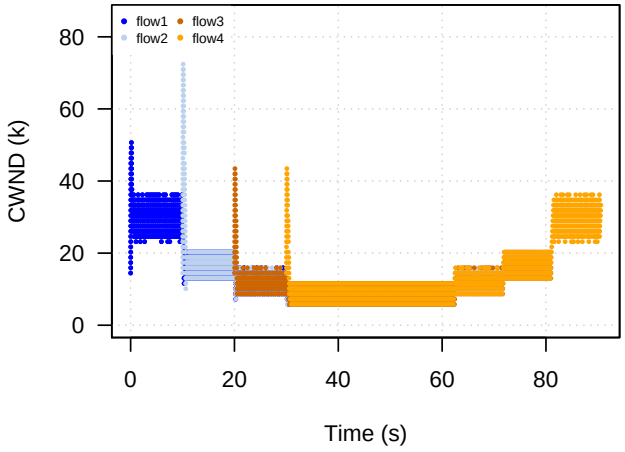
(a) FreeBSD fq_codel Throughput vs. Time



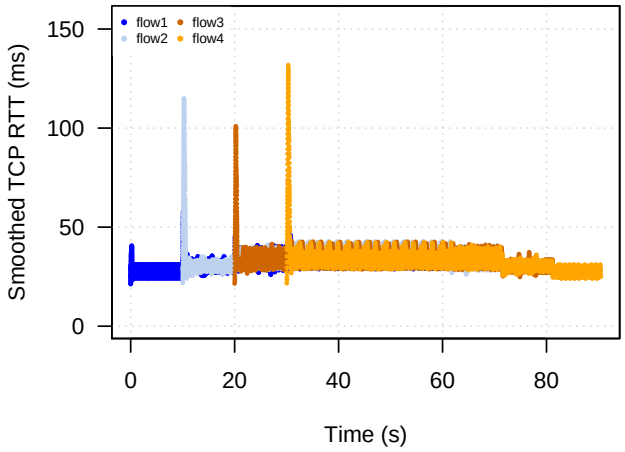
(b) Linux fq_codel Throughput vs. Time



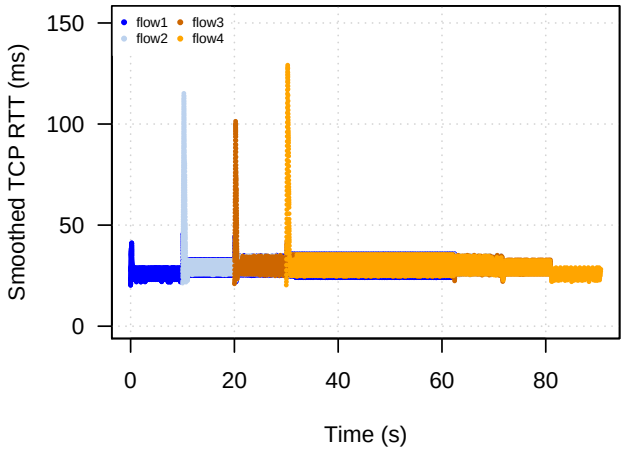
(c) FreeBSD fq_codel CWND vs. Time



(d) Linux fq_codel CWND vs. Time

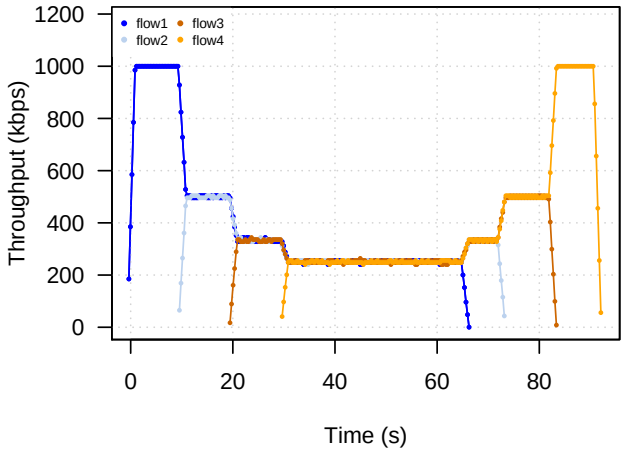


(e) FreeBSD fq_codel smoothed TCP RTT vs. Time

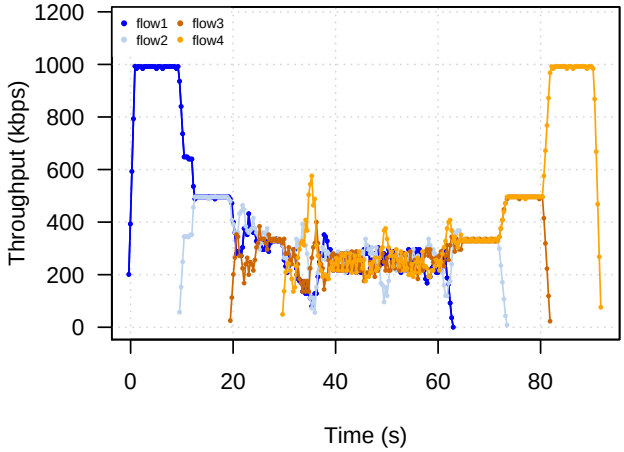


(f) Linux fq_codel smoothed TCP RTT vs. Time

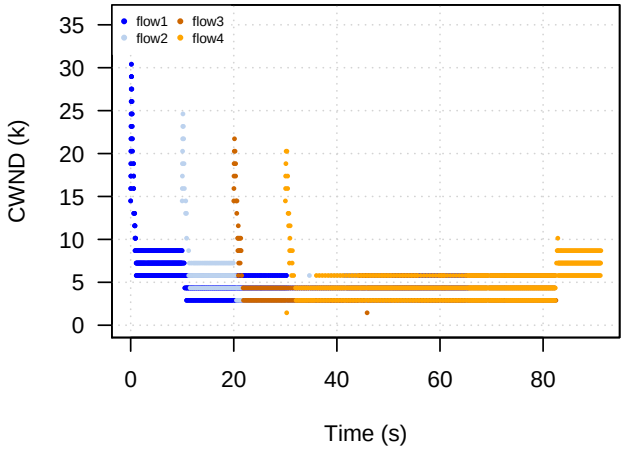
Figure 5: Four Linux CUBIC flows, 20ms RTT path, 10Mbps limit. Both FQ-CoDels configured for target 5ms, interval 100ms, 1024 queues, quantum 1514 bytes, 10240-packet buffering, ECN disabled (Scenario 4)



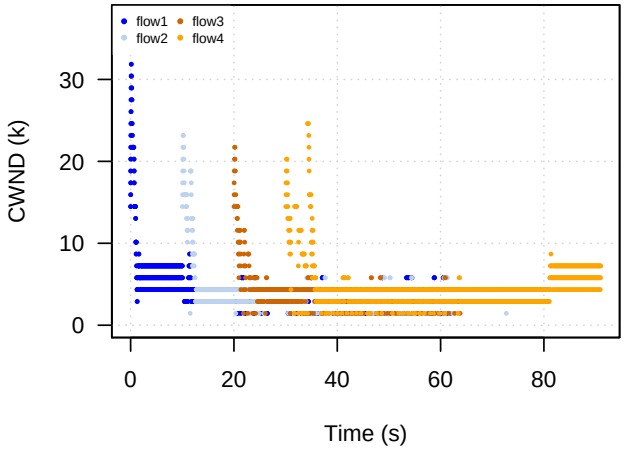
(a) FreeBSD fq_codel Throughput vs. Time



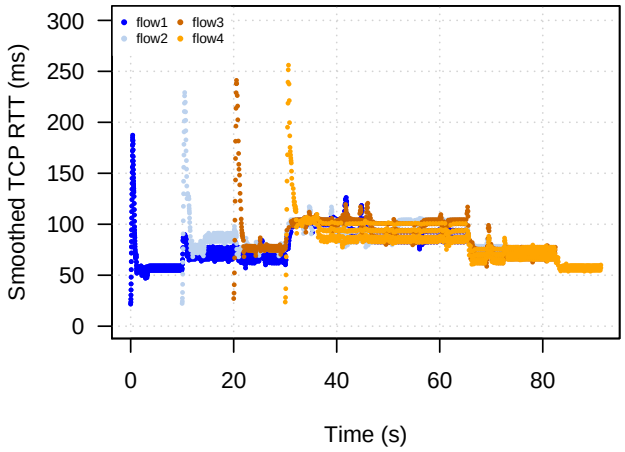
(b) Linux fq_codel Throughput vs. Time



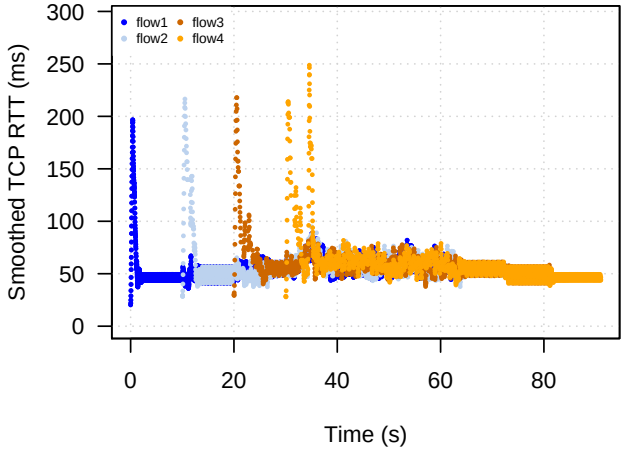
(c) FreeBSD fq_codel CWND vs. Time



(d) Linux fq_codel CWND vs. Time



(e) FreeBSD fq_codel smoothed TCP RTT vs. Time



(f) Linux fq_codel smoothed TCP RTT vs. Time

Figure 6: Four Linux CUBIC flows, 20ms RTT path, 1Mbps limit. Both FQ-CoDels configured for target 5ms, interval 100ms, 1024 queues, quantum 1514 bytes, 10240-packet buffering, ECN disabled (Scenario 5)