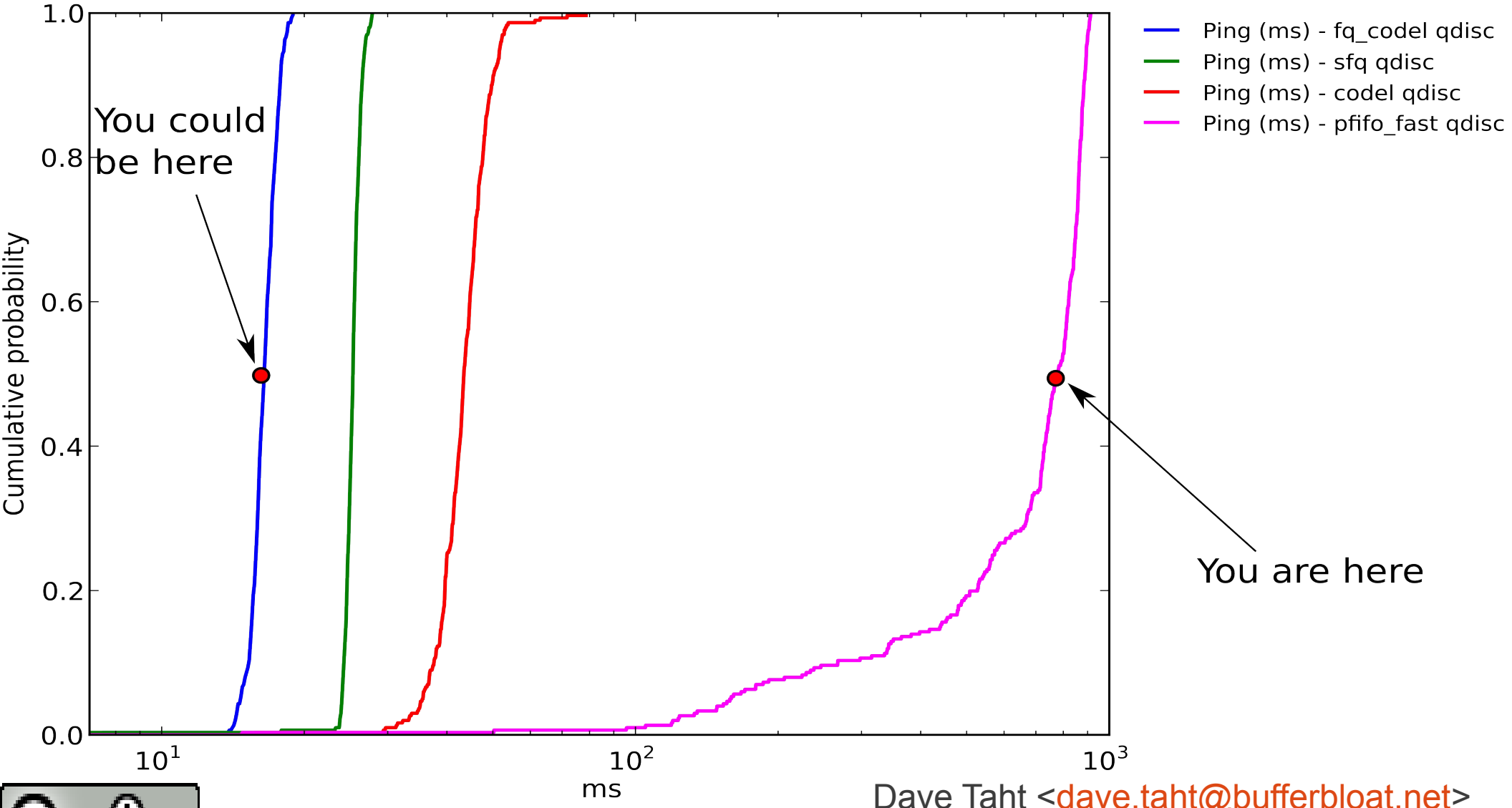
Beating Bufferbloat with FQ_Codel



Dave Taht <dave.taht@bufferbloat.net>
Co-Founder, Bufferbloat.net

Bufferbloat

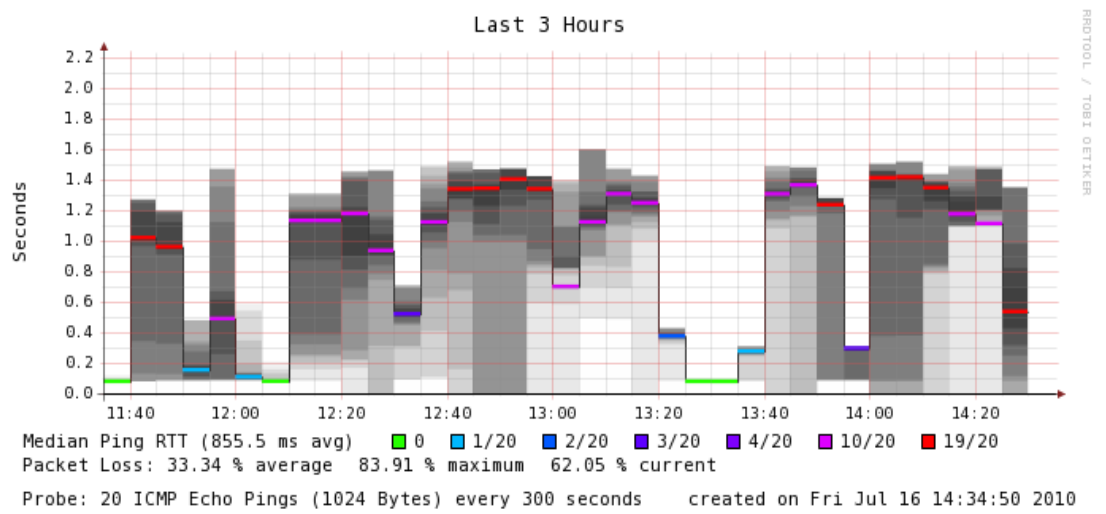
- Wikipedia: “ *a phenomenon in a packet-switched computer network whereby excess buffering of packets inside the network causes high latency and jitter, as well as reducing the overall network throughput.*”

- Bufferbloat is really two things:
  - Excessive buffering at the device, device driver, network queue and tcp/udp queue layers in network stacks on all modern operating systems (windows, mac, Linux, etc)
  - Lack of good packet scheduling and active queue management at ANY layer in all Oses and in common edge devices such as home network gateways, dslams, and cable head ends.
  - (A very few DSL edge networks implement “shortest queue first”, which helps a lot)
- You only see the latency spikes when under load.
- Queues are usually either empty, or full.
- All sorts of loads exist, from constant, to transient. Transient spikes exist, but are hard to see. Easy to feel or hear, however. It's easier to create constant loads and measure against those... but not necessarily an accurate representation of reality.

This is Bufferbloat!

bash

```
Request timeout for icmp_seq 202
Request timeout for icmp_seq 203
Request timeout for icmp_seq 204
Request timeout for icmp_seq 205
Request timeout for icmp_seq 206
Request timeout for icmp_seq 207
Request timeout for icmp_seq 208
Request timeout for icmp_seq 209
Request timeout for icmp_seq 210
Request timeout for icmp_seq 211
Request timeout for icmp_seq 212
64 bytes from 74.125.230.112: icmp_seq=131 ttl=58 time=82507.940 ms
64 bytes from 74.125.230.112: icmp_seq=132 ttl=58 time=81507.469 ms
64 bytes from 74.125.230.112: icmp_seq=133 ttl=58 time=80506.771 ms
64 bytes from 74.125.230.112: icmp_seq=134 ttl=58 time=79506.587 ms
64 bytes from 74.125.230.112: icmp_seq=135 ttl=58 time=78507.154 ms
64 bytes from 74.125.230.112: icmp_seq=136 ttl=58 time=77506.916 ms
64 bytes from 74.125.230.112: icmp_seq=137 ttl=58 time=76515.519 ms
64 bytes from 74.125.230.112: icmp_seq=138 ttl=58 time=75637.441 ms
64 bytes from 74.125.230.112: icmp_seq=139 ttl=58 time=74814.515 ms
64 bytes from 74.125.230.112: icmp_seq=170 ttl=58 time=51346.859 ms
64 bytes from 74.125.230.112: icmp_seq=171 ttl=58 time=50445.706 ms
64 bytes from 74.125.230.112: icmp_seq=172 ttl=58 time=49868.339 ms
64 bytes from 74.125.230.112: icmp_seq=173 ttl=58 time=48868.145 ms
64 bytes from 74.125.230.112: icmp_seq=174 ttl=58 time=47867.477 ms
64 bytes from 74.125.230.112: icmp_seq=175 ttl=58 time=46867.589 ms
64 bytes from 74.125.230.112: icmp_seq=176 ttl=58 time=46201.652 ms
64 bytes from 74.125.230.112: icmp_seq=177 ttl=58 time=45221.153 ms
64 bytes from 74.125.230.112: icmp_seq=178 ttl=58 time=44460.582 ms
64 bytes from 74.125.230.112: icmp_seq=179 ttl=58 time=43479.863 ms
64 bytes from 74.125.230.112: icmp_seq=180 ttl=58 time=42498.994 ms
64 bytes from 74.125.230.112: icmp_seq=181 ttl=58 time=41498.114 ms
64 bytes from 74.125.230.112: icmp_seq=182 ttl=58 time=40518.115 ms
64 bytes from 74.125.230.112: icmp_seq=183 ttl=58 time=39516.909 ms
64 bytes from 74.125.230.112: icmp_seq=184 ttl=58 time=38534.589 ms
64 bytes from 74.125.230.112: icmp_seq=185 ttl=58 time=37533.669 ms
64 bytes from 74.125.230.112: icmp_seq=186 ttl=58 time=36552.234 ms
64 bytes from 74.125.230.112: icmp_seq=187 ttl=58 time=35692.118 ms
64 bytes from 74.125.230.112: icmp_seq=188 ttl=58 time=34713.787 ms
64 bytes from 74.125.230.112: icmp_seq=189 ttl=58 time=33749.172 ms
64 bytes from 74.125.230.112: icmp_seq=190 ttl=58 time=32773.104 ms
64 bytes from 74.125.230.112: icmp_seq=191 ttl=58 time=31809.864 ms
64 bytes from 74.125.230.112: icmp_seq=192 ttl=58 time=30809.192 ms
64 bytes from 74.125.230.112: icmp_seq=193 ttl=58 time=29824.379 ms
64 bytes from 74.125.230.112: icmp_seq=194 ttl=58 time=28848.364 ms
64 bytes from 74.125.230.112: icmp_seq=195 ttl=58 time=27962.353 ms
64 bytes from 74.125.230.112: icmp_seq=196 ttl=58 time=26982.090 ms
64 bytes from 74.125.230.112: icmp_seq=197 ttl=58 time=26000.297 ms
64 bytes from 74.125.230.112: icmp_seq=198 ttl=58 time=25024.054 ms
64 bytes from 74.125.230.112: icmp_seq=199 ttl=58 time=24038.550 ms
64 bytes from 74.125.230.112: icmp_seq=201 ttl=58 time=22056.121 ms
64 bytes from 74.125.230.112: icmp_seq=200 ttl=58 time=23057.466 ms
64 bytes from 74.125.230.112: icmp_seq=202 ttl=58 time=21094.977 ms
64 bytes from 74.125.230.112: icmp_seq=203 ttl=58 time=20114.617 ms
64 bytes from 74.125.230.112: icmp_seq=204 ttl=58 time=19153.674 ms
64 bytes from 74.125.230.112: icmp_seq=205 ttl=58 time=18197.613 ms
```

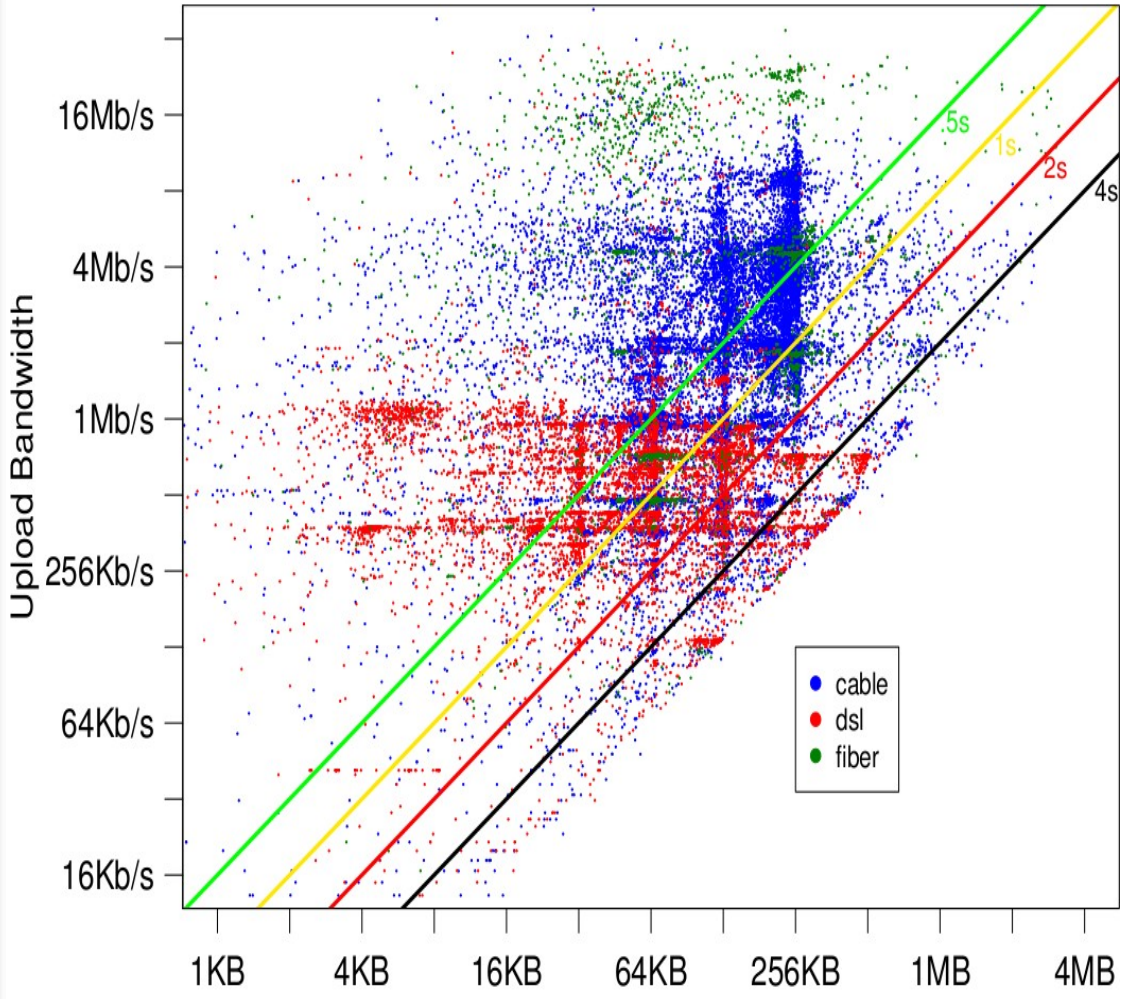


Image by: **Stöckli**, Nelson, Hasler
Laboratory for Atmospheres

USGS

Hurricane Linda west of Mexico
September 9, 1997 17:45 UTC

Where we are now

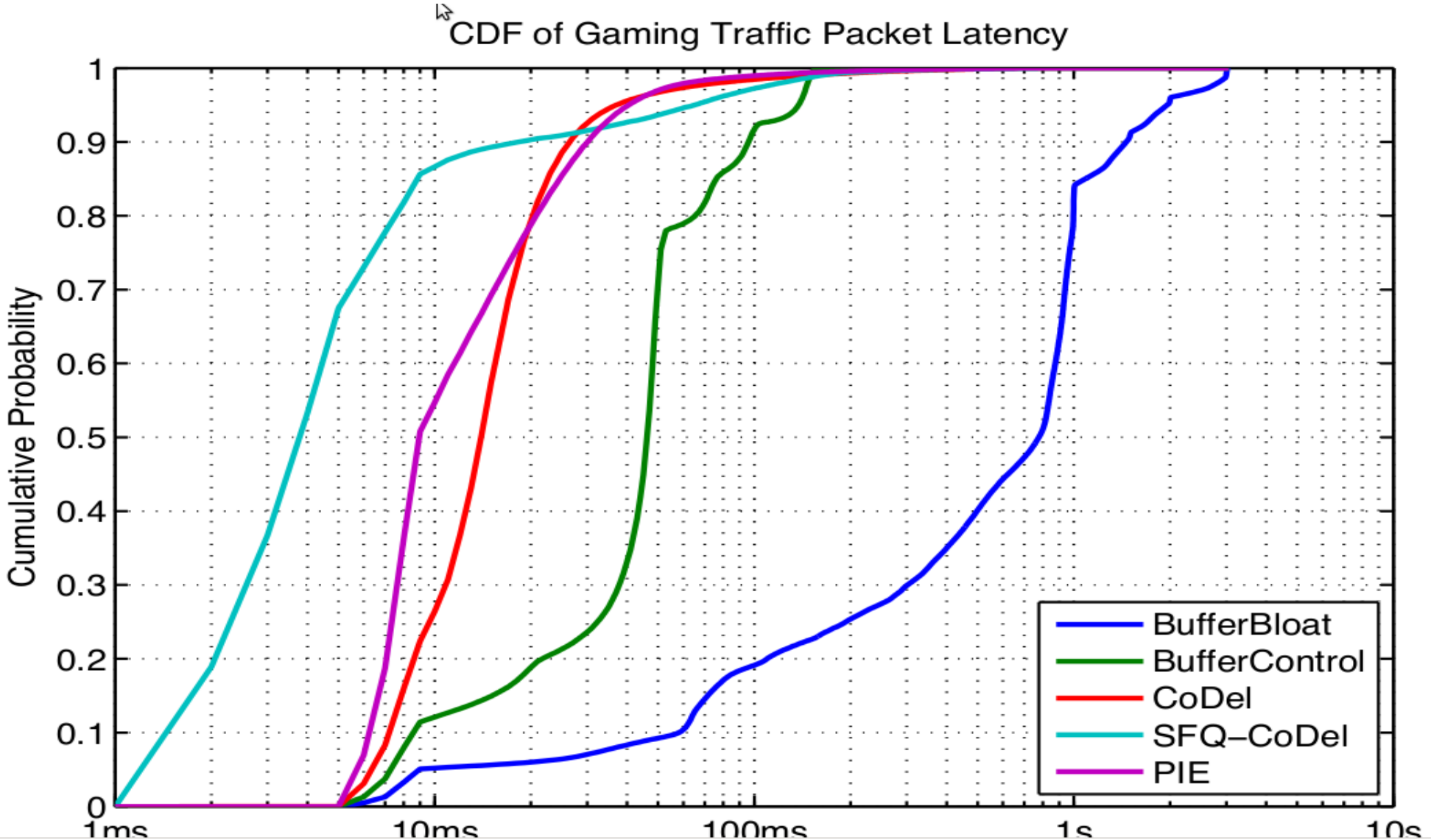
- We seem to have won!
  - fq_codel in increasingly wide deployment
    - Vast improvements in web, voip, and gaming traffic
    - Huge throughput increases as a side effect!
    - “It's no longer worth even talking about tail drop queues.”
      - Andrew Mcgregor, IETF core co-chair
  - New latency under load Tests (RRUL)
  - Cablelabs Simulation Study
  - Some very convincing demos
  - IETF “aqm” mailing list

How we've Fixed Bufferbloat

- Better Packet Scheduling (SFQ-like or DRR-like proven to work)
  - 5 Tuple Flow Queuing
  - “Sparse” packet optimizations - “new” packets go to the head of the flow queue
  - Scales past 10GigE
- Smarter Packet Drop policy (codel)
  - Designed by Kathie Nichols and Van Jacobson as a RED replacement
  - Drop packets from Fattest flows in a TCP-friendly way
  - Works with variable bandwidth

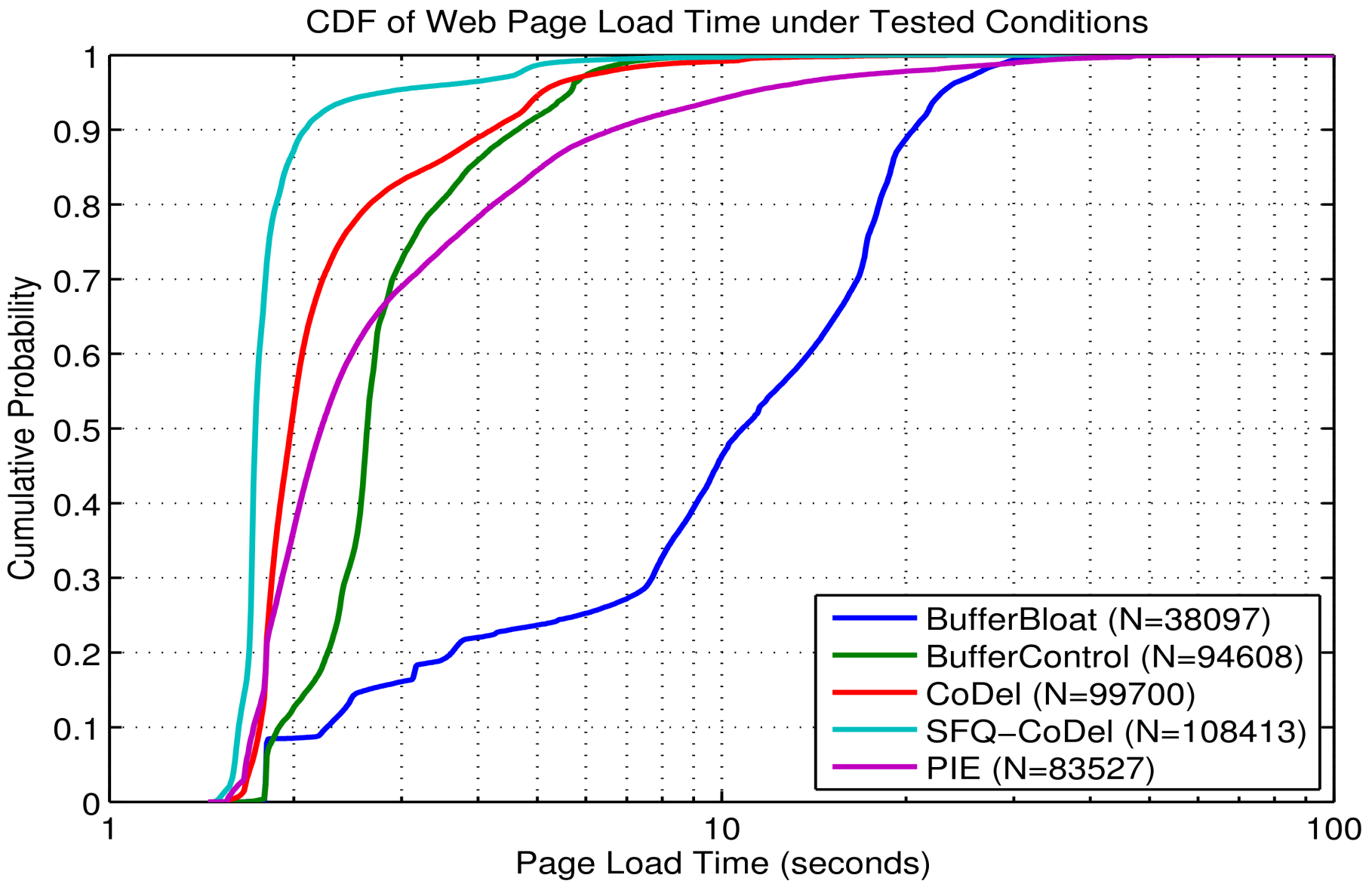
Combination of the two algorithms allows for “head drop” rather than tail drop from fat queue***s***.

Gaming traffic win with sfq_codel



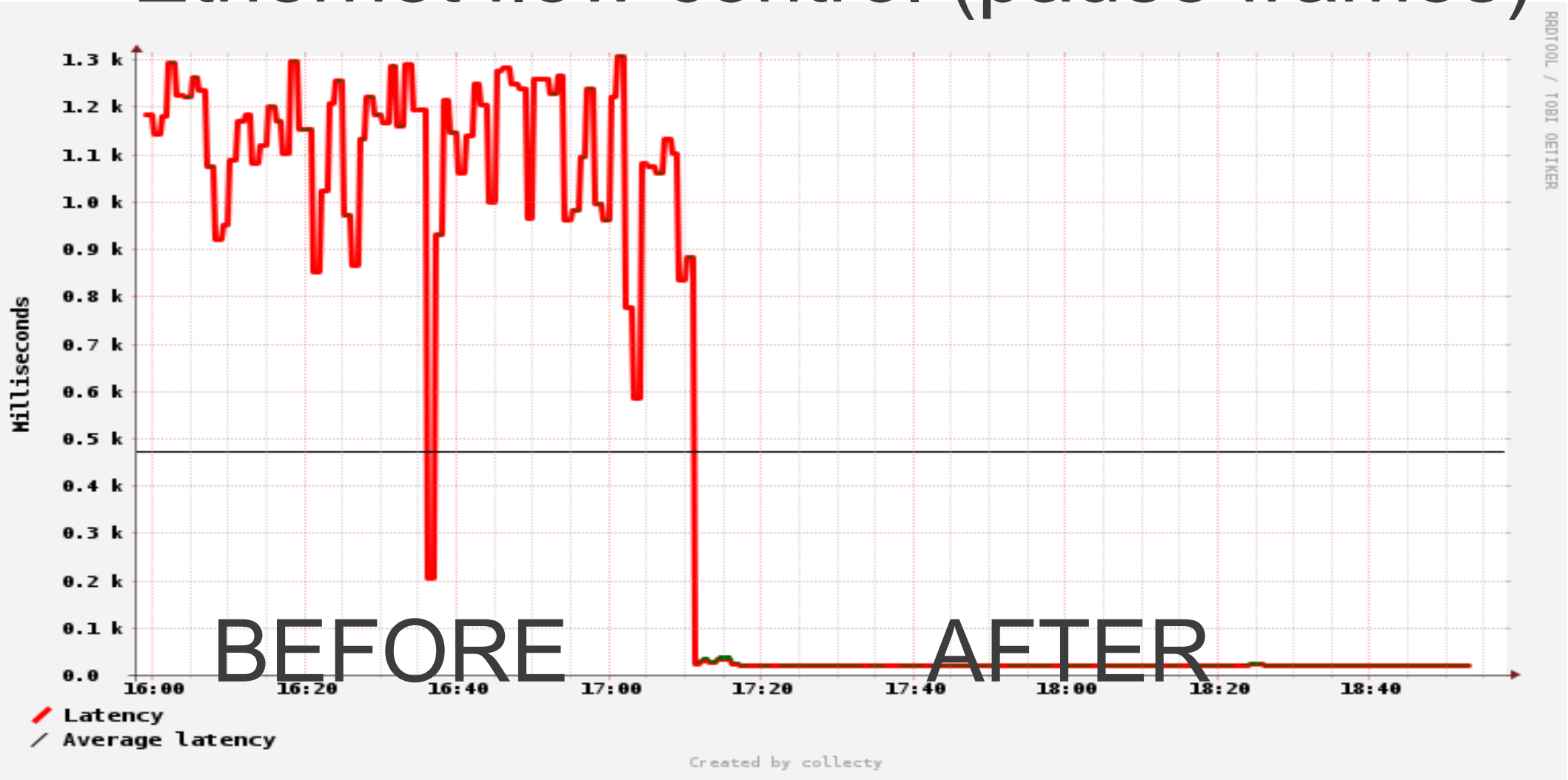
(cablelabs iccrg report)

Web Pages do even better



Video at: http://www.youtube.com/watch?v=NuHYOu4aAqg

ADSL modem latency w/FQ_Codel with: Ethernet flow control (pause frames)



- http://planet.ipfire.org/post/ipfire-2-13-tech-preview-fighting-bufferbloat

Realtime Response Under Load test (RRUL)

- Tests 4 up, 4 down TCP streams against icmp and udp traffic.
- Also tries diffserv (802.11e) classification
- Json data output
- Native Plots
- Web Interface

Extensions

- rrul46compete: RRUL using ipv4 and ipv6 at the same time.
- rtt_fair: test tcp performance between two or more hosts to see if a system is RTT-fair (meaning that connections to machines at different distances eventually or not get a fair share of the bandwidth)
- reno_cubic_westwood_lp: test performance of different TCPs

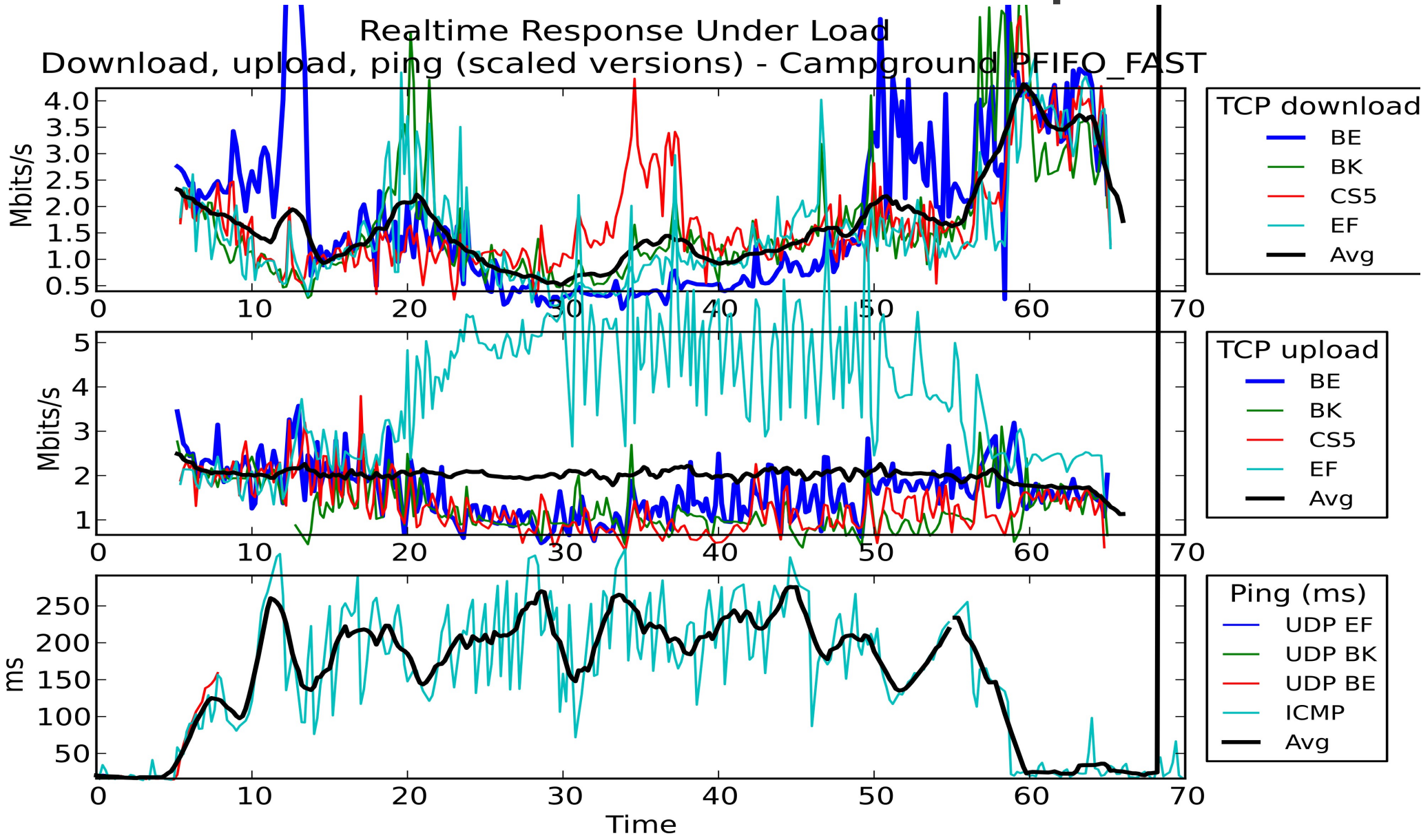
Simpler Tests

- tcp_bidirectional: a basic test intended to give a "textbook" result of two competing streams against a ping
- tcp_upload: multiple tcp uploads against ping
- tcp_download: multiple tcp downloads against ping

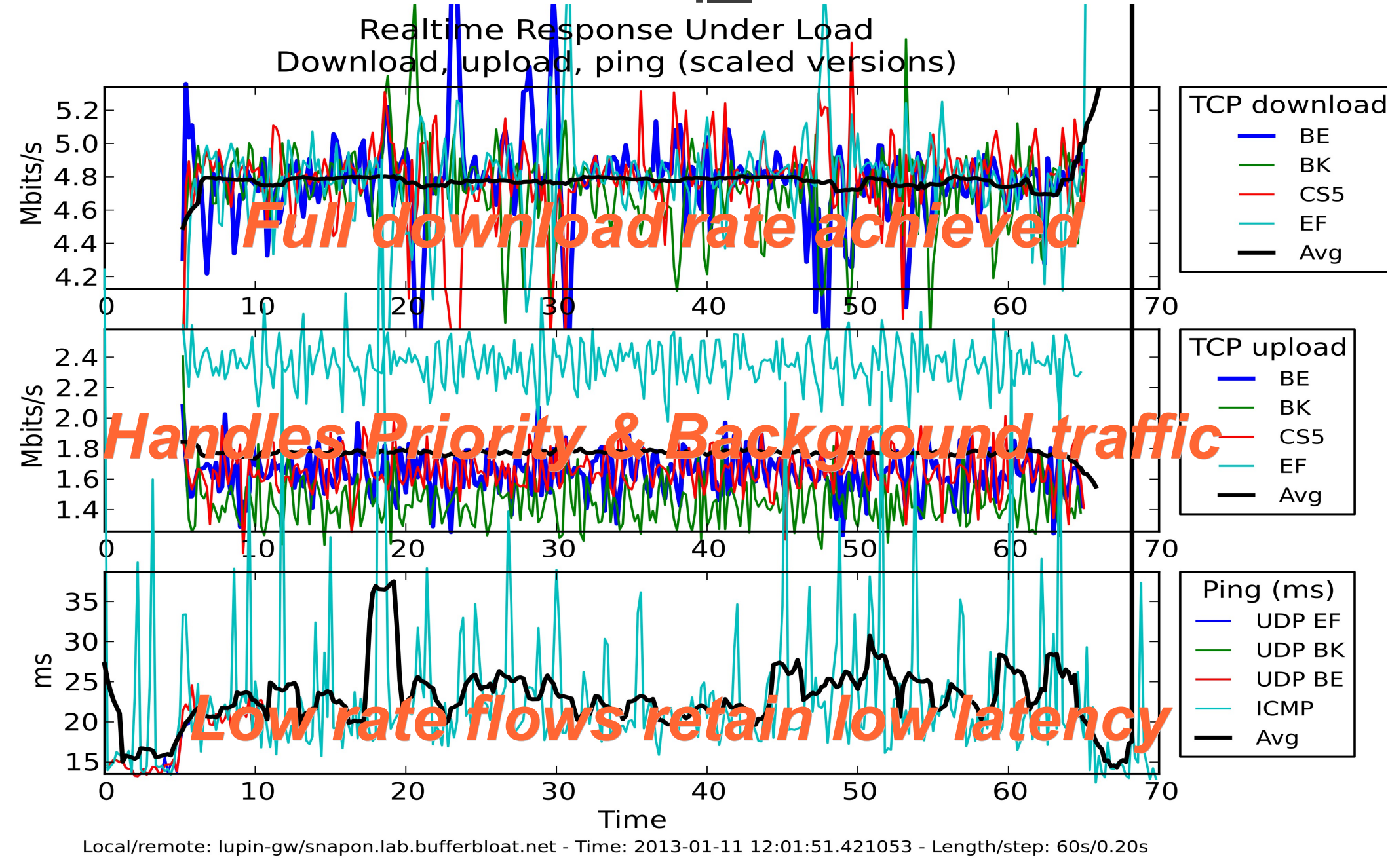
RRUL test helps

- See latency under load as induced by TCP behavior
- See behavior of other applications when under load
- Analyze problems in classification or routing
- Show things like that “smaller buffers are not better”...

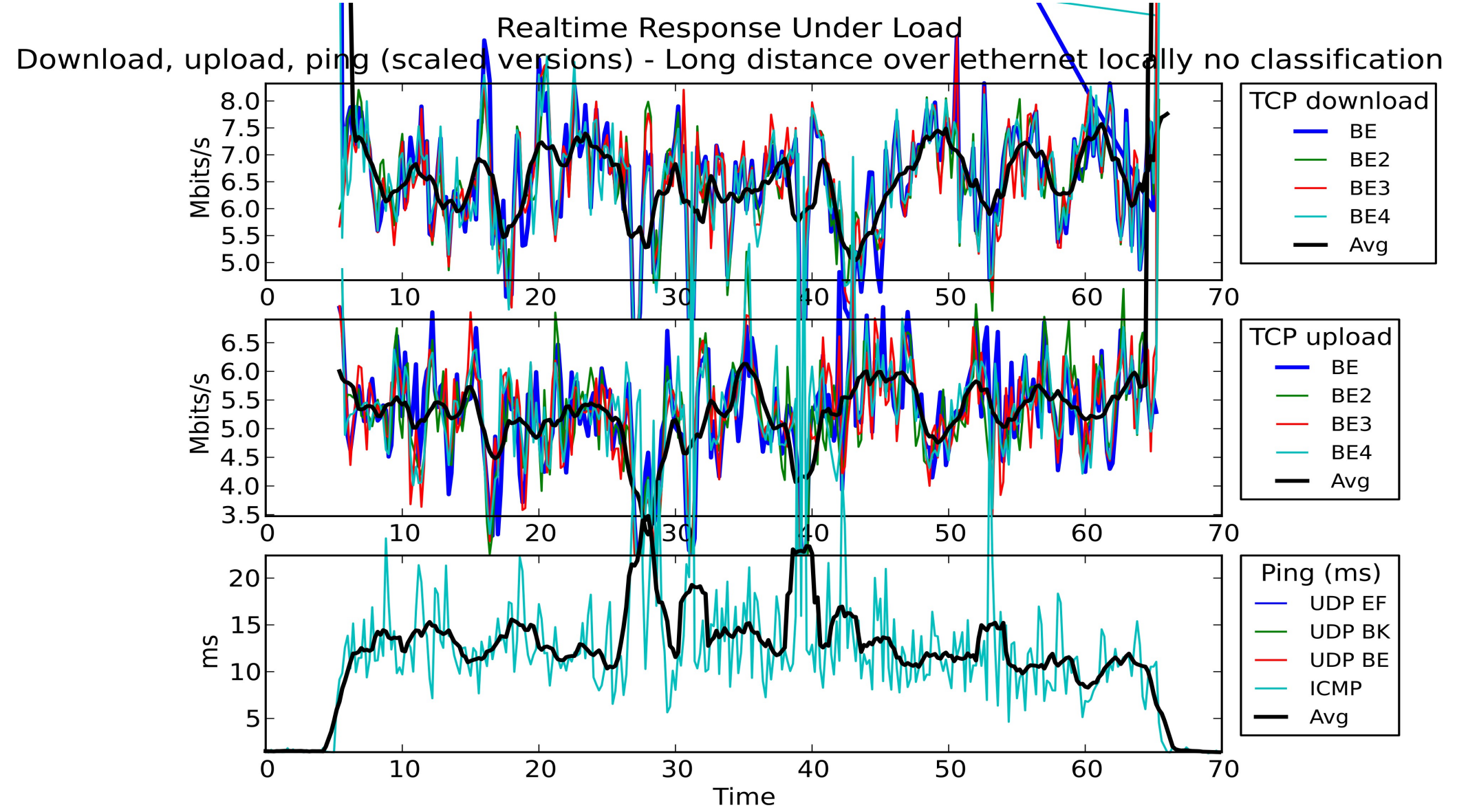
Current cable modem performance 20Mbit Down/8Mbit up



Cable modem performance w htb + 3 tier fq_codel

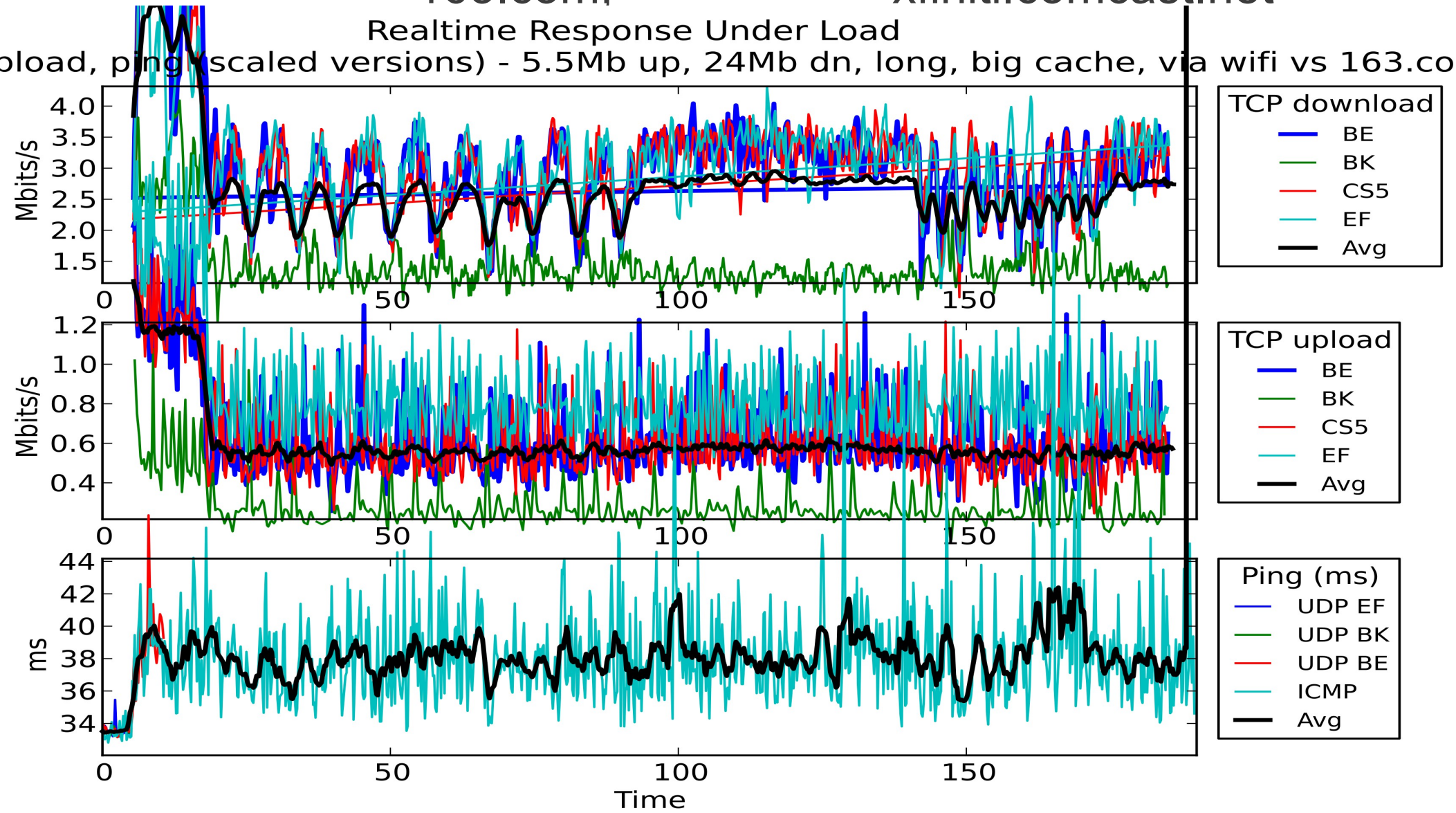


Wifi Latency under load (CeroWrt to CeroWrt)



(htb + nfq_codel) RRUL test vs Chrome Web Page Benchmark

163.com, xfiniti.comcast.net



More on the RRUL test...

While I have data sets of google hangouts, audio streaming, voip, gaming and bittorrent against the RRUL test... against various combinations of pfifo_fast (drop tail), codel, ns2 codel, fq_codel, nfq_codel, cake...I didn't have time to plot them all.

Prototypes of the RRUL test suite are available at:

- https://github.com/tohojo/netperf-wrapper
- Give it a try yourself! The CDF plots are great, too! Please upload your interesting rrul plots to the RRUL Rogues Gallery
- Paper: http://akira.ruc.dk/~tohojo/bufferbloat/bufferbloat-final.pdf
- Major server/test expansion is in the works
- Huge thanks to Toke Høiland-Jørgensen <toke@toke.dk> for turning 1/3 of the RRUL specification into code in 2 months flat.

“FQ_Codel provides great isolation... if you've got low-rate videoconferencing and low rate web traffic they never get dropped. A lot of issues with IW10 go away, because all the other traffic sees is the front of the queue. You don't know how big its window is, but you don't care because you are not affected by it. FQ_Codel increases utilization across your entire networking fabric, especially for bidirectional traffic...”

“If we're sticking code into boxes to deploy codel, *don't do that*.

Deploy fq_codel. It's just an across the board win.”

- ***Van Jacobson***

IETF 84 Talk

Fq_Codel deployment Status (4/15/13)

- fq_codel is now the default on all interfaces in OpenWrt/CeroWrt
- IpFire QoS
- Linux kernel mainline since Linux 3.5, ubuntu, fedora, arch support among many others
- Google deployment
- Trials in Android
- Preliminary BSD OS work commencing
- Under evaluation at cablelabs and elsewhere
- Standardization effort starting at ietf – join the new “aqm”mailing list!

Where should fq_codel be deployed?

- Anywhere there is a fast to slow transition on the internet
  - Edge networks (cable, lte, 3g, wifi, dsl)
  - Underlying hardware/rate limited services on Virtual Machines
  - Load balancers
  - highly interactive services like Web, Voip, gaming, videoconferencing services
- Clients/servers ***all*** benefit from the “fq” component and configuration can be made transparent, and ***on by default***
- Lastly - we generally don't care all that much about the “core” of the internet... where the overhead of these algorithms is high... but there are 2.4 billion people on the edges that will benefit from wide deployment of this technology.

*Fq_codel Licensing... none!

- Codel algorithm placed in the public domain
- Ns2 codel and sfqcodel code available under the BSD license
- Ns3 codel and fq_codel are BSD/GPL
- Linux GPL/BSD (for codel), GPL (fq_codel)

GO FORTH, TEST, AND DEPLOY!

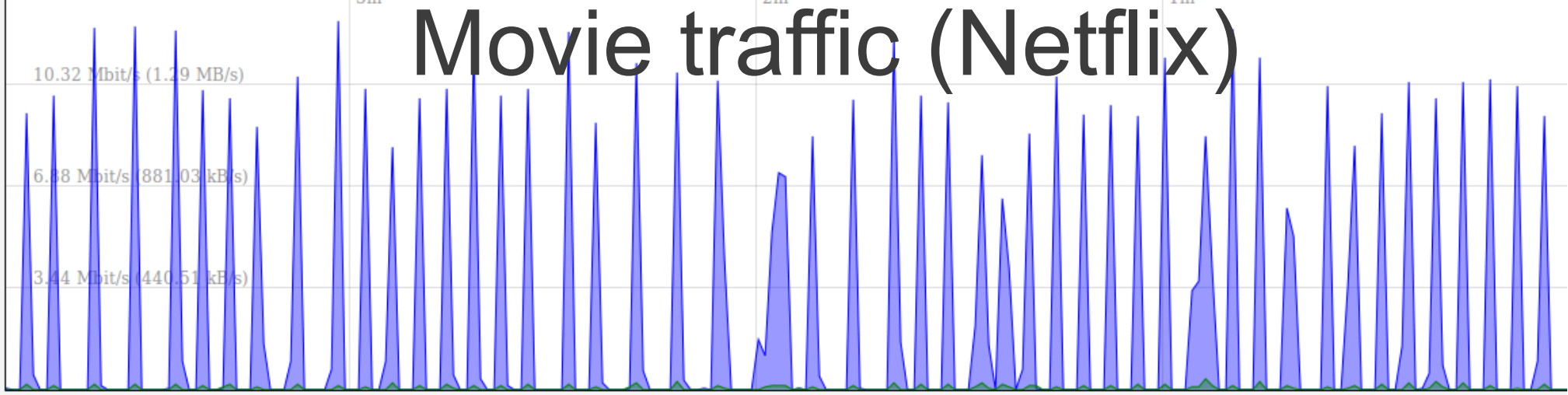
Challenges Ahead

- Big Challenge -
  - fq_codel is needed in cable modems, GPON, LTE, wifi and other technologies in the end user devices and the head ends
  - End user devices look easy to fix – cpu glut
  - Fatter servers (like load balancers), also
  - Line cards, dslams, far less so – but the overhead is more in the rate limiter than the AQM!

Further challenges

- Quest for a full replacement for PFIFO_Fast (diffserv support)
- Adding support for software rate limiting at higher rates
- Other delay based AQM systems (fq_PIE, etc)
- Further research into the interrelationship of drop mechanisms and fair queuing
- Developing better tests
- Pouring codel and fq_codel derivatives into hardware and other operating systems
- Coaxing the universe to try it and deploy it
- And there are a few problematic protocols like uTP and DASH, and new ones like webrtc, that need to be looked at harder

Movie traffic (Netflix)

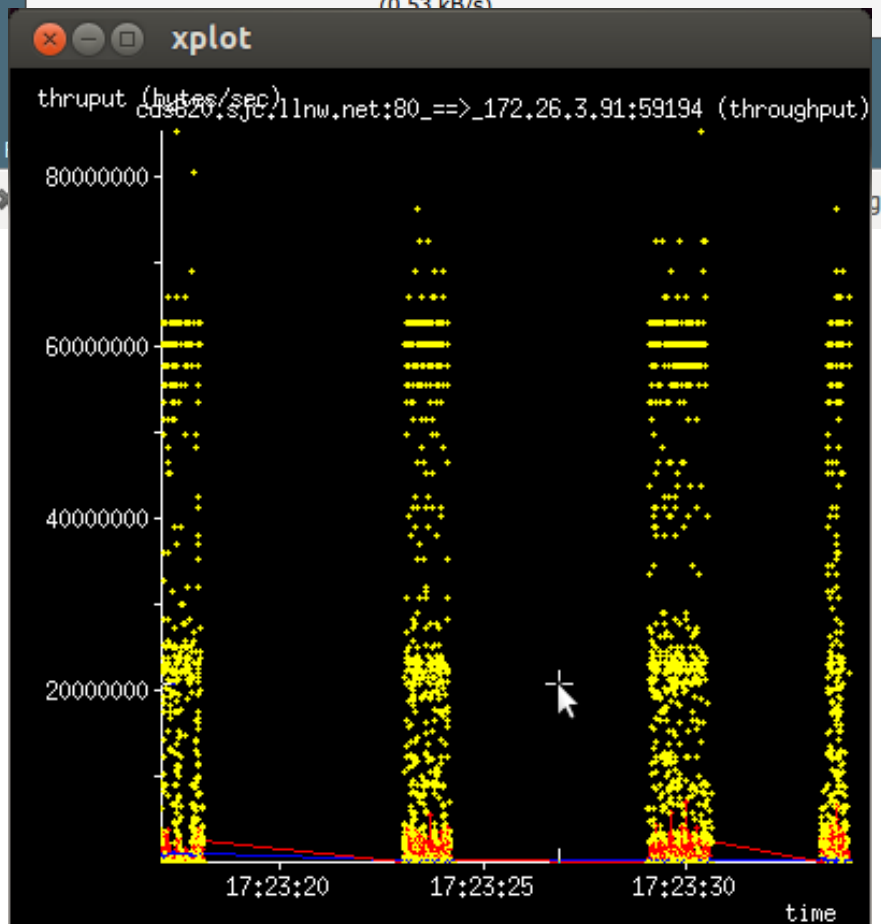


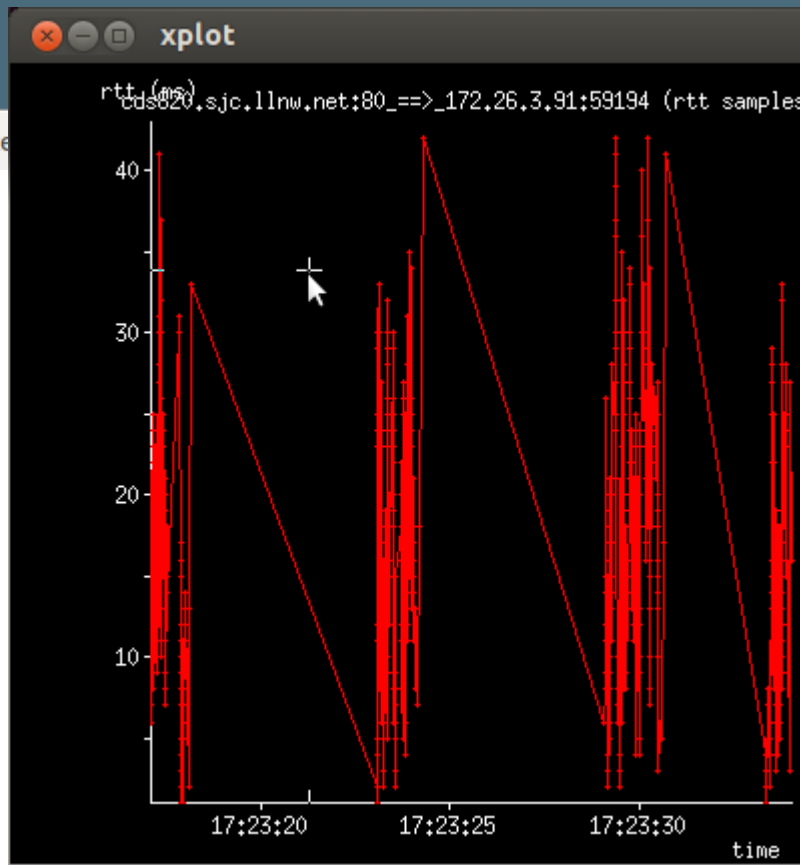
(3 minute window, 3 second interval)

Inbound: 7.55 kbit/s (0.94 kB/s)
Outbound: 4.27 kbit/s (0.53 kB/s)

Average: 677.4 kbit/s (84.68 kB/s)
Average: 18.56 kbit/s (2.32 kB/s)

Peak: 12.51 Mbit/s (1.56 MB/s)
Peak: 412.42 kbit/s (51.55 kB/s)





Bufferbloat.net Resources

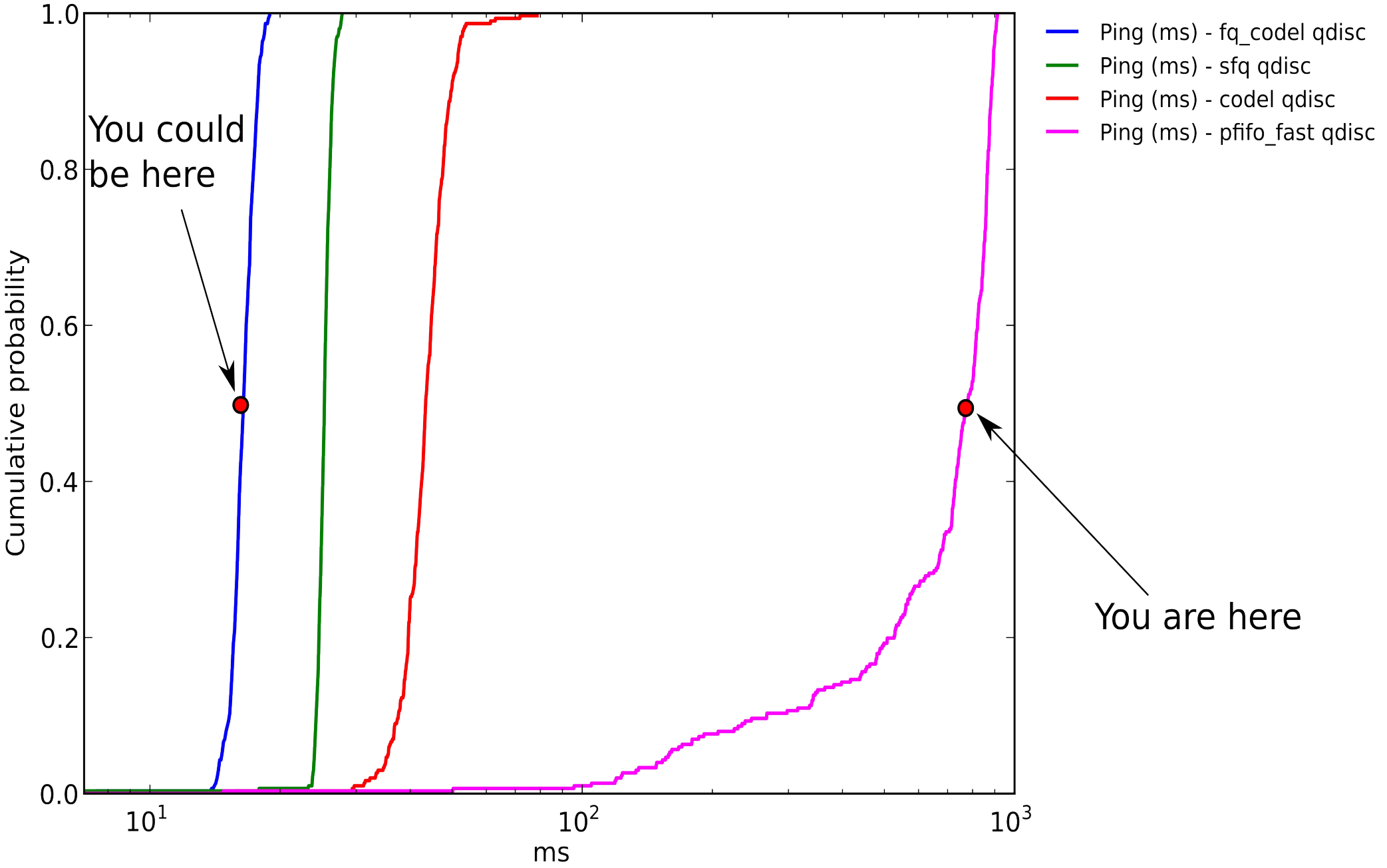
Bufferbloat.net: http://bufferbloat.net
Email Lists: http://lists.bufferbloat.net
IRC Channel: #bufferbloat on chat.freenode.net
CeroWrt: http://www.bufferbloat.net/projects/cerowrt
Other talks: http://mirrors.bufferbloat.net/Talks
Jim Gettys Blog – http://gettys.wordpress.com
Educational Videos:

http://www.bufferbloat.net/projects/cerowrt/wiki/Bloat-videos

Fixing bufferbloat is a volunteer effort!

A big thanks to the 400+ members of the bloat mailing list, Jim Gettys, Kathie Nichols, Van Jacobson, and Eric Dumazet, ISC, ICEI, the CeroWrt contributors, OpenWrt, the Linux core network team, Google, and the Comcast Technology Research and Development Fund.

Questions?


1.0
0.8
0.6
0.4
0.2
0.0
Cumulative probability
You could be here
You are here
10^1
10^2
10^3
ms
Ping (ms) - fq_codel qdisc
Ping (ms) - sfq qdisc
Ping (ms) - codel qdisc
Ping (ms) - pfifo_fast qdisc