Making Wifi **Fast**

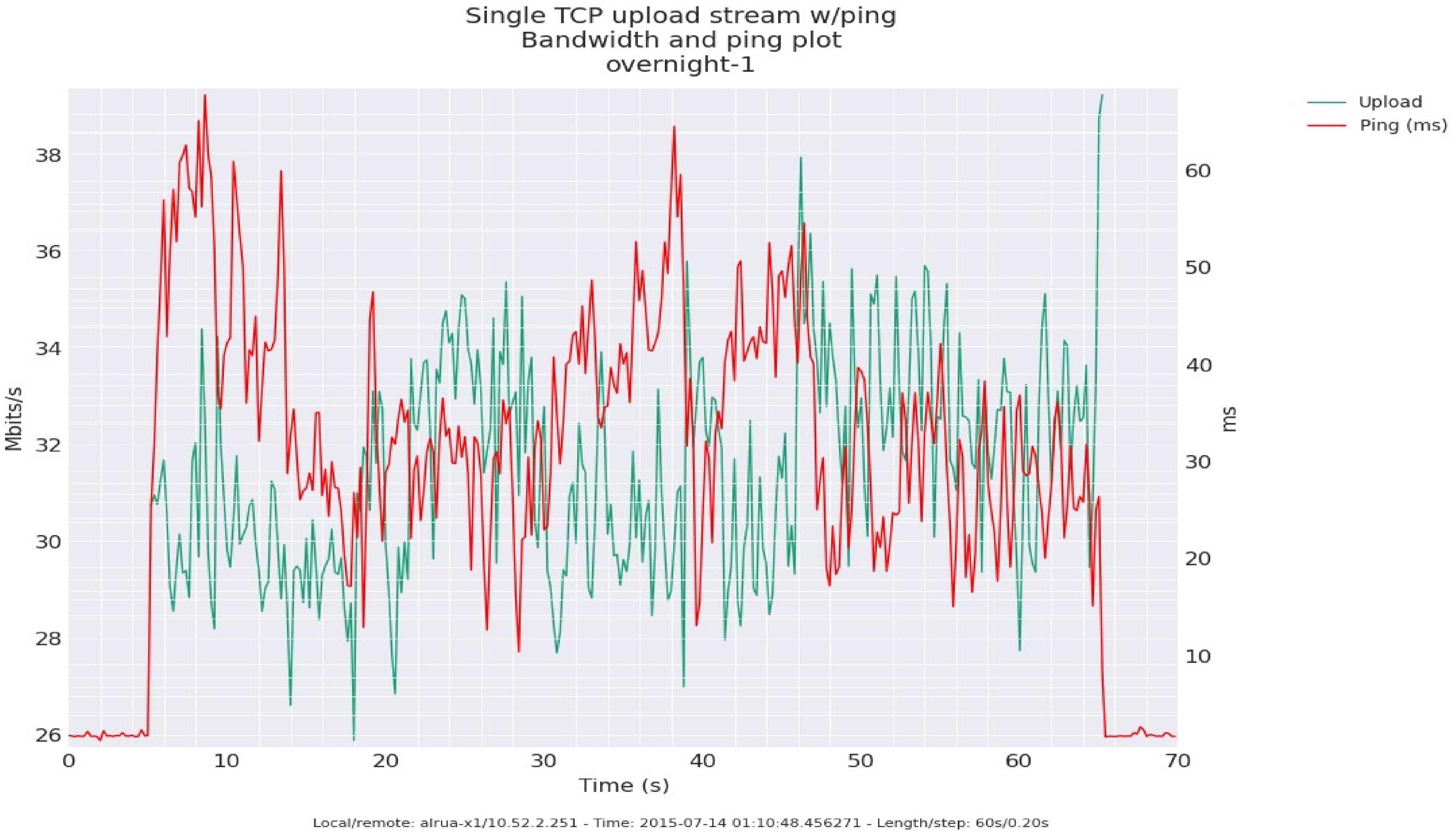
Date: 2015-08-7

Name	Affiliations	Address	Phone	Email
Dave Taht	Annoyer -in-Chief! Bufferbloat.net	2104 W First Street Apt 2002 Ft Myers, FL, 33901		dave.taht@gmail.com

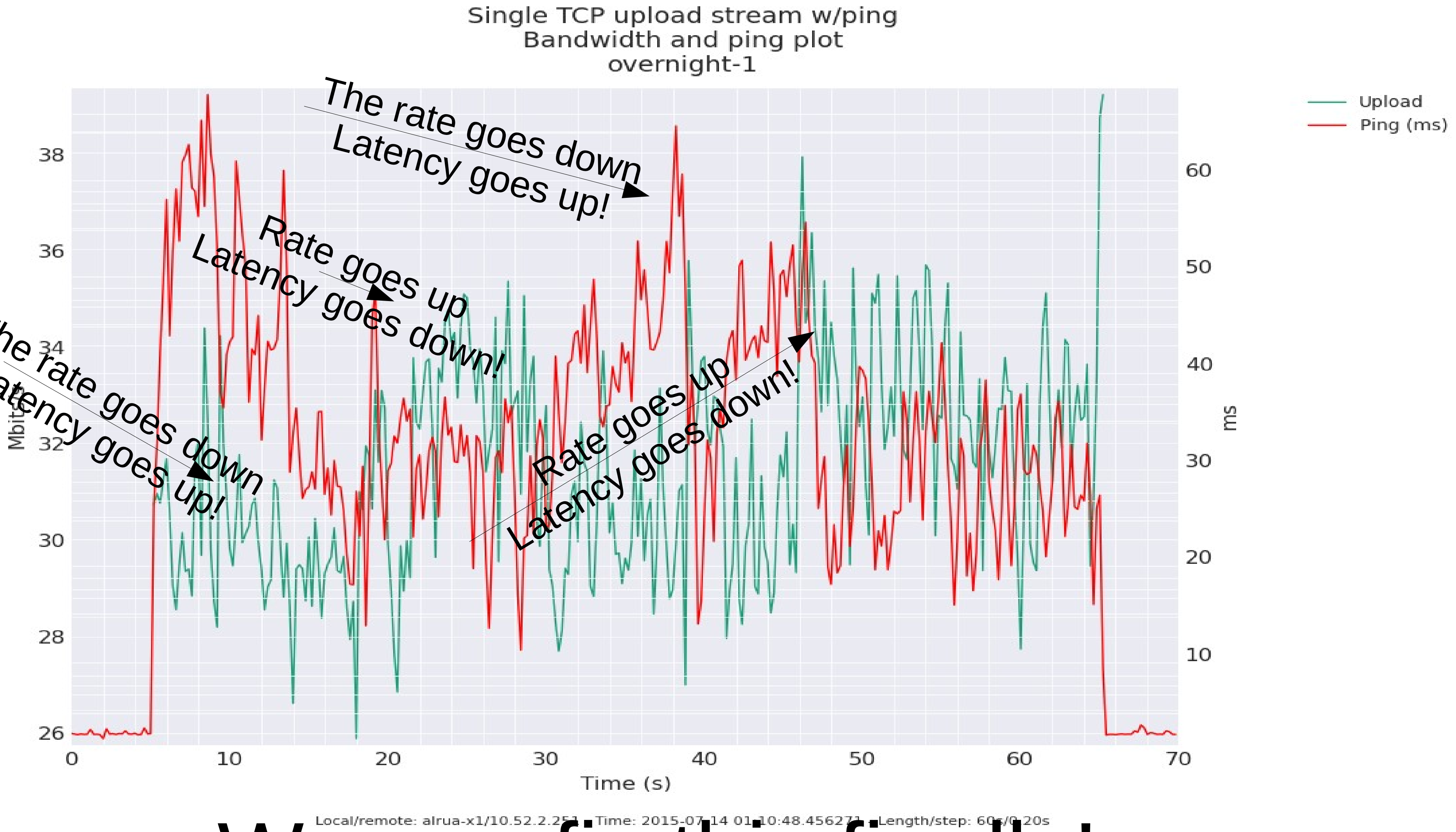
Overview

- On Reducing induced latency on wifi
- How Codel and FQ_Codel work

Today – delay and bandwidth in wifi are inversely correlated



Today – delay and bandwidth in wifi are inversely correlated



The rate goes down Latency goes up!

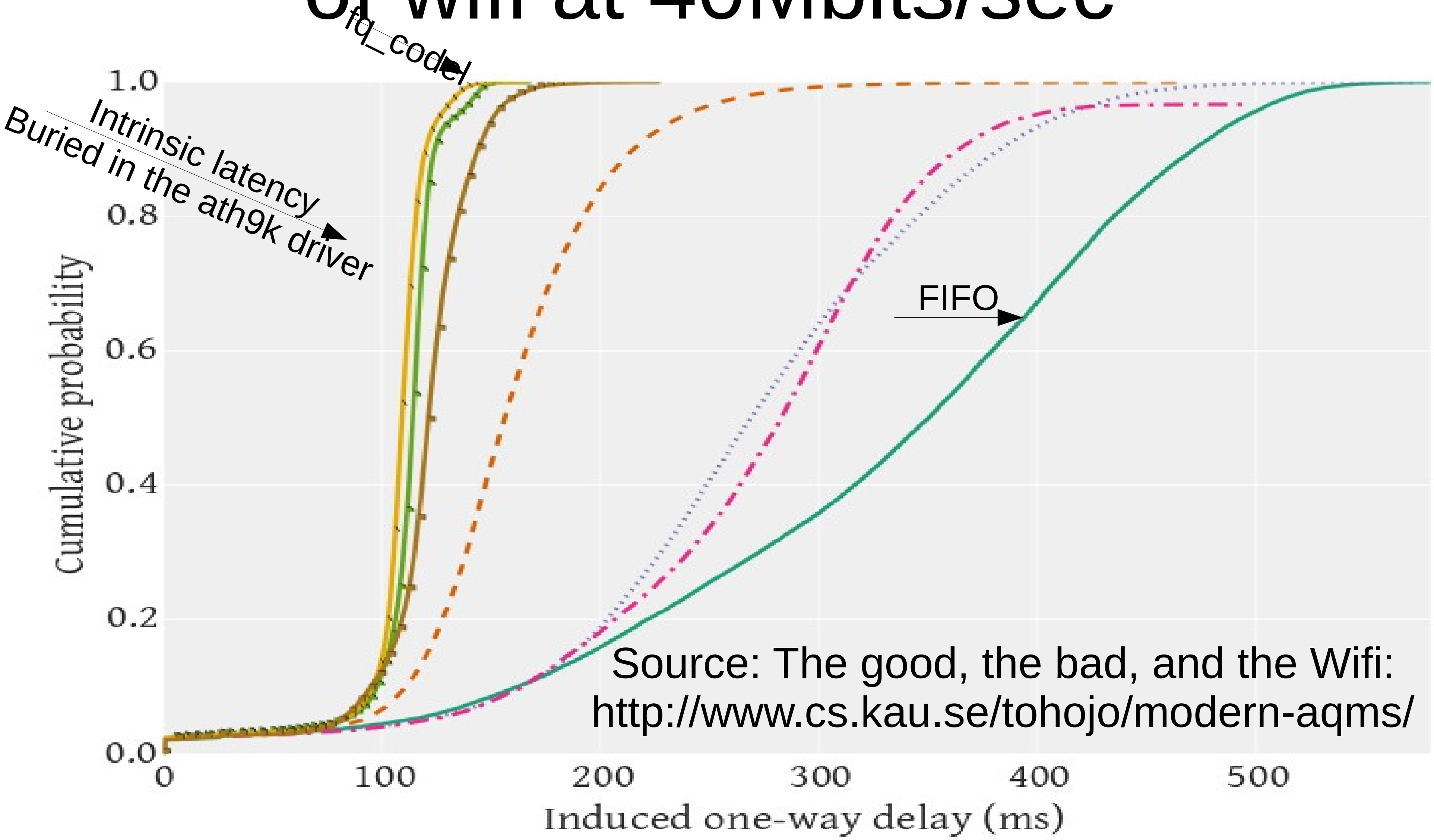
Rate goes up Latency goes down!

The rate goes down Latency goes up!

Rate goes up Latency goes down!

We can fix this finally!

Induced Latency under load of wifi at 40Mbits/sec



Source: The good, the bad, and the Wifi: http://www.cs.kau.se/tohojo/modern-aqms/

Abstract

Reducing Latency and Jitter in Wifi

Bandwidth does not equal “Speed”. Bandwidth = Capacity/interval. Real “speed”, in human terms, is measured by the amount of latency (lag), between an action and a response. In the quest for headline bandwidth in new network standards the industry has lost track of what real speed means.

The presence of large, unmanaged network buffers, primarily across the edge devices of the Internet. In wifi, especially, with wildly variable rates, shedding load to match the available bandwidth, doesn't presently happen, leading to huge delays for much wireless traffic, when under load.

The lag sources in wifi are by no means limited to bufferbloat, but buried deep in stacks that did not successfully absorb wireless-n concepts in the first place.

This talk goes into the problems that the large network queuing delays (bufferbloat) causes. It touches upon some new algorithms, now deployed and in the process of standardization, that produce enormous reductions in latency, and then details some steps the make-wifi-fast project plans to take to address it, on WiFi.

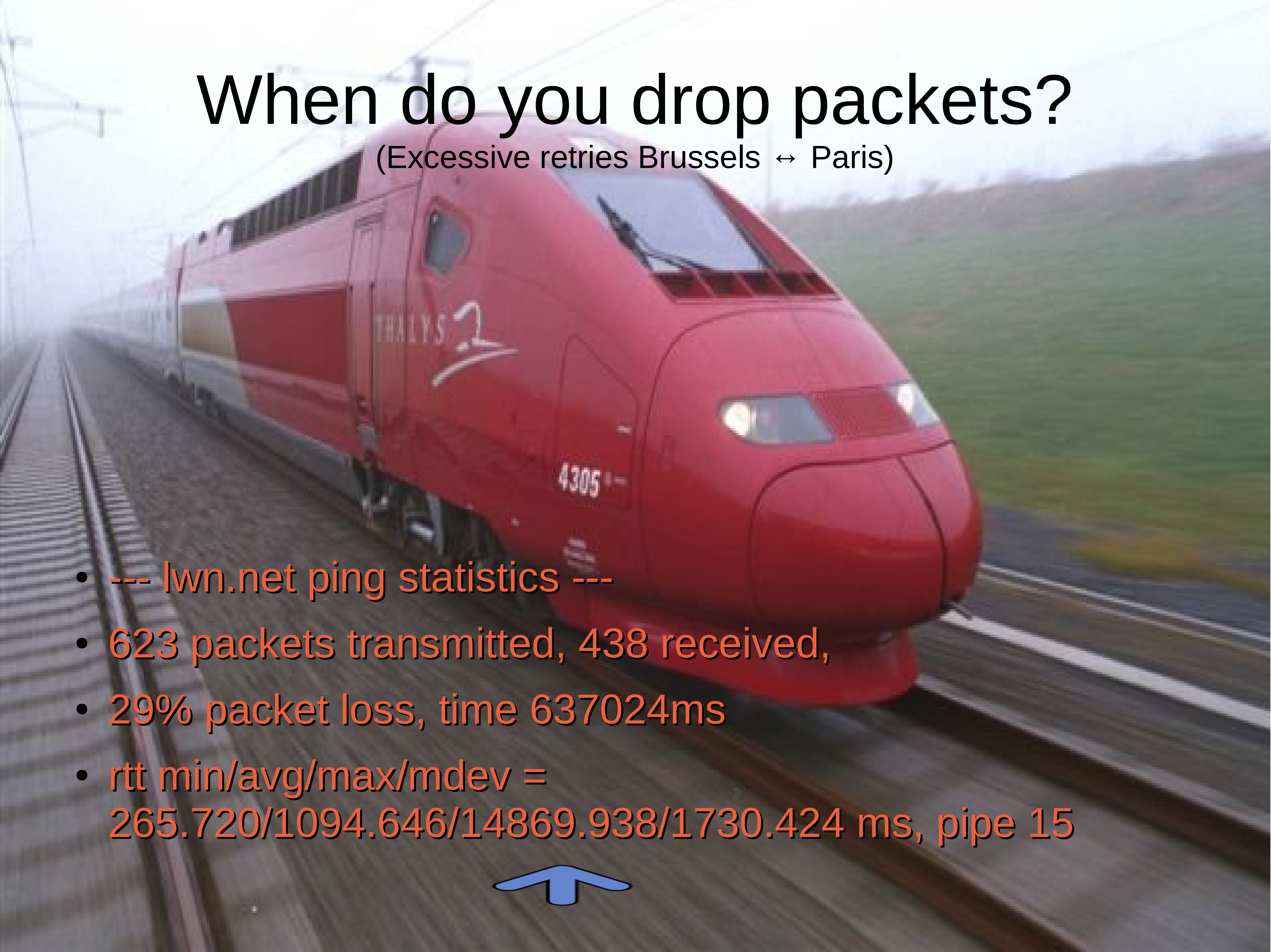
Why am I here?

- IETF is not the place to work on WiFi
- Bufferbloat.net is starting up a new project, “make-wifi-fast” leveraging CeroWrt and OpenWRT – and any other OS-es we can find!

  “The use of open source software to promote broad adoption and use of new technology is now well demonstrated... The CeroWrt/OpenWrt effort could have a similar effect.” – Vint Cerf, “Bufferbloat and other Internet Challenges”
- We fixed ethernet, cable, fiber, and DSL already.
- Experiences thus far with 802.11ac have been *dismal.*
- Perhaps working with everyone that cares about is the right thing to find and fix all the sources of latency and jitter in the Wifi architecture.
- It would be nice to get a grip on what's going on in 802.11ax, ak, etc

When do you drop packets?

(Excessive retries Brussels ↔ Paris)



- --- lwn.net ping statistics ---
- 623 packets transmitted, 438 received,
- 29% packet loss, time 637024ms
- rtt min/avg/max/mdev = 265.720/1094.646/14869.938/1730.424 ms, pipe 15

About Bufferbloat.net

An all-volunteer organization providing wikis, project management, and email lists for those interested in speeding up the internet.

We've:

Gathered together experts to tackle networking queue management and system problem(s), particularly those that affect wireless networks, home gateways, and edge routers.

Spread the word to correct basic assumptions regarding goodput and good buffering on the laptop, home gateway, core routers and servers.

Produced tools to demonstrate and diagnose the problems.

Led a major advance in network Queueing theory

Did and continue to do experiments in advanced congestion management.

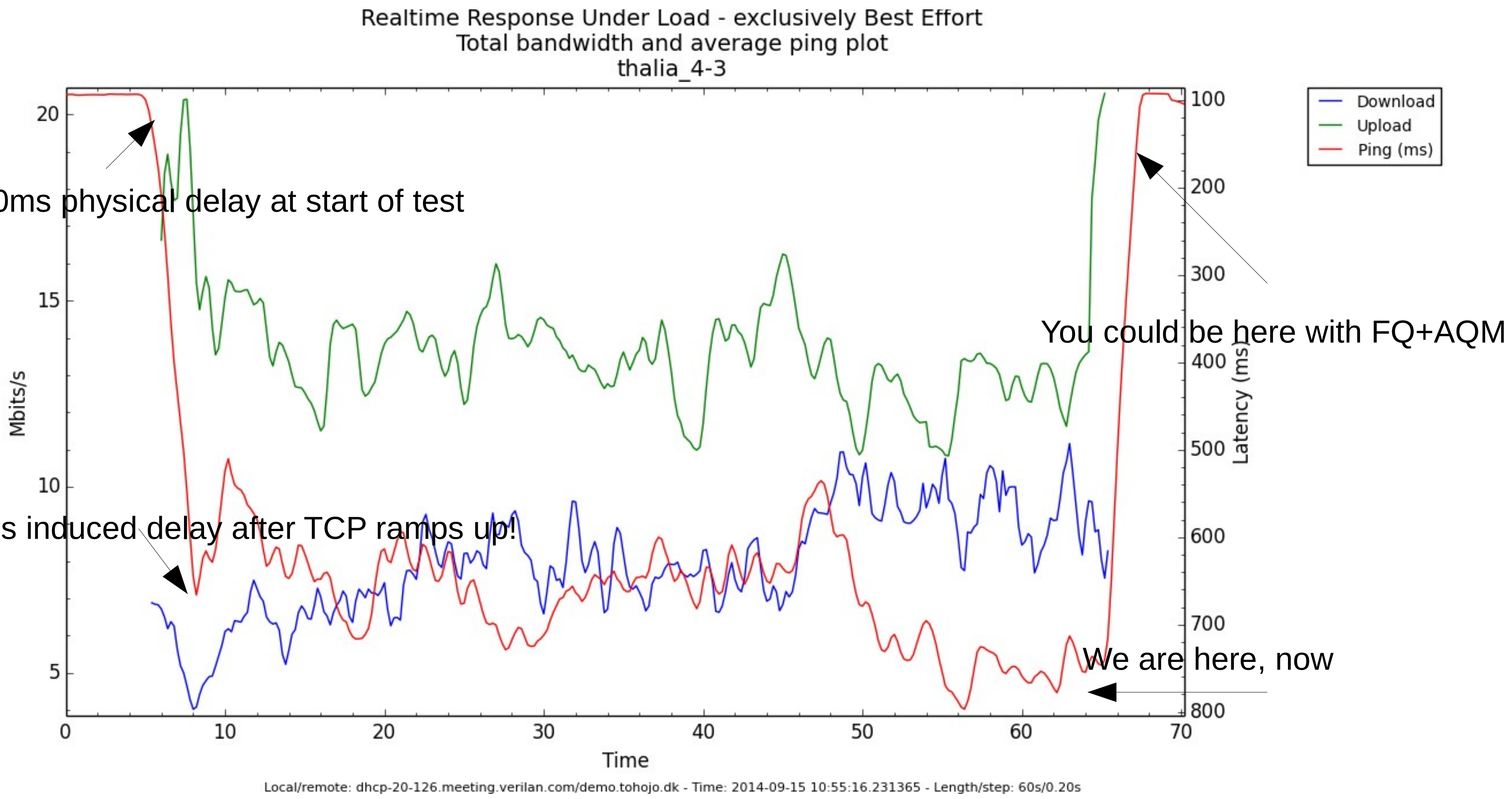
Produced patches to popular operating systems at the device driver, queuing, and TCP/ip layers to dramatically reduce latency for many devices.

Developed reference devices and firmware to push the state of the art forward, for everyone.

Bufferbloat.net Projects

- Bloat – general site for bufferbloat info
- Codel and fq_codel: Algos for shedding load
- CeroWrt – Reference router for the debloaters
- Make-Wifi-Fast – Lowering lag on wifi

Potential Benefits of Fair/Flow Queuing and Active Queue Management on Internet Edge Gateways



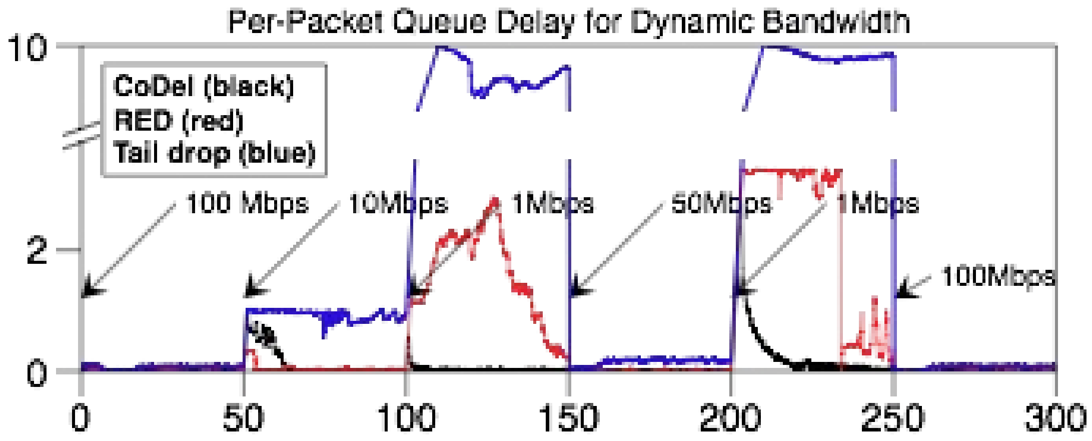
100ms physical delay at start of test

500+ms induced delay after TCP ramps up!

You could be here with FQ+AQM

We are here, now

2012: Codel concept shows promise in adapting to varying link rates on WiFi



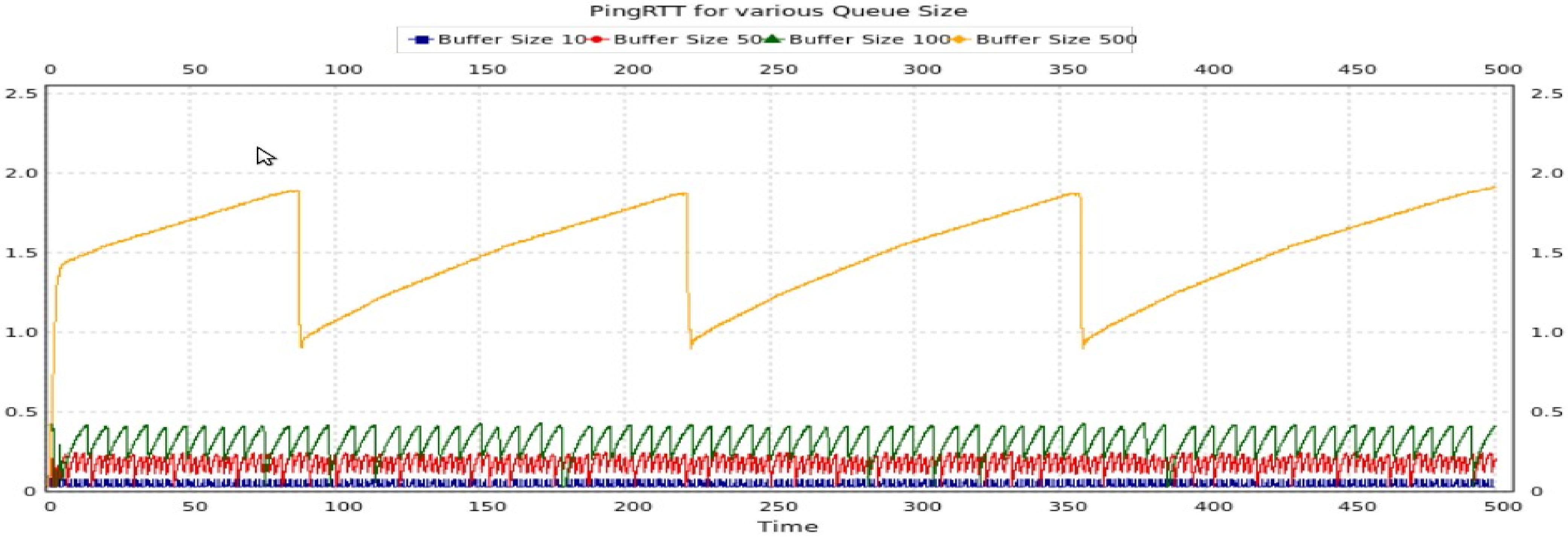
ACM Queue: “Controlling Queue Delay”
– Kathie Nichols and Van Jacobson
http://queue.acm.org/detail.cfm?id=2209336

- See also: “Bufferbloat”
http://queue.acm.org/detail.cfm?id=2071893

- Nominal 100 Mbps link with rate changes, buffer size of 830 packets
- 4 FTPs, 5 packmime connections/sec
- Better than tail drop or RED by a lot.
- fq_codel: *already shown to work decently on wireless p2p links*

Why does Bufferbloat happen?

It comes from TCP's bandwidth probing behavior (TCP 101)



- TCP will always fill the biggest buffer on the path
- As the delays get larger – congestion avoidance mode geometrically gets slower
- With CUBIC, the sawtooth looks more like an S-curve

http://staff.science.uva.nl/~delaat/netbuf/bufferbloat_BG-DD.pdf (.5Mbit uplink)

And dramatic overbuffering, everywhere

- Unmanaged buffers in network stacks
  - Sized for the maximum bandwidth the device can sustain
  - Not managed for the actual bandwidths achieved
  - Often behind proprietary firmware where it can't be fixed.
  - With things like packet aggregation providing illusory gains on simple minded benchmarks but not real traffic

Consequences of TCP's design

A single connection will fill any size buffer put in front of it at the path's bottleneck, given time: adds one packet/ack to the buffer

Timely dropping or marking of packets is necessary for correct operation of TCP on a saturated link.

Even IW4 is an issue at low bandwidths (e.g. VOIP over busy 802.11 network): do the math. Just "fixing" TCP does not fix the network.

Smarter queuing is essential. Congestion window is not shared among connections (currently). Current web server/client behavior means head of line blocking causes bad transients.

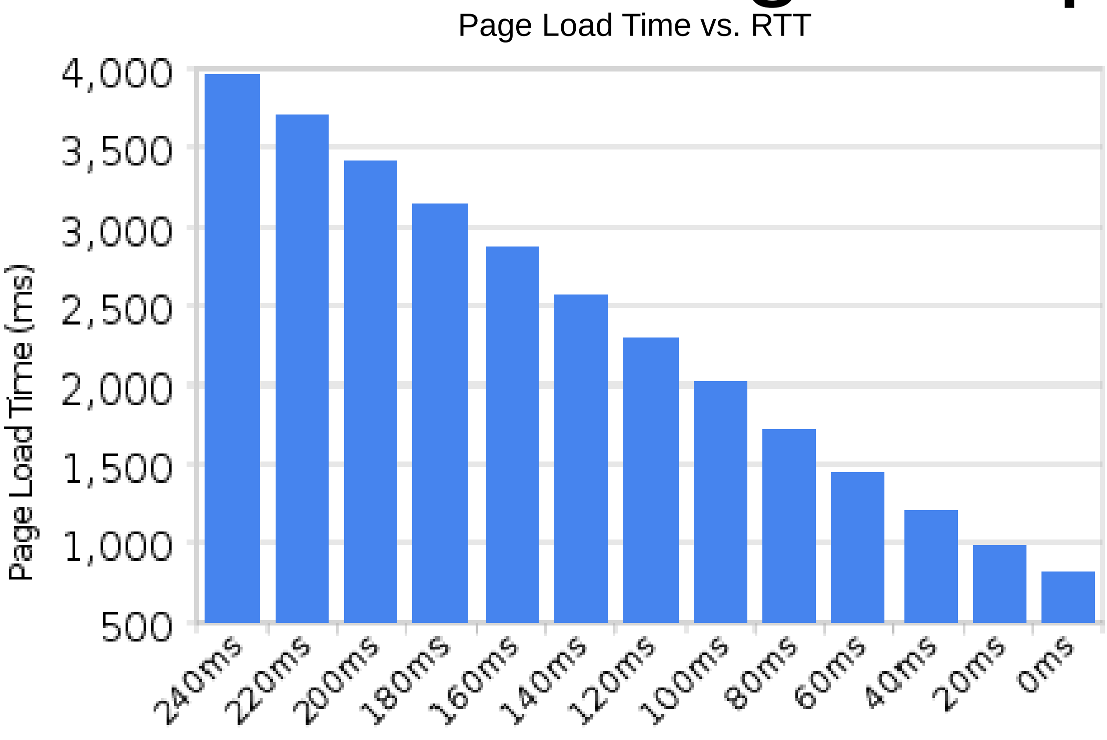
Sharing responsiveness is quadratic in delay. Elephant flows become mammoth flows in the face of overbuffering.

In a Web TCP transaction

SYN → SYN/ACK 1 RTT
SSL NEGOTIATION 2 RTTs
DATA REQUEST/Transfer – 1 RTT 10 packets
(90% of all web transactions are one IW10 burst)
FIN – FIN/ACK 1 RTT
CLOSE 1 packet

Only 1 TXOP in 10 can aggregate!
Excessive base RTT adds up!

Web Browsing is dependent on **RTT**



Page Load Time vs. BW

Effective HTTP Throughput

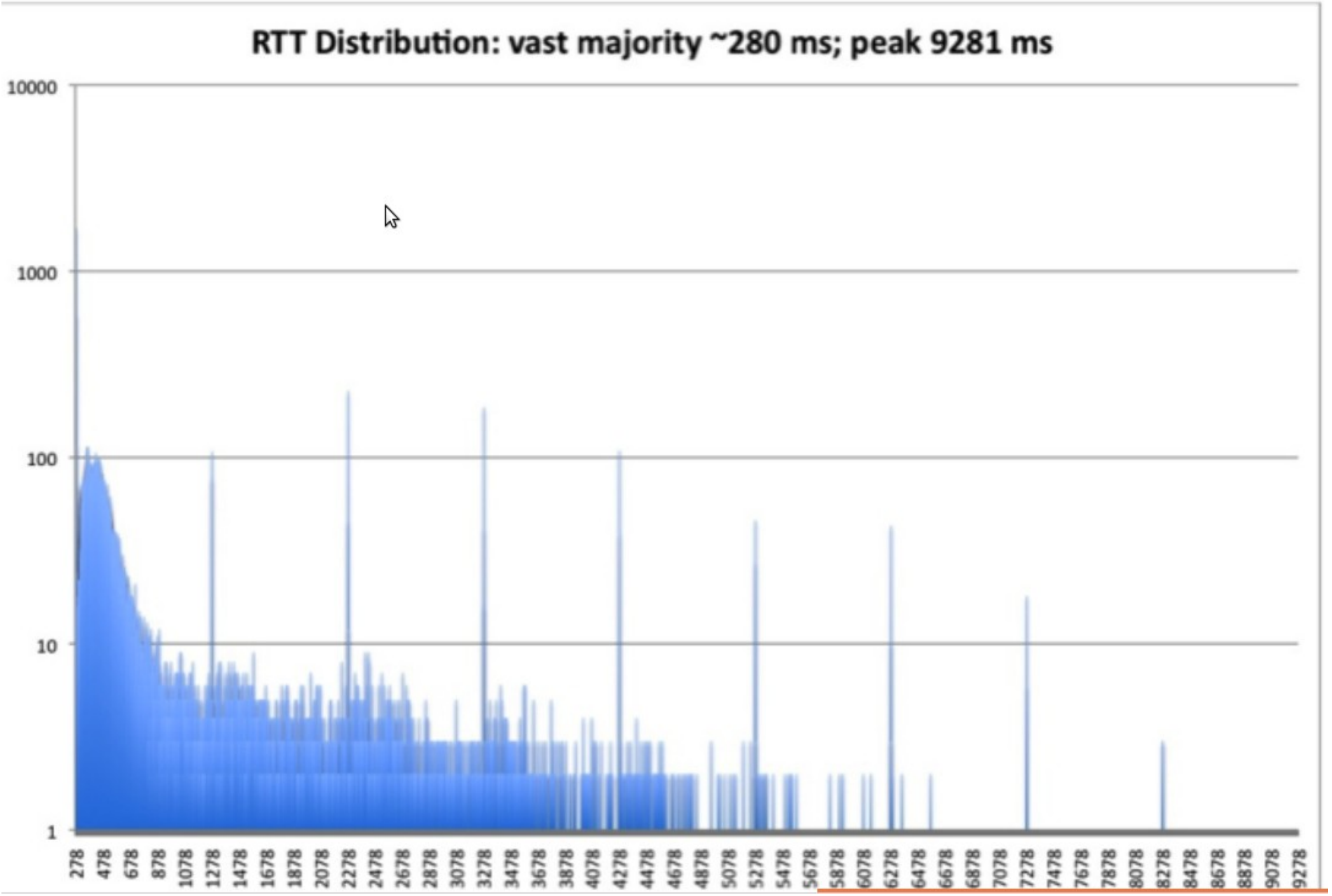
- Page Load Time is sensitive to *round-trip* latency
  - Google data shows 14x multiplier
    - +200ms RTT = +2.8 seconds PLT

- Diminishing returns from increased data rate
  - Page Load Time at 10 Mbps almost indistinguishable from 6 Mbps

Source: SPDYEssentials, Roberto Peon & William Chan, Google Tech Talk, 12/8/11

Gaming, DNS, and VOIP traffic are even more sensitive to RTT!

Too much delay and you get layer 3 adding even more packets...

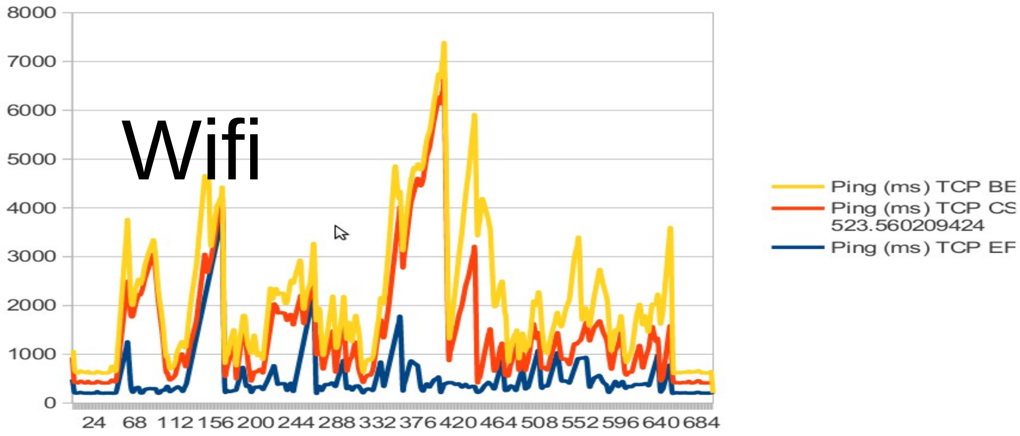


The good news

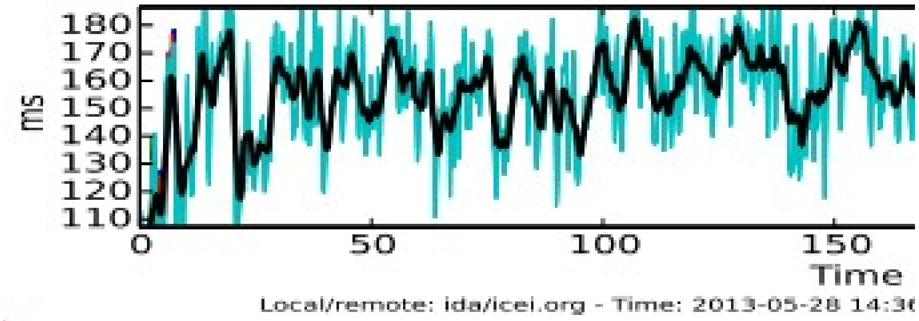
- Bufferbloat is basically fixed on ethernet, cable, dsl, and fiber, with the new algorithms (codel, fq_codel, pie) ,and improvements in the Linux network stacks,
  - 2-3 orders of magnitude reductions in network latency being seen AND improvements in goodput
- Algos are now deployed in several products, and in nearly every third party router firmware.
- Two algorithms are patent free and open sourced code widely available for them
- Standardization activities in the IETF

Typical Latency with Load Today

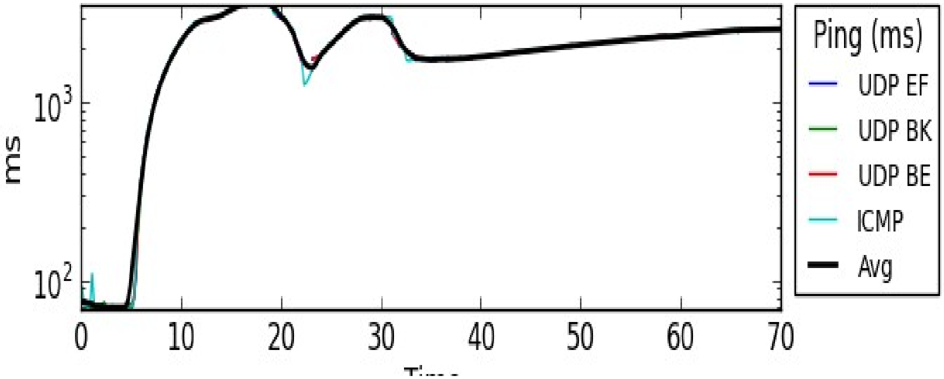
Wifi



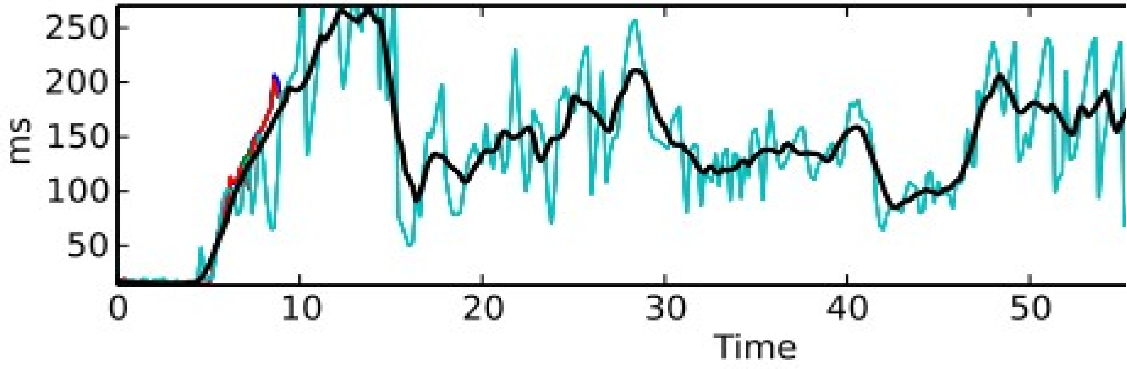
Fiber



ADSL

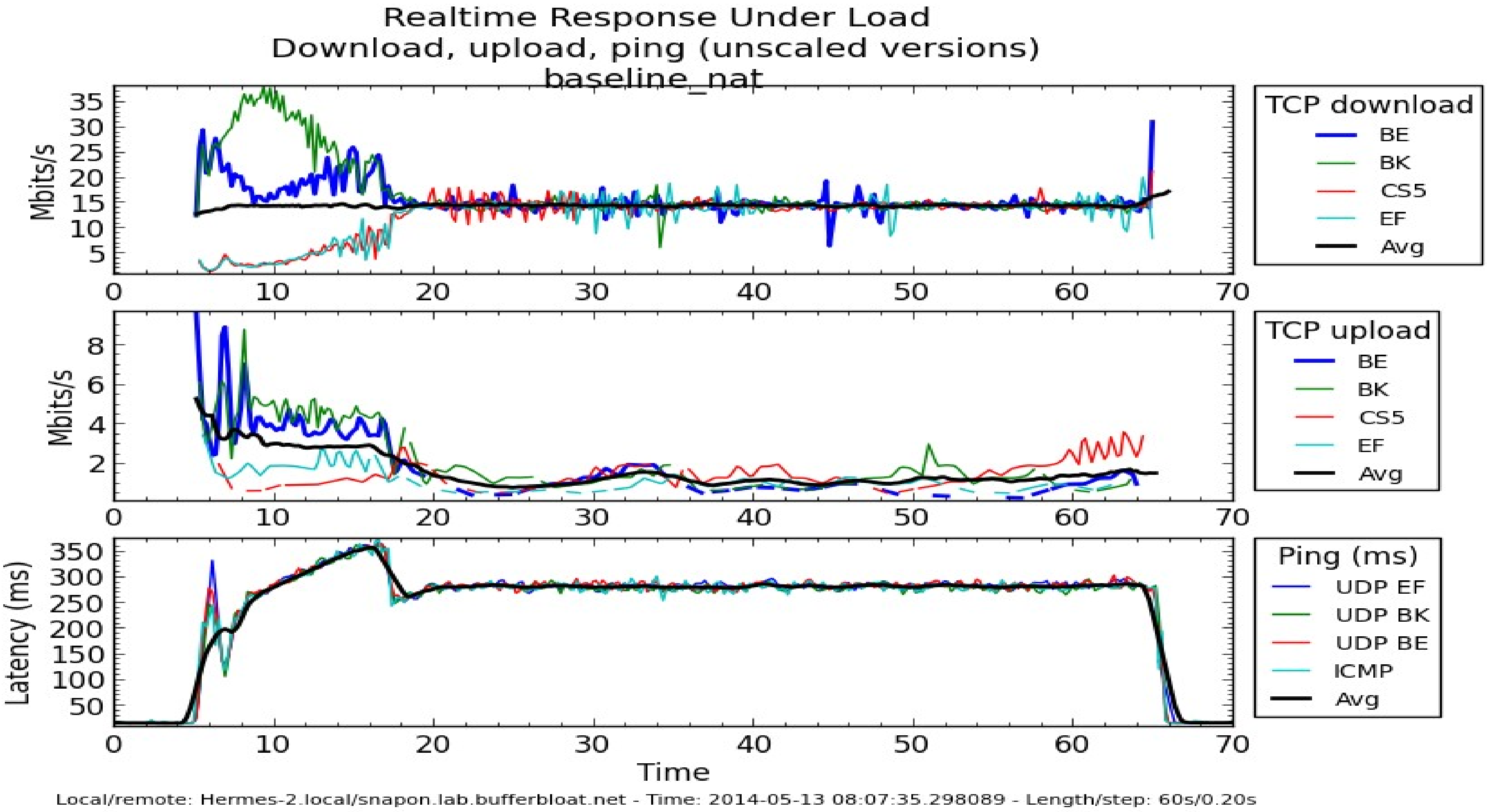


Cable (DOCSIS 2.0)

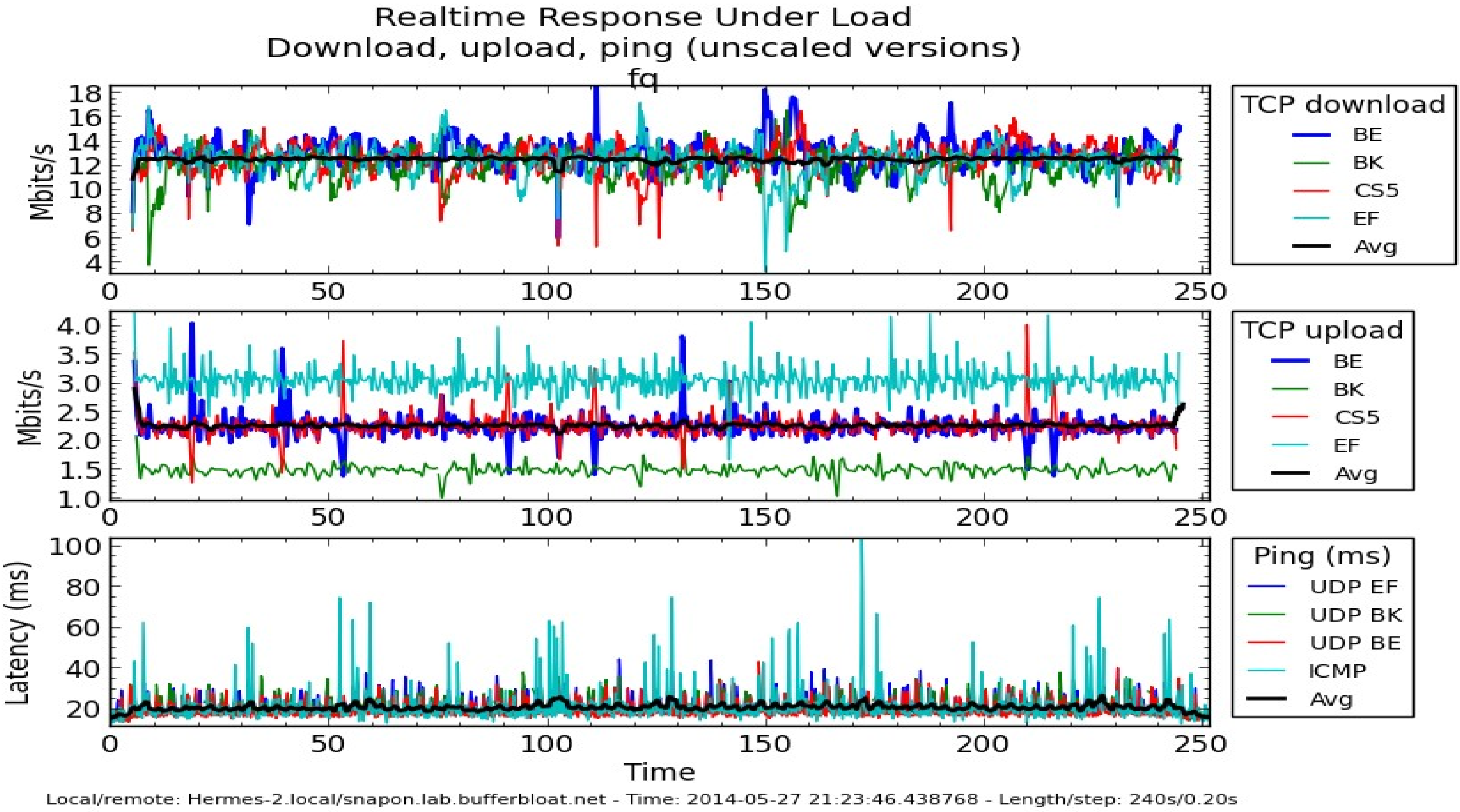


Much of this latency comes from Queue Delay (bufferbloat)

Ex: 300ms excess latency on cable

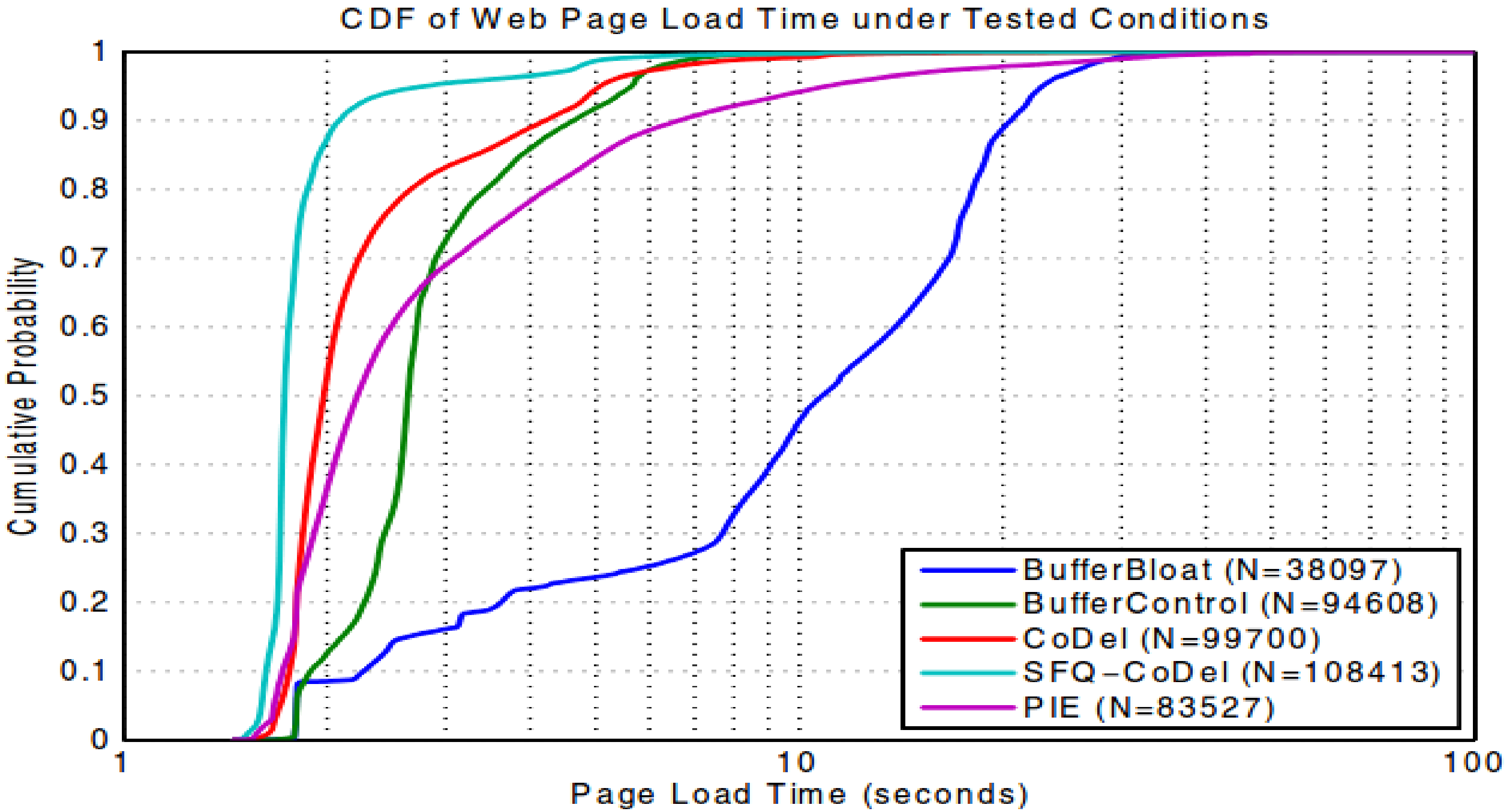


Cut to ~10ms with fq_codel



Source:
http://burntchrome.blogspot.gr/2014_05_01_archive.htm
l

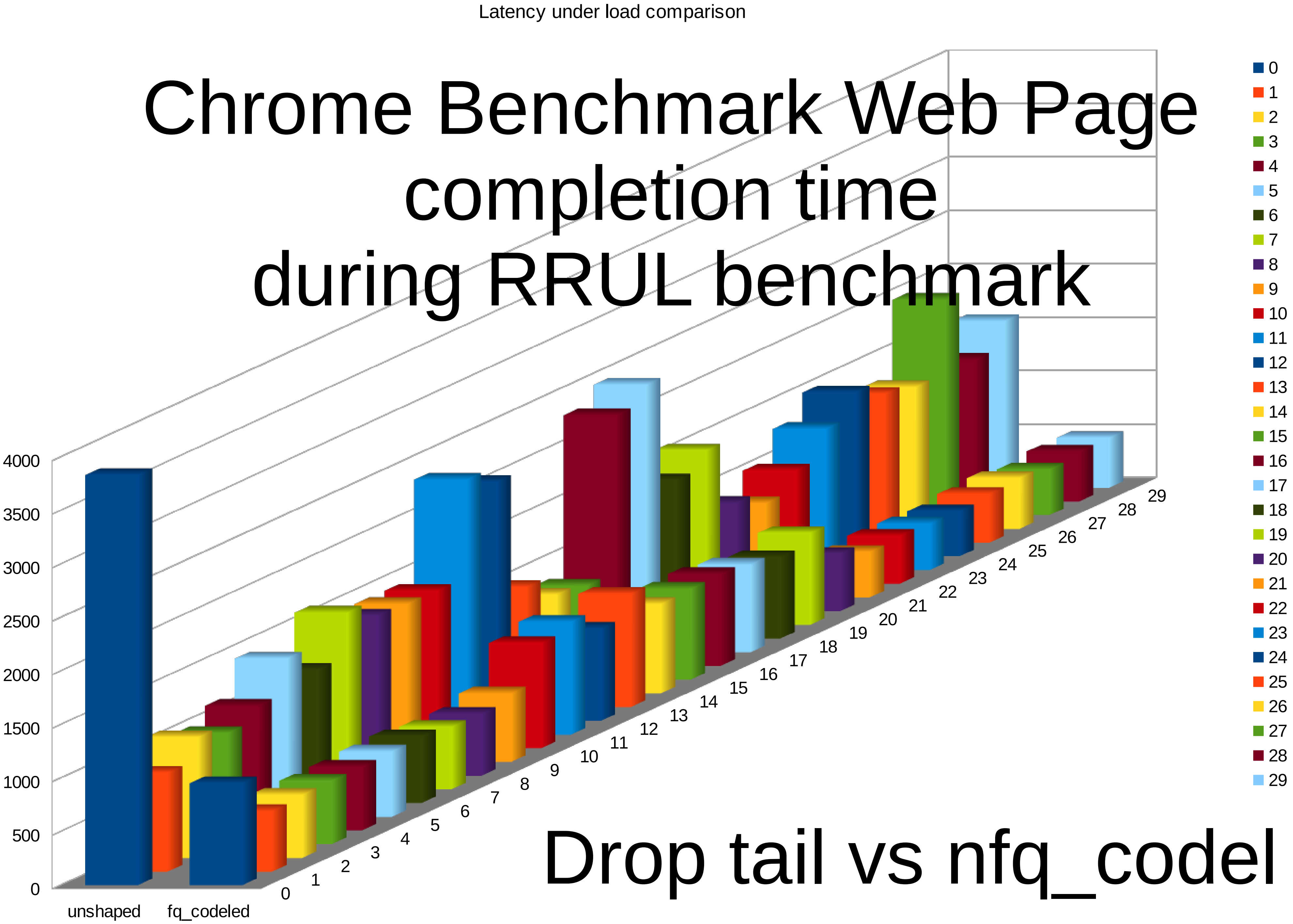
Web page load time wins



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

From: http://www.cablelabs.com/wp-content/uploads/2013/11/Active_Queue_Management_Algorithms_DOCSIS_3_0.pdf

Chrome Benchmark Web Page completion time during RRUL benchmark



Drop tail vs nfq_codel

Linux TCP/AQM/FQ Advances: 2010-2014

- Linux 3.0: Proportional Rate Reduction
- Linux 3.3: Byte Queue Limits
- Linux 3.4 RED bug fixes & IW10 added & SFQRED
- Linux 3.5 Fair/Flow Queuing packet scheduling (fq_codel, codel)
- Linux 3.6 Stability improvements to fq_codel
- Linux 3.7 TCP small queues (TSQ)
- Linux 3.8 HTB breakage
- Linux 3.11 HTB fixed
- Linux 3.12 TSO/GSO improvements
- Linux 3.13 Host FQ + Pacing
- Linux 3.14 Pie added
- Linux 3.15 Change to microseconds from milliseconds throughout networking kernel
- Linux 3.17? Network Batching API
- The Linux stack is now mostly “pull through”, where it used to be “push”, and looks nothing like it did 4 years ago.
- fq_codel now std on openwrt, fedora, arch (anything with systemd), many others

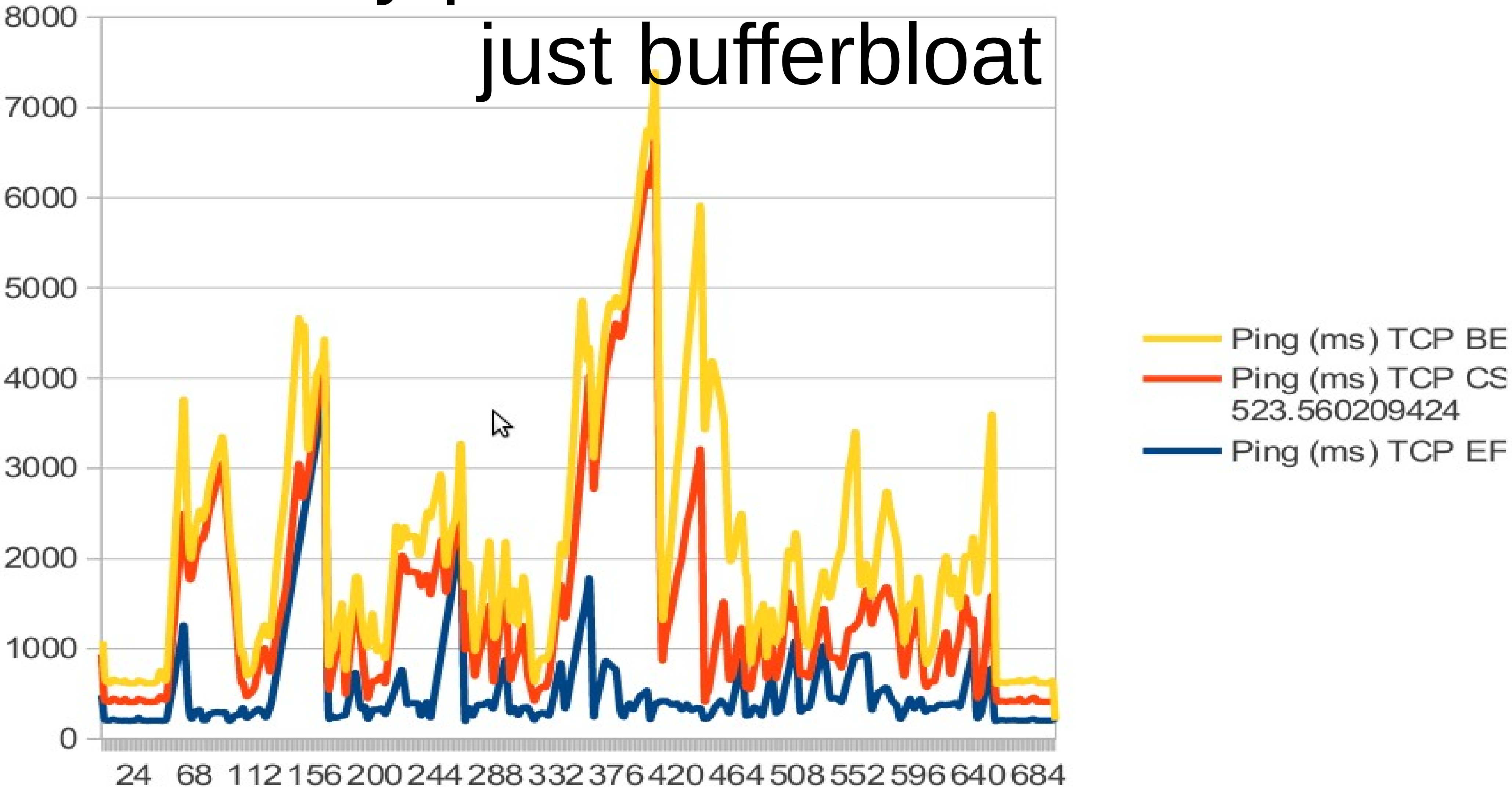
But: not much progress on wifi

IETF AQM Working Group Status

- AQM working group

  https://datatracker.ietf.org/doc/charter-ietf-aqm/
- Pending drafts:
  - http://snapon.lab.bufferbloat.net/~d/draft-taht-home-gateway-best-practices-00.html
  - http://tools.ietf.org/html/draft-white-aqm-docsis-pie-00
  - http://tools.ietf.org/html/rfc2309 Is beiing revised
  - http://tools.ietf.org/html/draft-ietf-aqm-recommendation-03
  - http://tools.ietf.org/id/draft-kuhn-aqm-eval-guidelines-00.txt
  - http://tools.ietf.org/html/draft-hoeiland-joergensen-aqm-fq-codel-00
  - http://tools.ietf.org/html/draft-nichols-tsvwg-codel-02
  - http://sandbox.ietf.org/doc/draft-baker-aqm-sfq-implementation/

The bad news: Latency problems on WiFi are not just bufferbloat



How to make WiFi truly faster on stations and Access points?

- With new AQM and packet scheduling algorithms?
- With Packet aggregation?
- With EDCA scheduling?
- With 802.11e prioritization?
- Increasing numbers of stations and contending access points?
- Excessive low rate multicasts and retransmissions coming from mdns and ipv6?
- With upcoming standards like 802.11ax?

? ?

- New project: MAKE-WIFI-FAST

https://lists.bufferbloat.net/listinfo/make-wifi-fast/

Linux WiFi stack – Planned fixes in make-wifi-fast

- Better Benchmarks and Tools
- Improve the minstrel rate selection algorithm
  - Smarter Math
  - Better aggregation awareness
  - W/Power aware scheduling (minstrel-ht-blues) – Now kernel mainlined!
  - Reducing retransmits & selective retransmit
- Rework the stack
  - Per station queueing
  - Single queue promotion to 802.11e
  - MU-MIMO support
  - Obsolete VO Queue
- Add rate control aware AQM/Fair Queuing Algorithms
  - Codel, Pie, fq_pie
  - Sort on dequeue
- Take advantage of new statistics and possibilities in new 802.11 standards
  - And in the end, reduce induced latency by at least a factor of 10, and the carrying capabilities of the 802.11n MAC by at least a factor of five.

Some useful tools for looking at the Bufferbloat Problem

- Tcptrace and xplot.org and isochronous burst tests
- flent (by Toke Hoeiland-Joergensen)
  - Standardized data format, 30+ network specific tests testing for latency under load, 20+ plot types, extensive support for batching and other automation, usage of alternate TCP algorithms, in combination with other web and voip-like traffic.
- Linux Kernel mainline, Codel, fq_codel and pie in most distributions now (Redhat 7, ubuntu, debian, etc) – Cake in progress
- Cerowrt is dead, long live Openwrt!
  - Was: Inexpensive actual router configurable with nearly every AQM and FQ algorithm using “SQM”
  - Emulations of common CMTS and DSLAM behavior
  - Results proven sane to 50mbits
  - Most of cerowrt is already in openwrt “Barrier Breaker”.
- SQM (“Smart Queue Management”)
  - Drop in replacement for wondershaper portable to all modern linuxes
  - Uses fq_codel by default, pie, sfq, codel optionally
- Openwrt “Barrier Breaker”, ipfire, pfsense, other third party router firmwares have support also
- NS2 models of codel, sfq_codel, and pie in ns2 mainline
- NS3 models of codel, fq_codel, asymmetric edge networks, etc,
  - From Google 2014 summer of code
  - Codel in in September ns-3.21, sfq_codel in ns-3.22, December, 2014
  - Wifi mac support, LTE, etc

All open source

Planned Rate selection Improvements

- Minstrel-2 rate selection algorithm improvements

The minstrel rate selection algorithm was originally developed against wireless-g technologies in an era (2006) when competing access points were far less prevalent. While updated significantly for wireless-n a thorough analysis has not been performed in the wide variety of rates and modern conditions. Also, some new mathematical techniques have been developed since 2009 that might make for better rate control overall. A new ns3 model will be developed to mirror these potential changes and a sample implementation produced for the ath9k chipset (at minimum). Minstrel will do a much better job on aggregation and in MU-MIMO conditions.

- Power aware rate control and scheduling

It may be possible to do transmits at "just the right power" for the receiving station. Minstrel-Blues is now mainlined!

Full description: http://d-nb.info/106738460X/34

- Reducing retransmits

Retransmit attempts will move from counter based to a time and other workload based scheduler. This will help keep bad stations from overwhelming the good, and reduce latencies overall. Losing more packets is fine in the pursuit of lower latency for all.

- Selective (re)transmit

Currently all Linux wifi drivers are dumb when it comes to retransmitting portions of an aggregate that fail, attempting to perfectly transmit the entire aggregate. In the general case, not all packets need to be retransmitted - examples include all but the final tcp ack in a flow, all but the last voip packet in a flow, and so on.

Per Station Queues is NEXT!

- Single queue promotion to 802.11e aggregates
- Per station queueing

The current structure of the linux wifi stack exposes only the 802.11e wifi queues, not multicast, and not the queues needed for multiple stations to be sanely supported. Repeated tests of the 802.11e mechanism shows it to be poorly suited for a packet aggregation world. By reducing the exposed QoS queue to one, we can instead expose a per-station queue (including a multicast queue) and manage each TXOP far more sanely.

There are a few other options as to what layer this sort of rework goes into. Given the current structure of the mac80211 stack, it may be that all this work (exposure of the station id), has to take place at that layer, rather than the higher level qdisc layer. Basic support for this (no code!) landed in the linux kernel in March, 2015

- MU-MIMO support

Nearly all of the changes above have potentally great benefit in a MU-MIMO world, and are in fact, needed in that world. Regrettably none of the major chipset makers nor router makers seem to be co-ordinating on a standard api structure for doing this right, and it is hoped that by finding and targetting at least one MU MIMO chipset that progress will be made.

Adding AQM and Fair Queuing

- Codel
  - While codel appears to be a great start in managing overall queue length, it is apparent that modifications are needed to manage txops rather than packets, and the parking lot half duplex topology in wifi leads to having to manage the target parameter (at least) as a function of the number of active stations, and closer integration into minstrel for predictive scheduling seems needed also.
- fq_codel
  - A perhaps saner approach than a stochastic hash is merely to attempt to better "pack" aggregates with different flows whenever possible, taking into account loss patterns, etc. Using a deterministic per-station hash and setting aside 42 buckets for each station is not a lot of overhead.
- Cake's algorithms
  - Improvements across the board in memory use, cpu utilization, and efficiency – but they are presently at the wrong layer of the wifi stack!

Sort on dequeue

An aggregate of packets arrives and is decoded all at once, and then delivered in FIFO order at a high rate (memory speeds) to another device, usually ethernet. However that high rate is often still too slow for a fq_codel qdisc attached to that ethernet device to actually do any good, so it would be better to sort on the dequeue (of up to 42 packets), then deliver them to the next device.

We believe that if the delivery is sorted (fair/flow queued), that more important packets will arrive first elsewhere and achieve better flow balance for multiple applications.

Multiple chipsets deal with packet aggregation in different ways, as does firmware - some can't decode any but the entire aggregate when encrypted, for example, they arrive as a binary blob, and there are numerous other chipset and stack specific problems.

Remove Reorder Buffers

- The linux tcp/ip stack can handle megabytes of packets delivered out of order. So can OSX. Windows can't. We don't care.
- In the quest for low latency, a few out of order packets shouldn't matter.
- When we can identify flows, clearly, we can do a better job...

Station improvements?

While many of the above improvements also apply to stations, the benefits are more limited. The overall approach should be to do better mixing and scheduling of the aggregates that a station generates, and to hold the queue size below 2 full aggregates whenever possible. Further improvements in station behavior include predictive codel-ing for measuring the how and when EDCA scheduling opportunities are occurring, and so on.

The primary focus is on APs, since that's where most of the problems are, today.

Some details on CoDel & Fq_CoDel

Various talks:

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

ACM Queue

http://queue.acm.org/detail.cfm?id=2209336

Internet Drafts:

https://datatracker.ietf.org/doc/draft-nichols-tsvwg-codel/

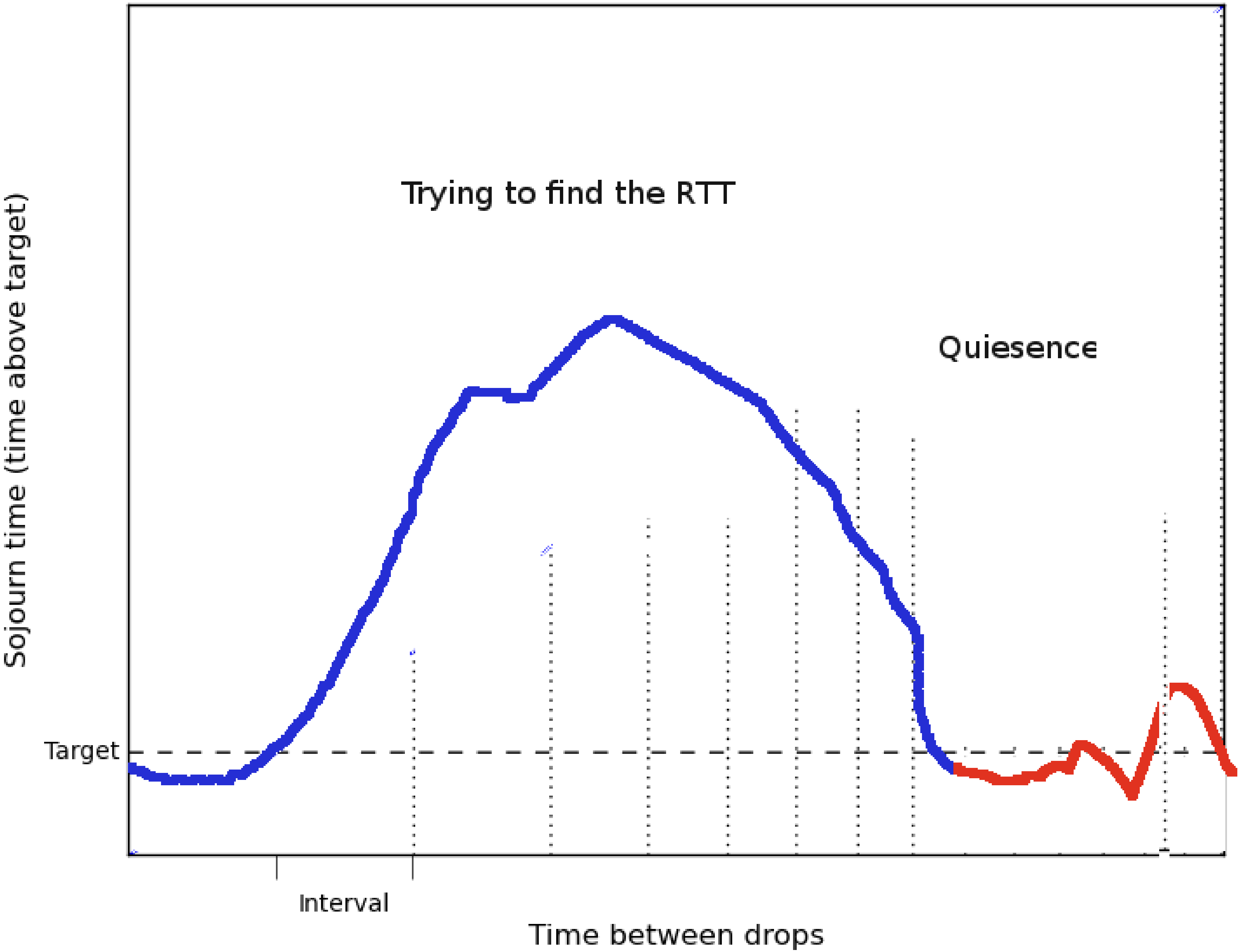
http://tools.ietf.org/html/draft-hoeiland-joergensen-aqm-fq-codel-00

Source Code in Linux 3.6 and later

Introducing Kathie Nichols and Van Jacobson's Codel algorithm

- Measure the latency in the queue, from ingress to egress, via timestamping on entry and checking the timestamp on exit.
- When latency exceeds target, think about dropping a packet
- After latency exceeds target for an interval, drop a packet at the HEAD of the queue (not the tail!)
- If that doesn't fix it, after a shorter interval (inverse sqrt), drop the next packet sooner, again, at the HEAD.
- Keep decreasing the interval between drops per the control law until the latency in the queue drops below target. Then stop. Save the value and increase it while no drops are needed.
- We start with 100ms as the interval for the estimate, and 5ms as the target. This is good on the world wide internet
- Data centers need much smaller values.

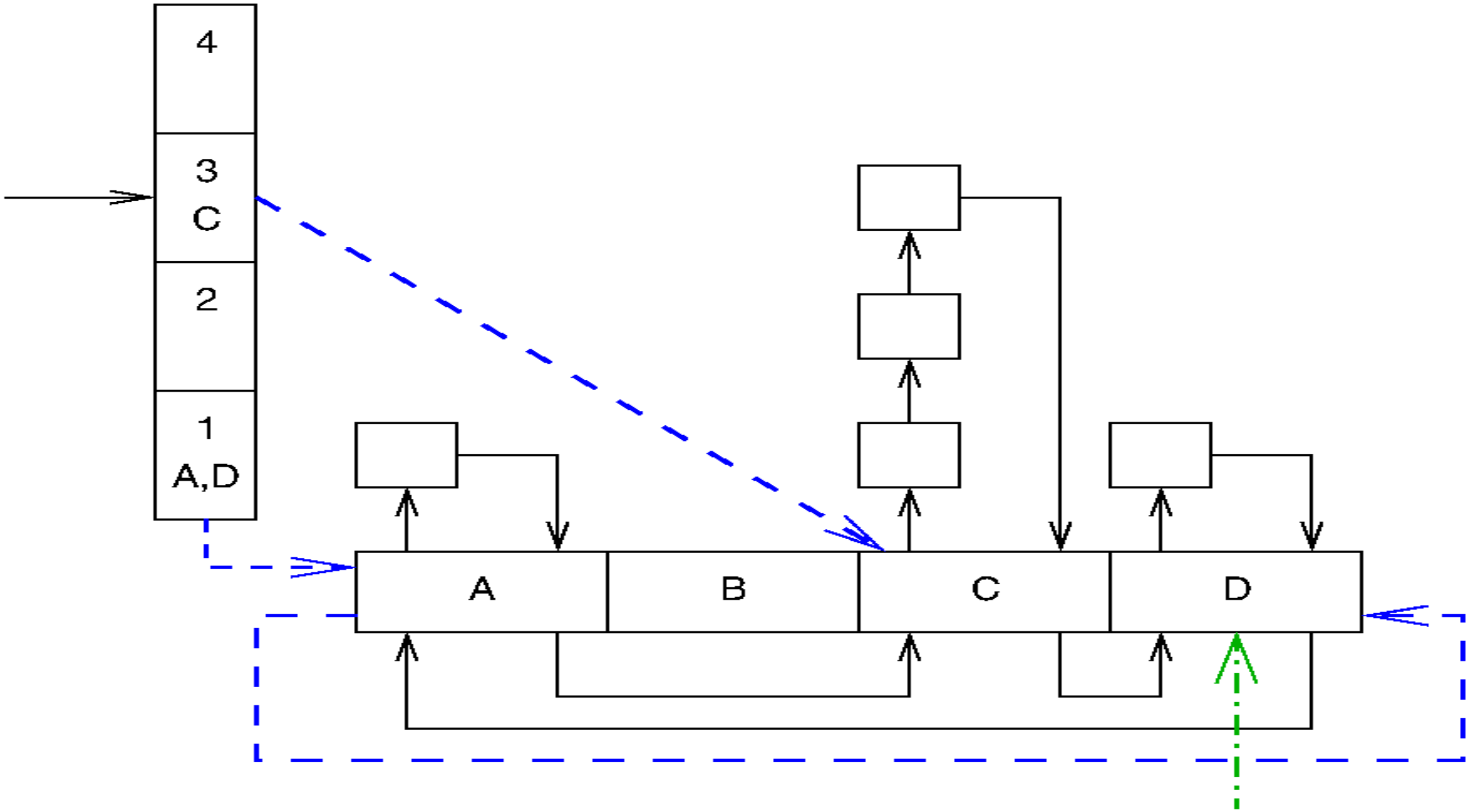
CoDel Drop Scheduler Behavior



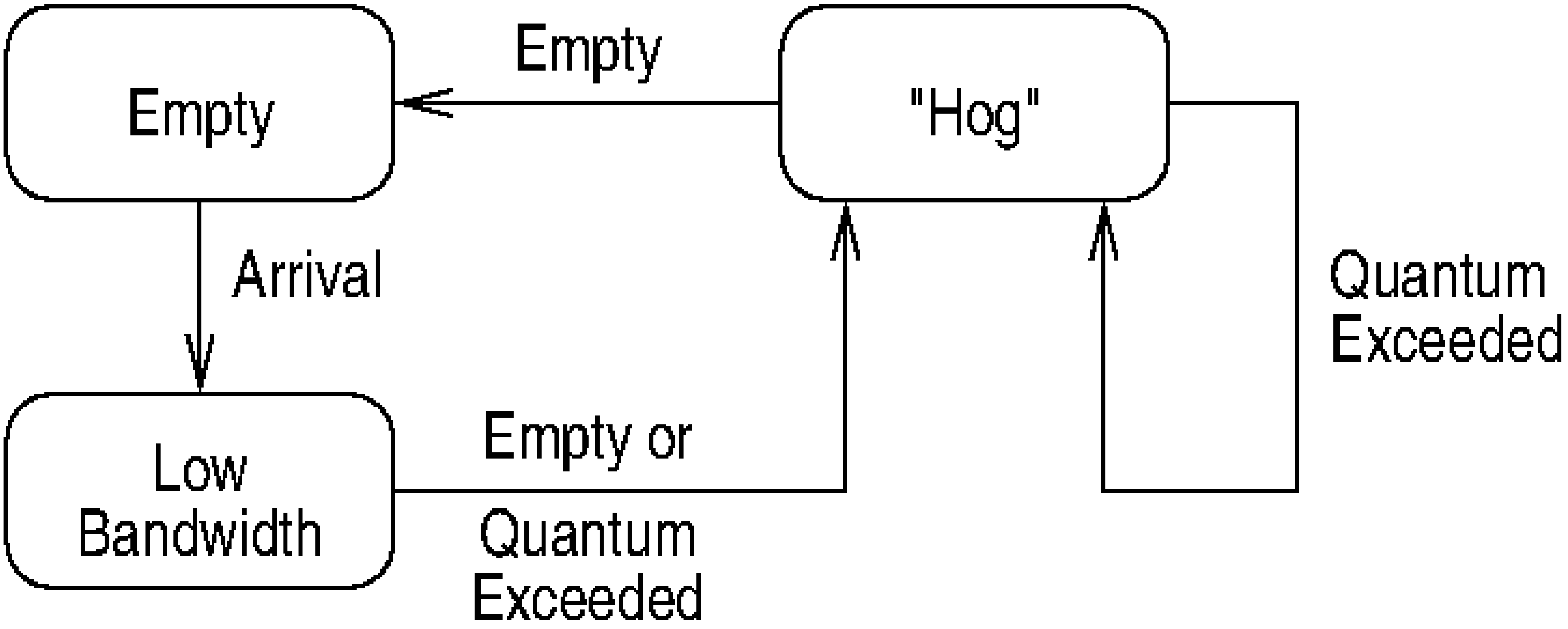
FQ_Codel Principles

- HEAD DROP, not tail drop.
- Fill the pipe, not the queue
- Queues are shock absorbers
- What matters is the delay within a flow.
- Shoots packets in elephant flows after they start accumulating delay. Don't shoot anything else!
- Provide better ack clocking for TCP and related protocols
- Let smaller, sparser streams, like VOIP, & DNS slip through

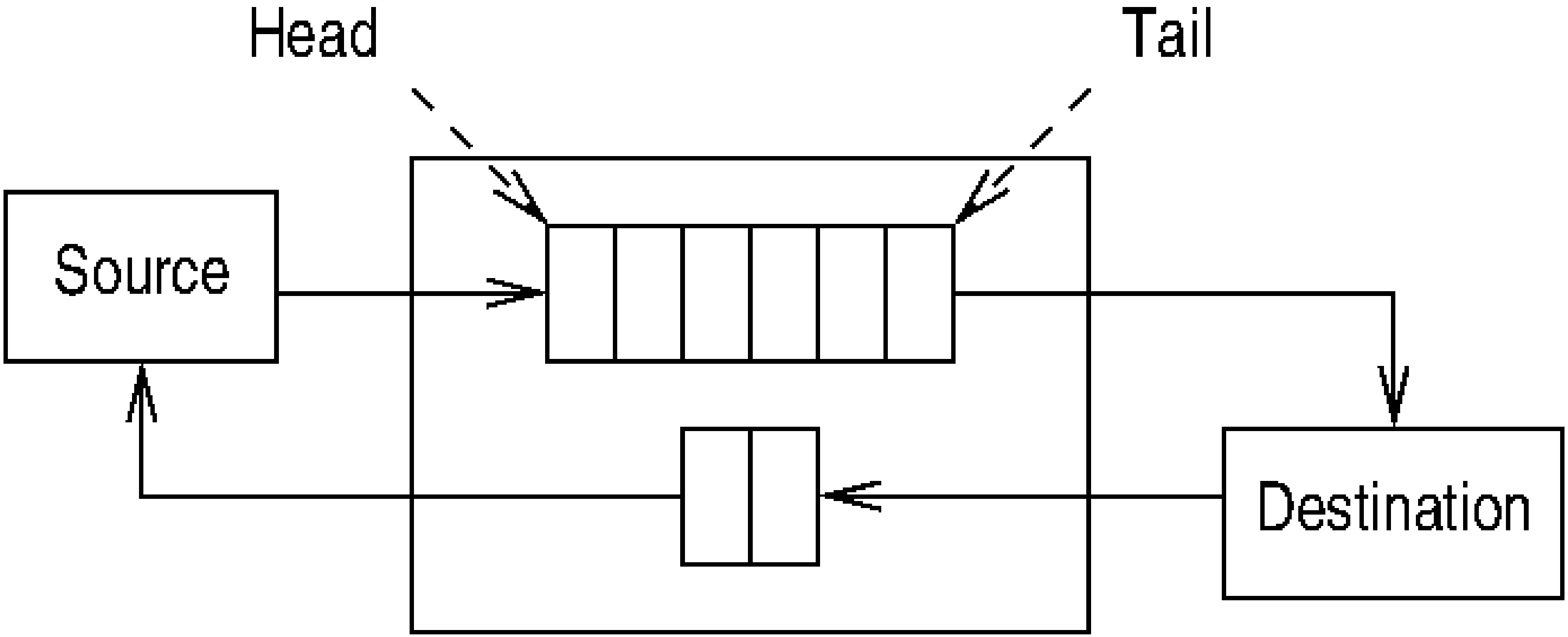
FQ_Codel starts with DRR Fair Queuing, flows by quintuple



Adds optimizations for sparse streams, which almost eliminates the need for prioritization



And given codel's measurements... drops packets from the HEAD of the queues when things get out of hand



fq_codel's short signaling loop and better mixing

- Increases network utilization
- Dramatically improves response time
- Improves interactive traffic enormously
- Makes for a more “shareable” home, small business, or corporate network
- And is implementable, in everything that needs it, in a few hundred lines of code.

“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

What role for this work with battlemeshers?

Way faster wifi networks – especially with multiple stations transmitting – after we get some new kernel code done!

Hopefully **more reliable** wifi networks!

Shared development, modeling and testing

Improvements to hardware offload architectures

And

Better Wifi for everyone...

Hopefully.

Bufferbloat.net Resources

Reducing network delays since 2011...

Bufferbloat.net: http://bufferbloat.net

Email Lists: http://lists.bufferbloat.net (codel, bloat, cerowrt-devel, etc)

IRC Channel: #bufferbloat on chat.freenode.net

Codel: https://www.bufferbloat.net/projects/codel

CeroWrt: http://www.bufferbloat.net/projects/cerowrt

Other talks: http://mirrors.bufferbloat.net/Talks

Jim Gettys Blog: http://gettys.wordpress.com

Talks by Van Jacobson, Gettys, Fred Baker, others:

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

Netperf-wrapper test suite:

https://github.com/tohojo/netperf-wrapper