How to reliably measure the performance of modern AQMs – and what comes of doing so?

Toke Høiland-Jørgensen

Karlstad University

toke.hoiland-jorgensen@kau.se

ICCRG, November 11, 2014

Outline

- Measuring network behaviour
- Measurement results
- The netperf-wrapper testing tool
- References and questions
- Appendix slides

Why do measurements?

Most algorithm evaluations are based on simulations, however:

- The algorithm implementations can differ
  - As can end-host network stack implementations
- Simulations are idealised
  - No interactions with network hardware and drivers
- Bugs. Different ones.

Difficulties when running experiments

- Ensuring the right configuration is applied
- Keeping track of configuration and test parameters afterwards
- Storing measurement data
- Coordinating different test tools
- Reproducing experiments

The netperf-wrapper tool

A Python wrapper for running tests; main features:

- Run several tools in concert
  - and parse their output to a common format (JSON)
- Store metadata along with the test results
- Automatic gathering of metadata
- Batch facilities
- Plotting; lots of plotting.

Outline

- Measuring network behaviour
- **Measurement results**
- The netperf-wrapper testing tool
- References and questions
- Appendix slides

The measurements

- Steady-state behaviour
  - The Realtime Response Under Load (RRUL) test
  - VoIP one-way delay
  - Web page retrievals
- Inter-flow fairness
  - Four TCP flows, 10, 50, 200, 500 ms RTTs
- Transient behaviour
  - RRUL latency over time, from when competing flows start

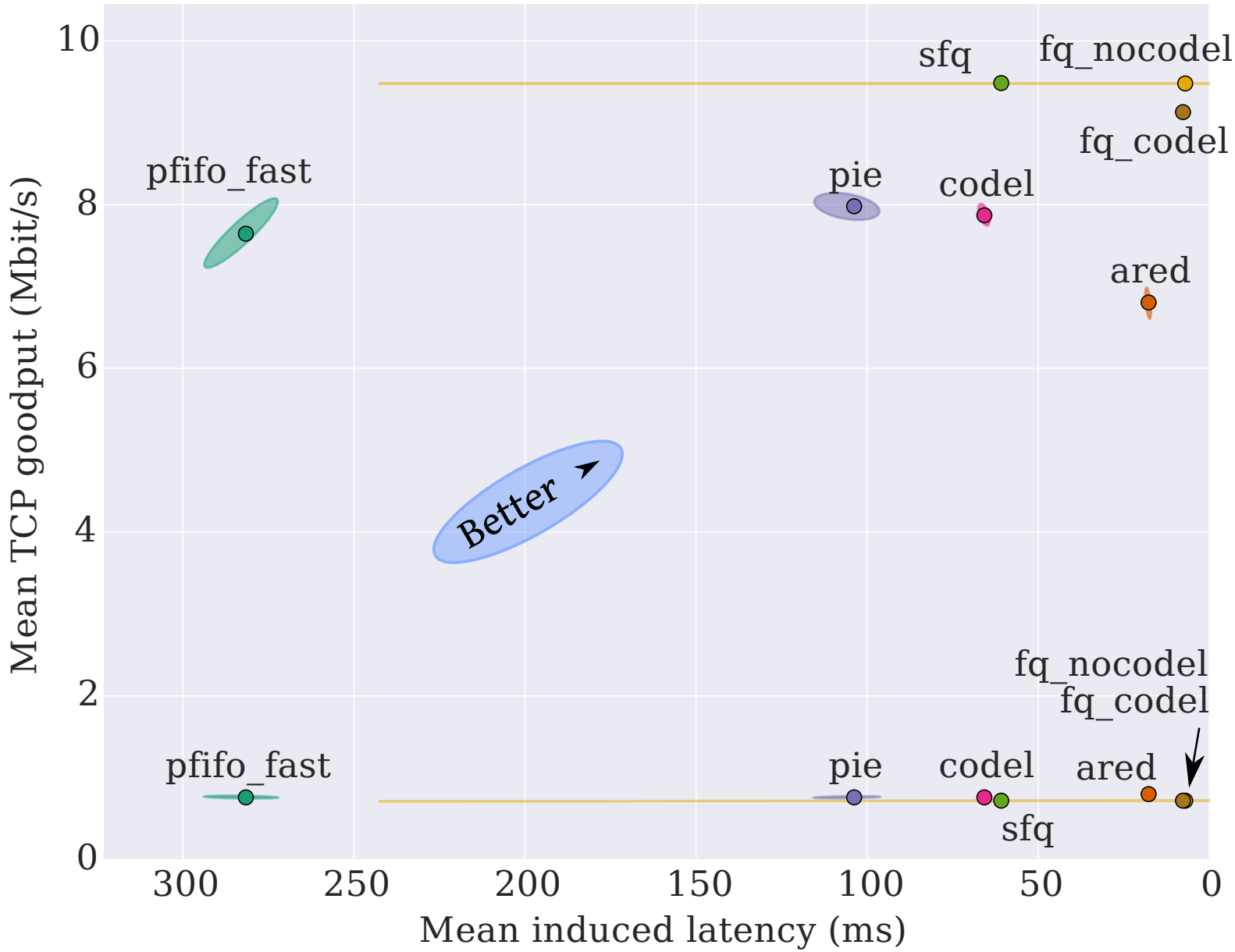
Paper under submission to USENIX NSDI '15

The scenarios

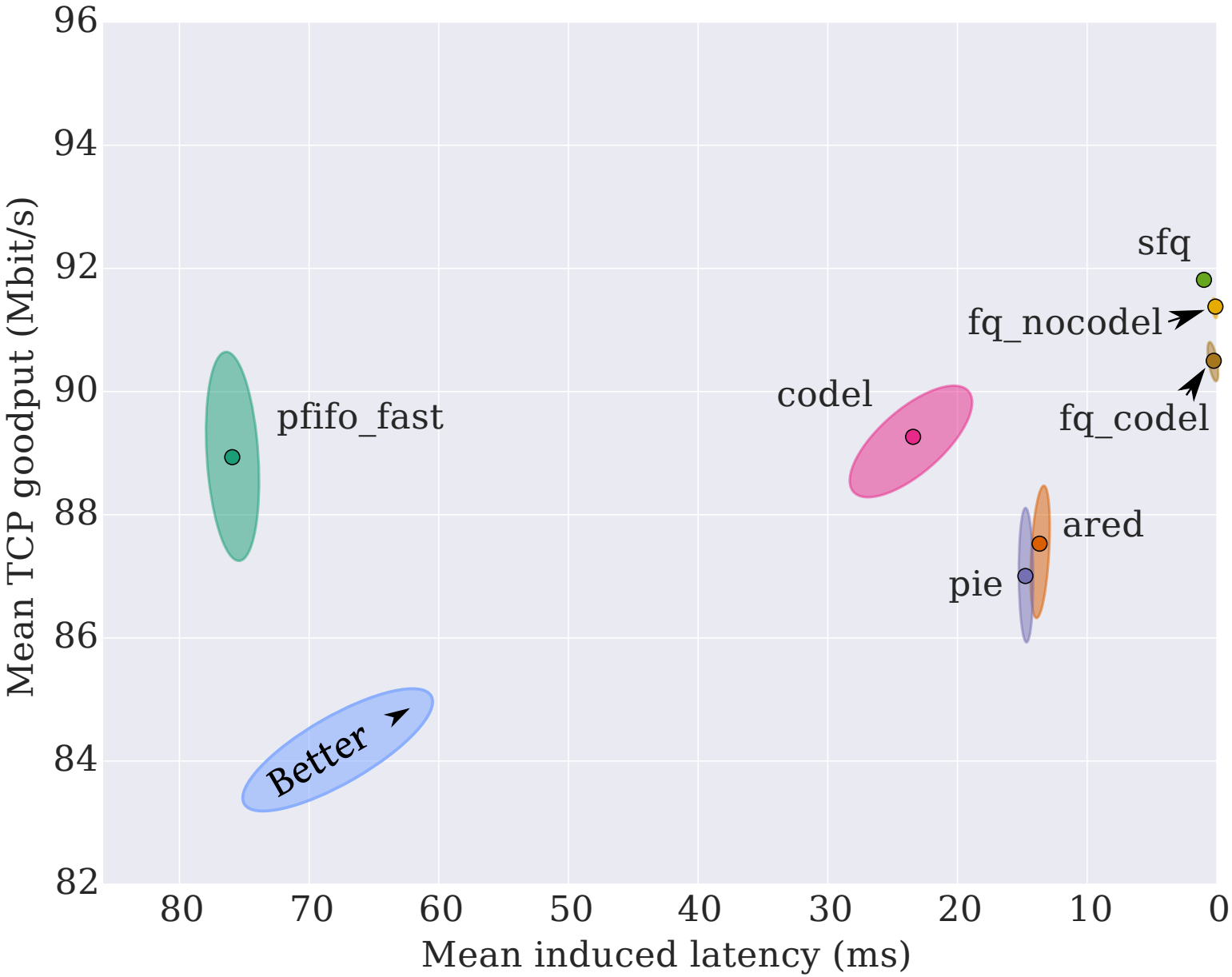
- Three bandwidth settings
  - 10/1, 10/10, 100/100 Mbps
- 50 ms base latency
- CUBIC TCP (except fairness tests)
- Three AQMs:
  - ARED
  - PIE
  - CoDel
- Three schedulers:
  - SFQ
  - fq_codel
  - fq_nocodel
- And pfifo_fast (Linux default FIFO queue)

The Good:
Steady state results

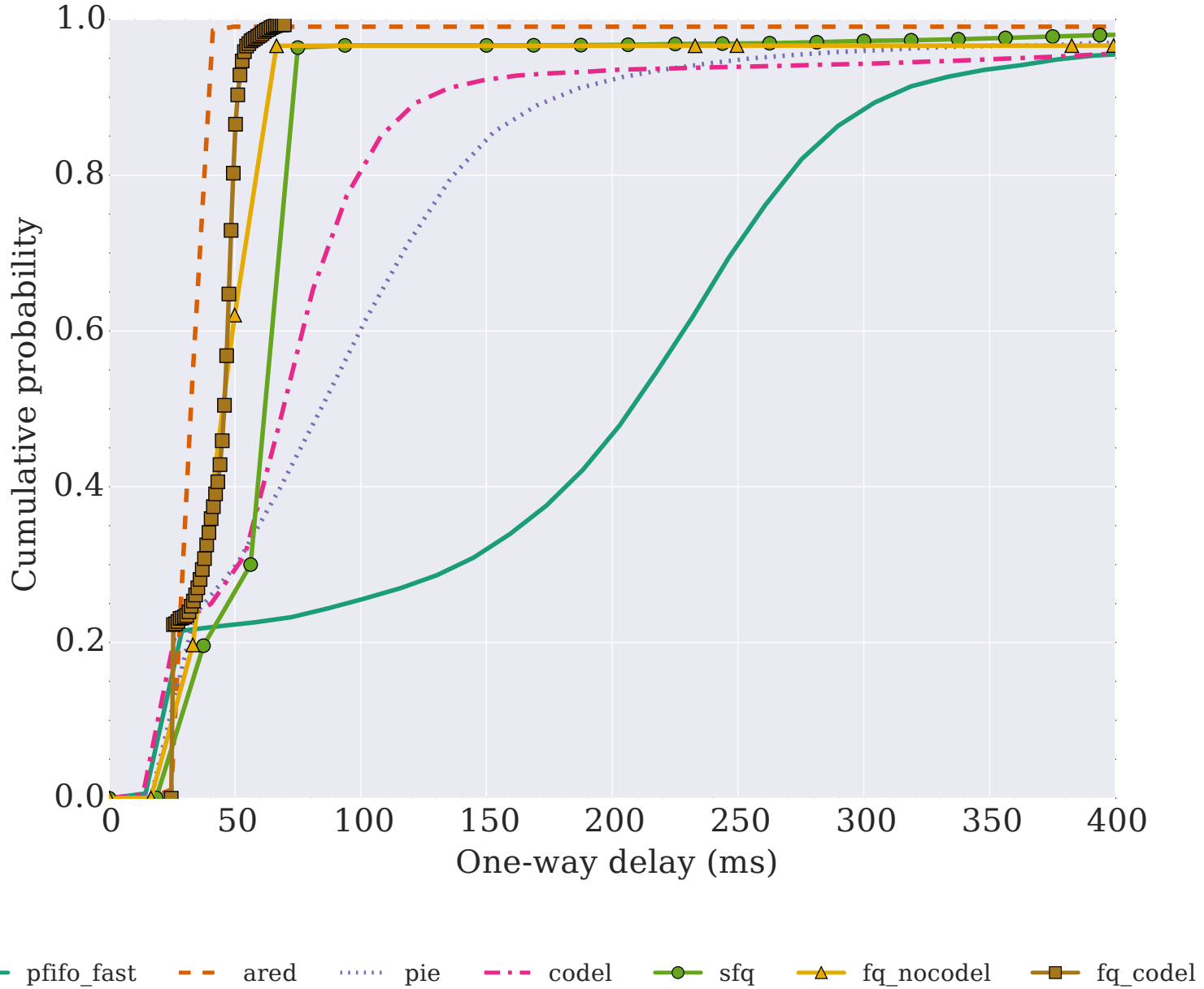
RRUL 10/1 Mbps



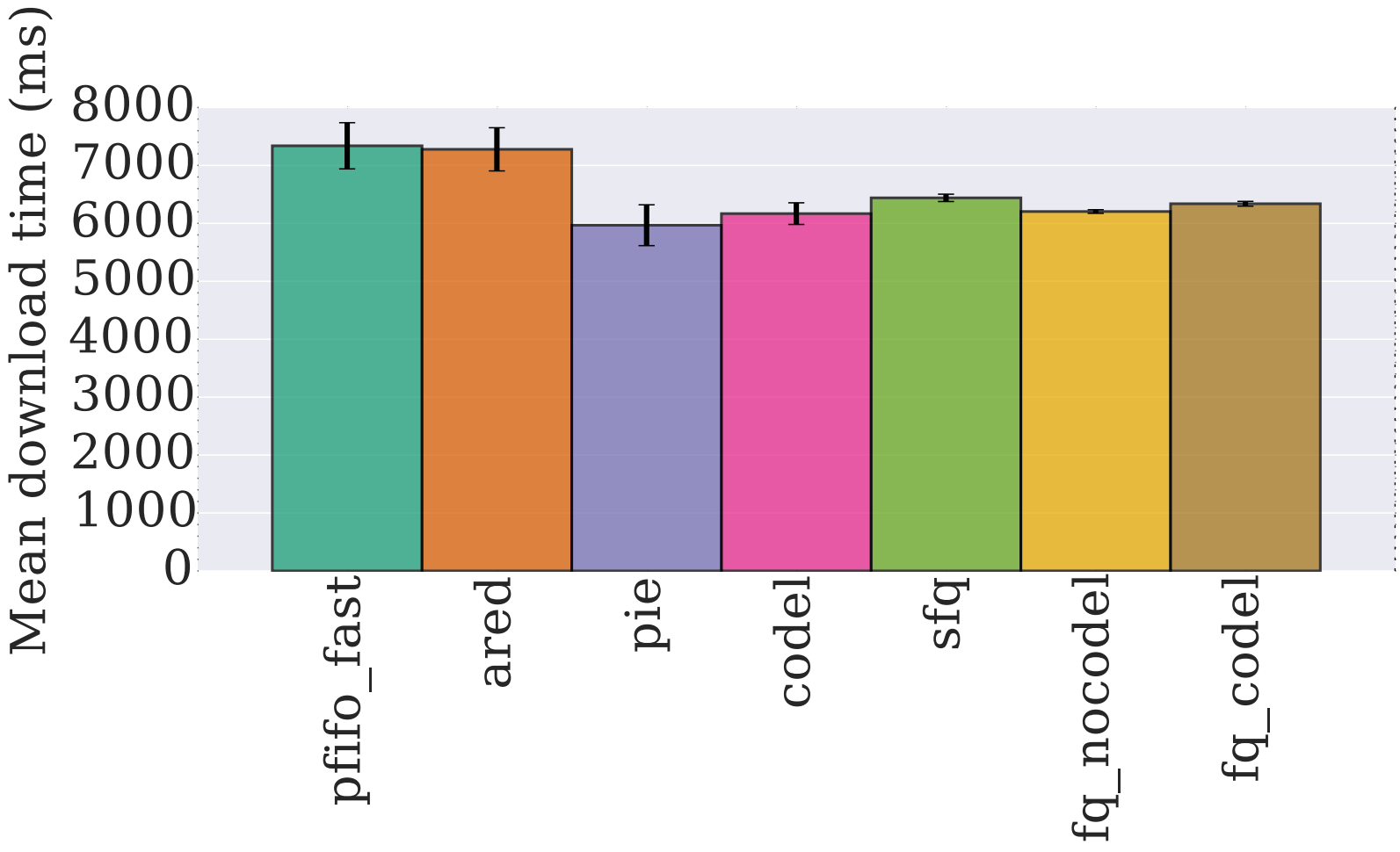
RRUL 100/100 Mbps



VoIP 10/1 Mbps

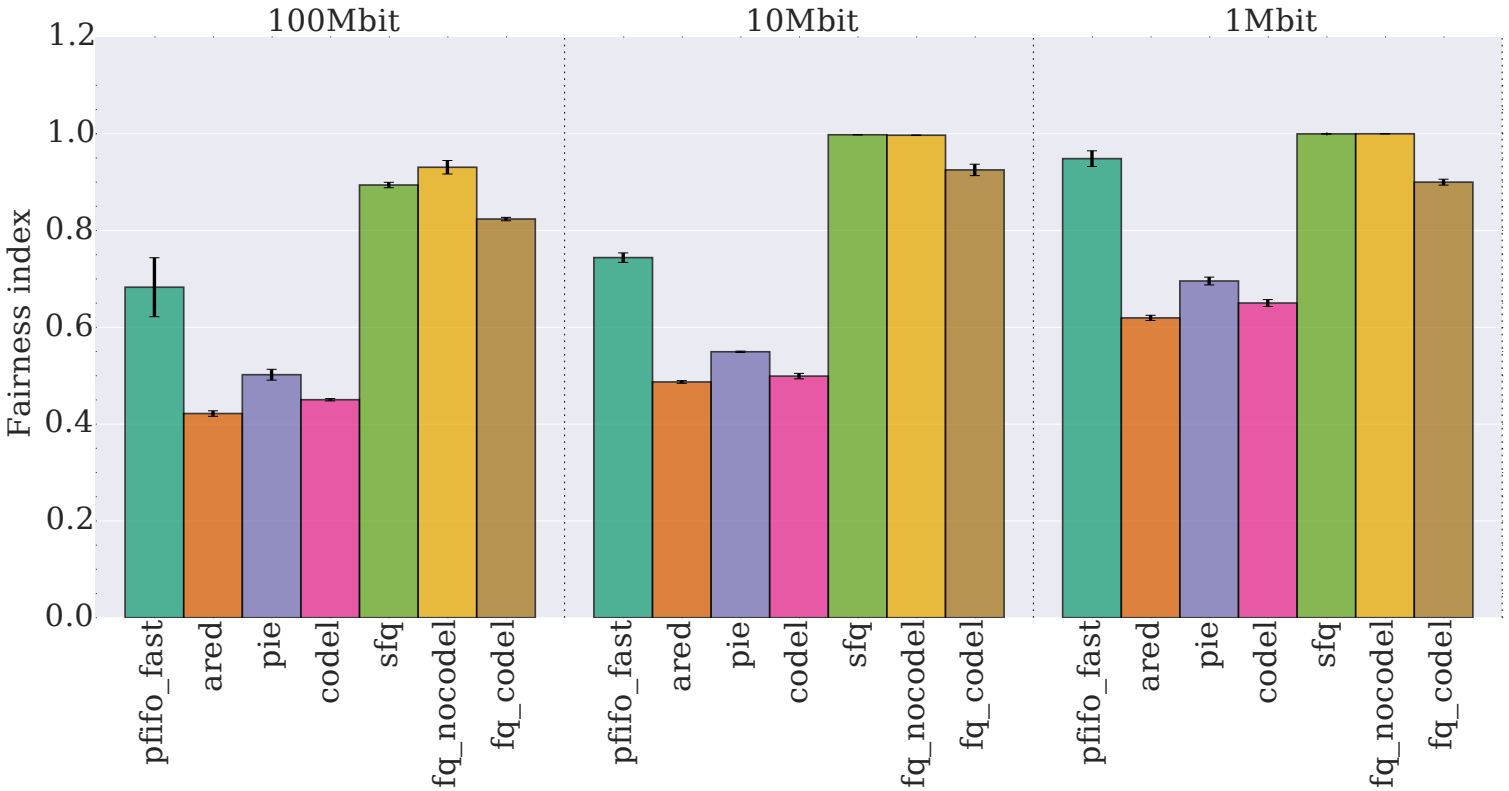


Web: Huffington Post 10/10 Mbps w/RRUL background



The Bad:
Inter-flow fairness

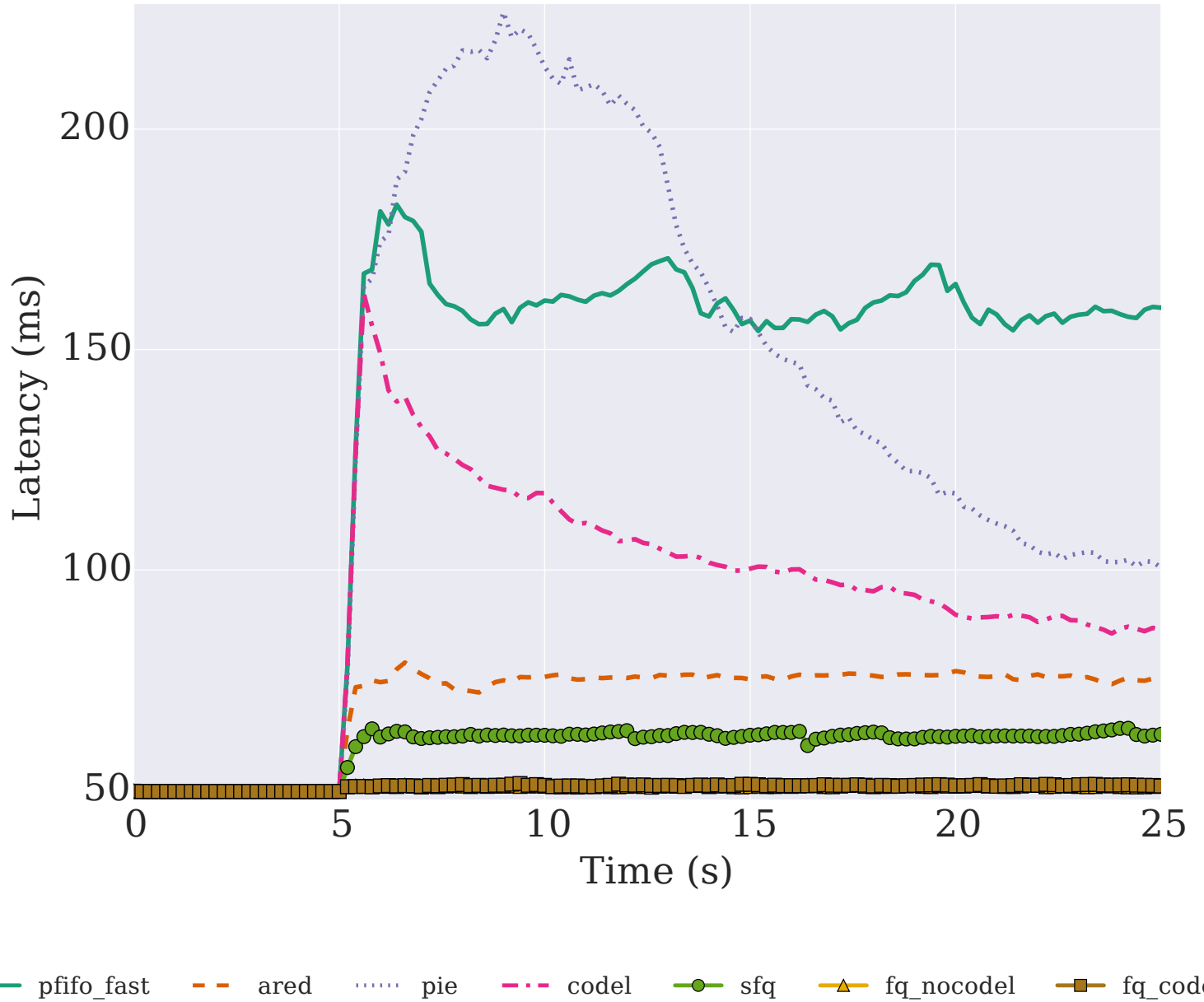
Fairness New Reno



Fairness flow throughput 100 Mbps

The Ugly: Transient behaviour

10/10 Mbps


Latency (ms)
200
150
100
50
0
5
10
15
20
25
Time (s)
pfifo_fast
ared
pie
codel
sfq
fq_nocodel
fq_codel

Summary of results

- The Good: Steady state behaviour
  - AQMs can significantly improve latency under load
  - FQ algorithms even more so
  - Although CoDel does have some issues at 100 Mbps
- The Bad: Inter-flow fairness
  - AQMs exacerbate TCP RTT unfairness
  - FQ does achieve almost perfect fairness
- The Ugly: Transient behaviour
  - AQMs take up to tens of seconds to contain latency at competing flow startup
  - FQ doesn't miss a beat

Outline

- Measuring network behaviour
- Measurement results
- **The netperf-wrapper testing tool**
- References and questions
- Appendix slides

Included tests

- Simple single-flow tests (ping, TCP ul/dl, UDP flood)
- Latency under load tests
  - 1 TCP flow up/down/bidirectional
  - RRUL variants
  - Periodic UDP bursts
  - On/off TCP flows
- RTT fairness tests
- Comparing TCPs (cubic, reno, westwood, ledbat)
- Application-specific (HTTP, VoIP)

Test specifications

```
DATA_SETS = o([
    ('TCP upload BE',
     {'command': find_netperf("TCP_STREAM", LENGTH, HOST,
                              marking="CS0,CS0"),
      'delay': DELAY, 'units': 'Mbits/s',
      'runner': 'netperf_demo',}),

    ('TCP upload BK',
     {'command': find_netperf("TCP_STREAM", LENGTH, HOST,
                              marking="CS1,CS1"),
      'delay': DELAY, 'units': 'Mbits/s',
      'runner': 'netperf_demo',}),

    ('TCP upload avg',
     {'apply_to': [glob("TCP upload*",
                        exclude=["TCP upload sum"])],
      'units': 'Mbits/s','runner': 'average',}),
```

Metadata collected automatically


```
"metadata": {
  "BATCH_NAME": "rrul",
  "BATCH_TIME": "2014-10-02T15:31:11.616664",
  "DATA_FILENAME": "batch-rrul-2014-10-02T153111-50ms-10Mbit-ared-cubic-01.json.gz",
  "EGRESS_INFO": {"bql": {"tx-0": "1879048192"},
                  "classes": null,
                  "driver": "e1000e",
                  "iface": "eth2",
                  "link_params": {"ether": "e8:39:35:14:03:31",
                                  "qlen": "1000"},
                  "nexthop": "10.60.1.2",
                  "offloads": {"generic-receive-offload": false,
                               "generic-segmentation-offload": false,
                               "large-receive-offload": false,
                               "tcp-segmentation-offload": false,
                               "udp-fragmentation-offload": false},
                  "qdiscs": [{"id": "0:",
                              "name": "pfifo_fast",
                              "params": {"0": "1","1": "","2": "0","bands": "3","priomap": "1","refcnt": "2"},
                              "parent": "root"}],
                  "src": "10.60.1.1",
                  "target": "10.60.4.2"},
  "GATEWAYS": [{"iface": "eth0","ip": "192.168.60.1"}],
  "HOST": "testserv-05",
  "HOSTS": ["testserv-05"],
  "IP_ADDRS": {"eth0": ["192.168.60.91","fe80::21e:4fff:fee6:3884"],
               "eth2": ["10.60.1.1","10.60.1.5","fe80::ea39:35ff:fe14:331"],
               "lo":   ["127.0.0.1","::1"]},
  "IP_VERSION": 4,
  "KERNEL_NAME": "Linux",
  "KERNEL_RELEASE": "3.14.4-tohojo-1",
  "LENGTH": 140,
  "LOCAL_HOST": "tohojo-testbed-01",
  "NAME": "rrul_be",
  "NETPERF_WRAPPER_VERSION": "0.7.0-git-cbbab94",
  "NOTE": "",
  "REMOTE_METADATA": {
    "testbed-02": {
      "EGRESS_INFO": {"bql": {"tx-0": "1879048192"},
                      "classes": [{"id": "1:1", "name": "tbf", "params": {"leaf": "2:"}, "parent": "1:"},
                                  {"id": "2:1", "name": "red", "params": {}, "parent": "2:"}],
```

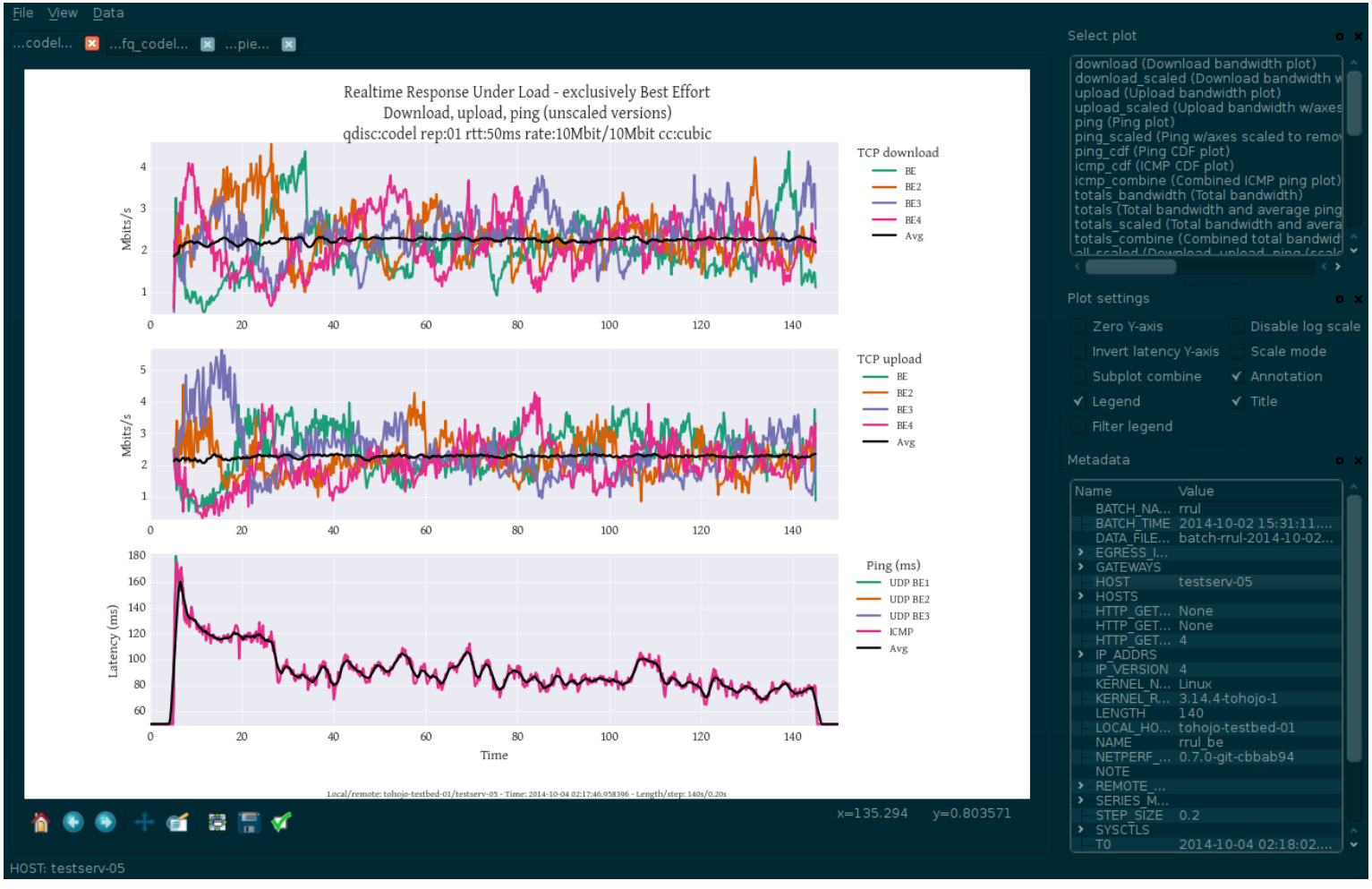
Batch facilities

```
[Batch::global]
# set options
ip_version = 4
length = 140
# build values from variable expansions
title = qdisc:${qdisc_label} rep:${repetition} rtt:${rtt} rate:${rate_down}/${rate_up} cc:${cc}
filename_extra = ${rtt}-${rate_up}-${qdisc_label}-${cc}-${repetition}
output_path = batch-${batch_time}/${batch_name}/${rate_up}-${repetition}
# run pre/post commands
commands = clear_caches, setup_qdiscs, tcpdump_client, tcpdump_egress, tcpdump_ingress, tcpdump_server
# iterate over arguments
for_qdiscs = ared, fq_codel, fq_nocodel, codel, pie, pfifo_fast, pfifo_fast_1000, sfq
for_bandwidths = 100mbit, 10mbit, 1mbit

# arguments
[Arg::pie]
inherits = global
qdisc_name = pie
qdisc_args =

# commands
[Command::tcpdump]
filename = ${data_filename}
exec = ssh ${hostname} "python tcpdump-wrapper.py start ${filename} -i ${interface} -s ${capsize}"
type = pre
essential = yes
```

GUI for exploring data sets



Installing netperf-wrapper

- Ubuntu/Debian: Go to `http://goo.gl/ysYJ7r`
- Arch Linux: Install from AUR.
- Others (including OSX w/macports):

```
$ sudo pip install netperf-wrapper
$ wget ftp://ftp.netperf.org/netperf/netperf-2.6.0.tar.gz
$ tar -xzf netperf-2.6.0.tar.gz
$ cd netperf-2.6.0
$ ./configure --enable-demo
$ make
$ sudo make install
```

Running the RRUL test

```
# Running the test
$ netperf-wrapper rrul netperf-west.bufferbloat.net \
    -t "IETF wifi test"

# Viewing the result -- PyQt4 installed
$ netperf-wrapper --gui <filename>.json.gz

# Viewing the result -- otherwise
$ netperf-wrapper -f plot <filename>.json.gz
```

Outline

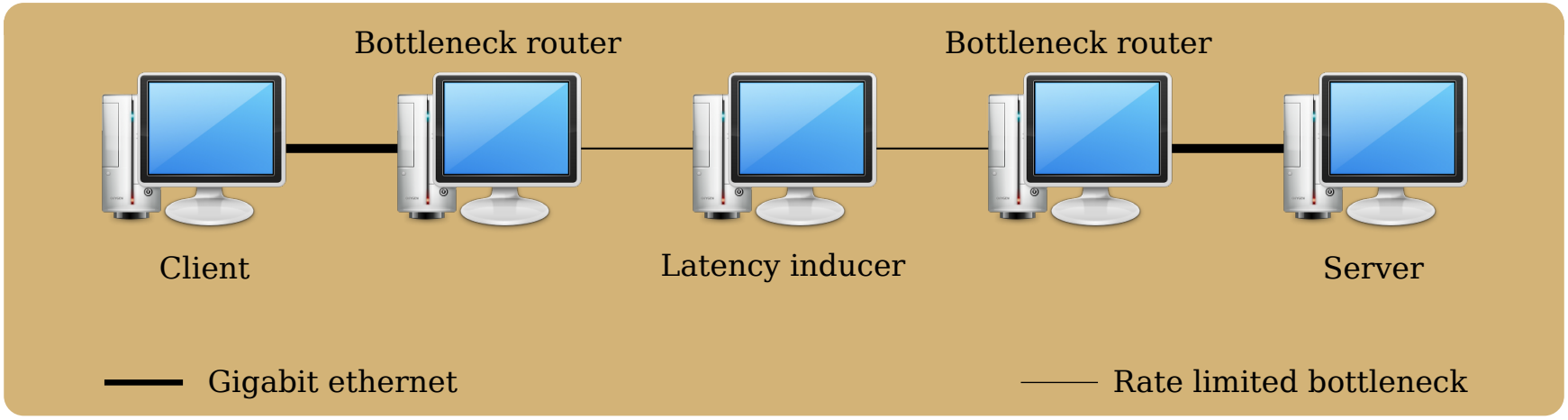
- Measuring network behaviour
- Measurement results
- The netperf-wrapper testing tool
- **References and questions**
- Appendix slides

References

- Software and websites:
  - The Bufferbloat project: `http://www.bufferbloat.net`
  - The CeroWrt router firmware: `http://www.bufferbloat.net/projects/cerowrt`
  - Netperf-wrapper: `https://github.com/tohojo/netperf-wrapper`
  - Test results dataset: `https://kau.toke.dk/modern-aqms/`
  - The RRUL test specification: `https://github.com/dtaht/deBloat/blob/master/spec/rrule.doc?raw=true`
- AQM algorithms:
  - Kathleen Nichols and Van Jacobson (2012). *Controlling queue delay*.
  - Rong Pan et al (2013). *PIE: A lightweight control scheme to address the bufferbloat problem*.
  - Sally Floyd, Ramakrishna Gummadi, and Scott Shenker (2001). *Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management*.
- FQ algorithms:
  - P.E. McKenney (1990). *Stochastic fairness queueing*.
  - M. Shreedhar and G. Varghese (1996). *Efficient fair queuing using deficit round-robin*.
  - M.H. MacGregor and W. Shi (2000). *Deficits for Bursty Latency-critical Flows: DRR++*.
  - T. Høiland-Jørgensen et al (2014). *FlowQueue-Codel*. `http://tools.ietf.org/html/draft-hoeiland-joergensen-aqm-fq-codel-01`.

Questions?

Test setup diagram

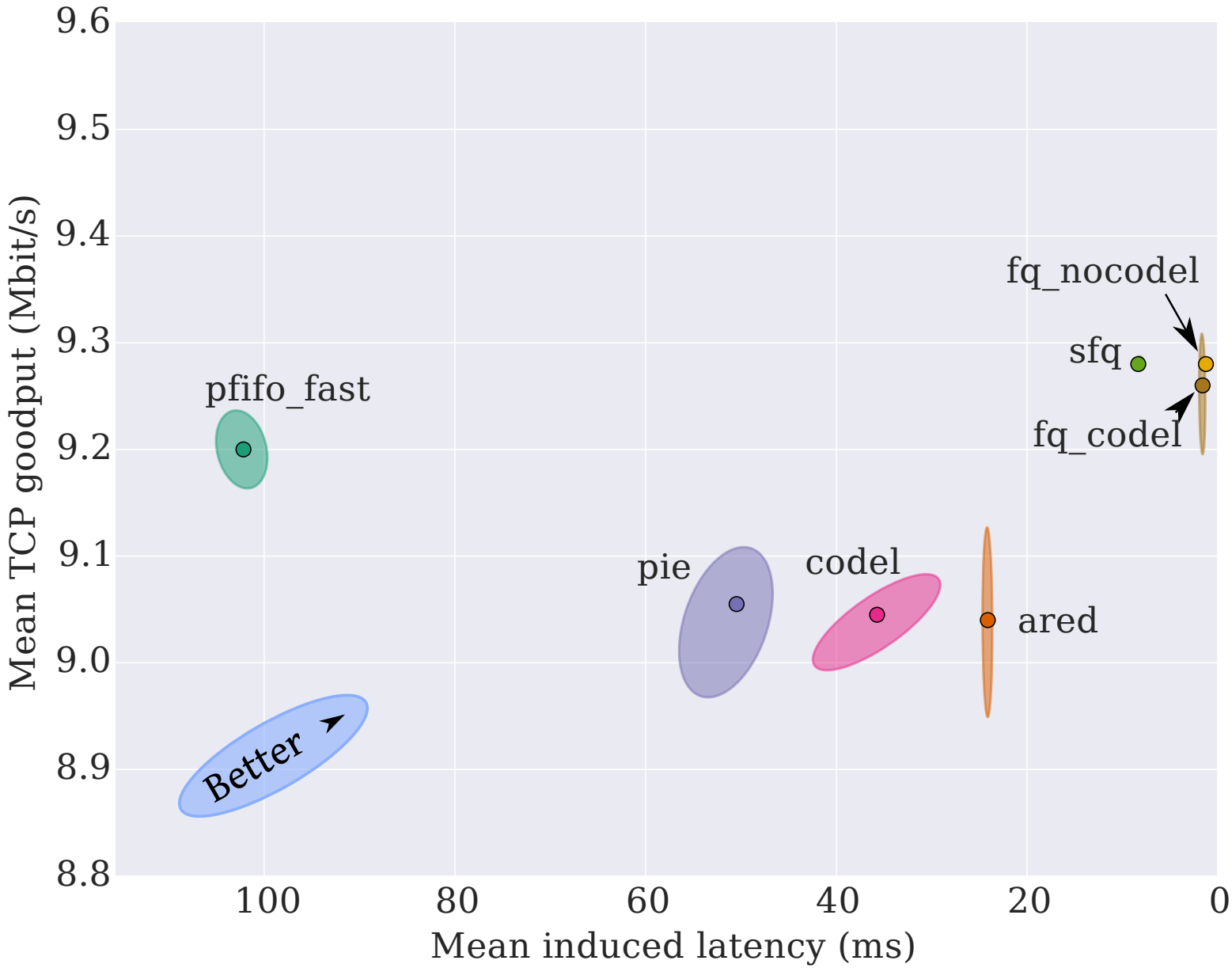


- Debian Linux – kernel v3.14
- Rate limiting via *tbf*, delay via *dummynet*

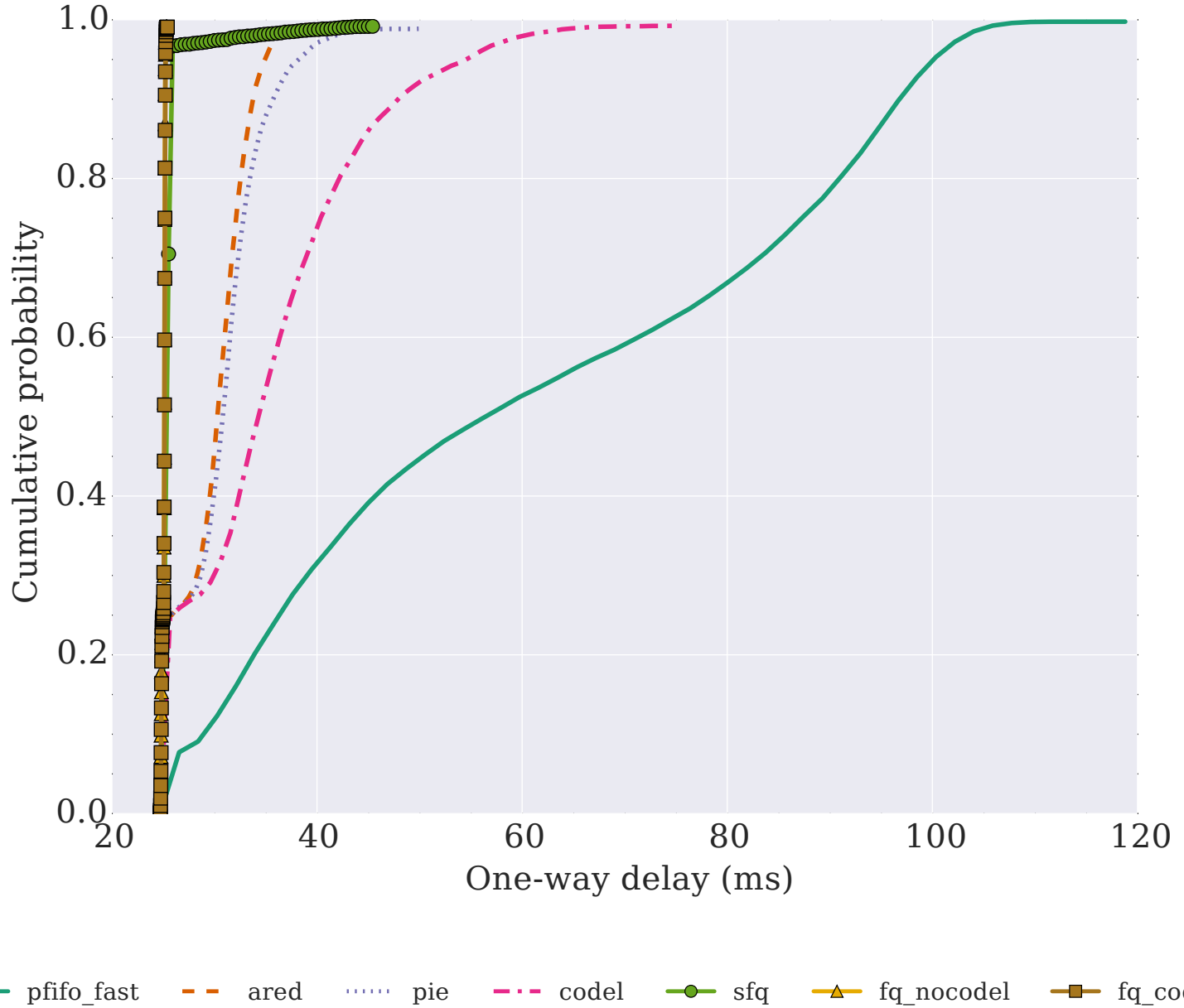
Parameterisation

	1 Mbps	10 Mbps	100 Mbps
pfifo_fast			
`txqueuelen`	127	127	*1000*
ARED			
`min`	1514	12500	125000
`bandwidth`	1 Mbps	10 Mbps	100 Mbps
`max`	3028	-	-
PIE			
`target`	*20 ms*	*20 ms*	*20 ms*
`tupdate`	*30 ms*	*30 ms*	*30 ms*
`limit`	*1000*	*1000*	*1000*
CoDel			
`target`	13 ms	*5 ms*	*5 ms*
`interval`	*100 ms*	*100 ms*	*100 ms*
`limit`	*1000*	*1000*	*1000*
SFQ			
`limit`	*127*	*127*	1000
fq_codel			
`target`	13 ms	*5 ms*	*5 ms*
`interval`	*100 ms*	*100 ms*	*100 ms*
`limit`	*10240*	*10240*	*10240*
fq_nocodel			
`limit`	127	127	1000
`interval`	100 s	100 s	100 s

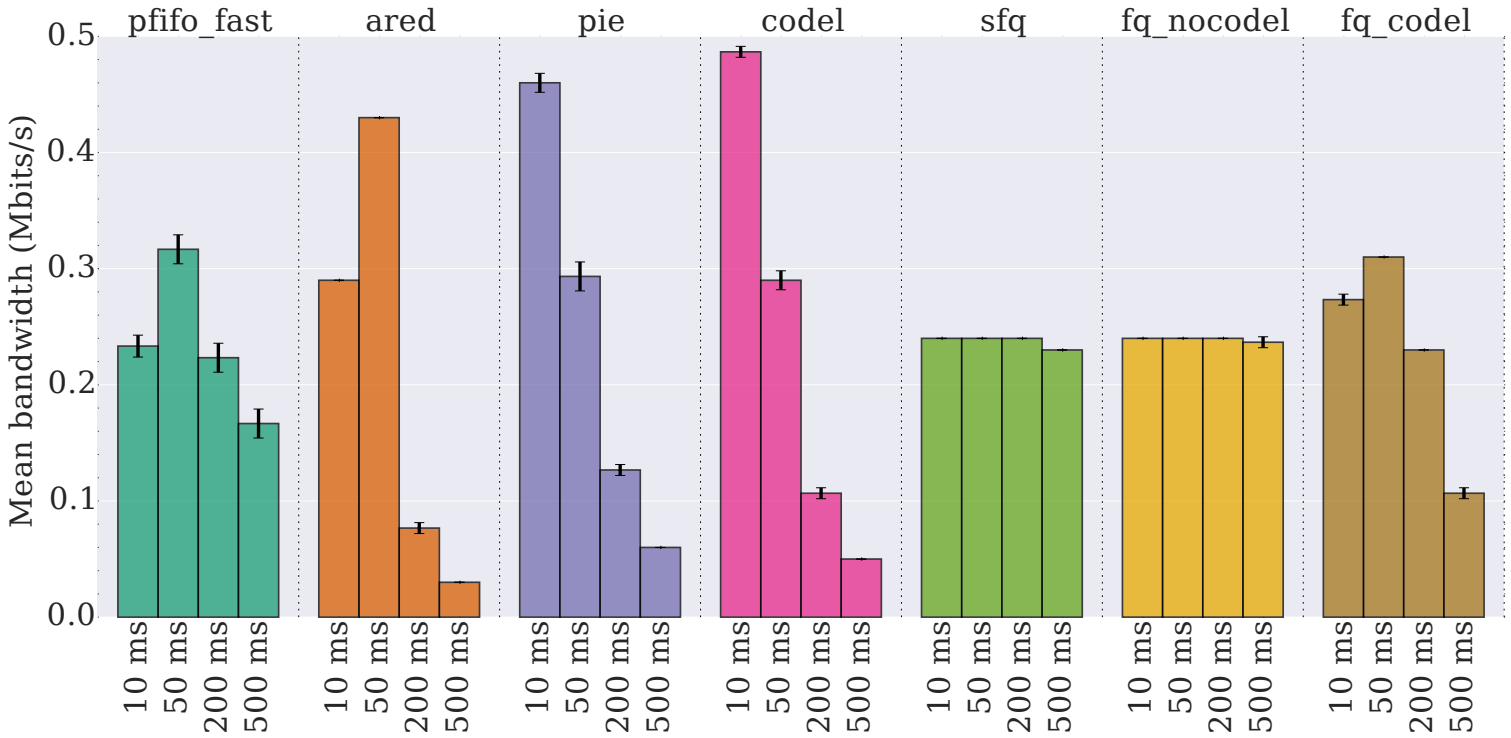
RRUL 10/10 Mbps



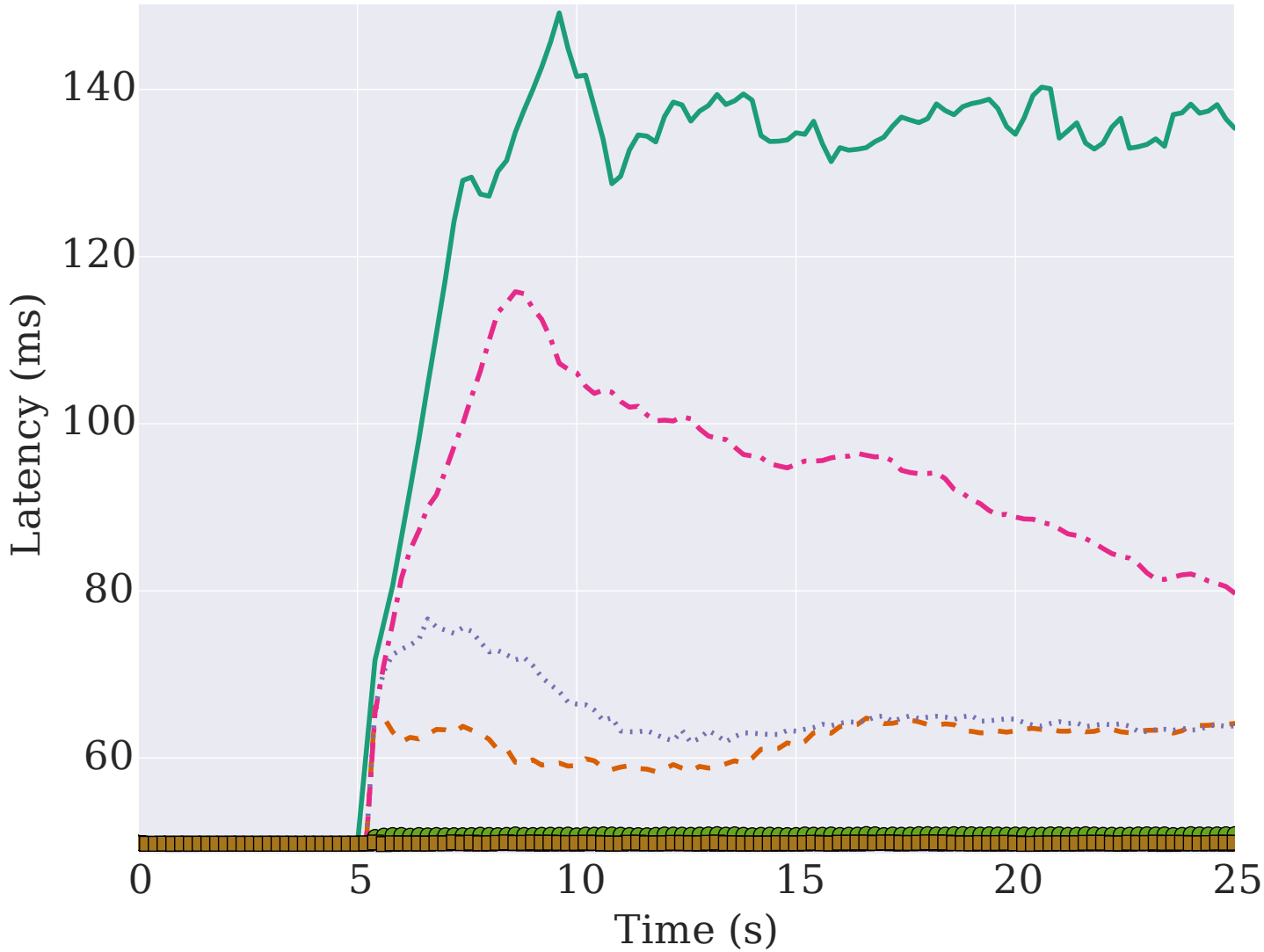
Steady-state VoIP 100/100 Mbps



Fairness flow throughput 1 Mbps



Transient behaviour, 100/100 Mbps



Transient behaviour, 10/1 Mbps