Cutting queuing delay

Some advice for network device driver writers, firmware developers, and hardware designers

Dave Taht
CTO, Teklibre, LLC
dave.taht@gmail.com
Co-Founder, Bufferbloat Project
Architect, CeroWrt project
Director ”Make-wifi-fast” project

One slide summary

- Get your firmware to buffer minimally (<1ms), provide hardware flow control and useful statistics. Punt complexity to the OS.
- Get BQL support in your drivers, and turn on fq_codel in your network stacks. You’re shipping 4.1! Add 6 lines of code, do some testing, and boom, delay free product.
- Ship sqm-scripts for quality queue management via shaping
- Test at the lower rates especially for “latency under load”, “Bufferbloat”
- Always try to match the input rate to the output rate
- Think about these principles in future hardware designs.
- Take a look at “fq_codel for wifi” as one way to handle wildly variable line rates
- What I’m going to talk about applies primarily to ethernet, dsl, cable, gpon, fiber, wifi, 5G enode bs, backhauls and handheld clients. Anybody using those?
- There are other solutions for data centers.
- I care primarily about the maddening parts of the internet that 6B people use in the last mile, and want to see the technologies we’ve developed in the bufferbloat project “cross the chasm” from early adoptors, geeks, and gamers into mainstream business gear.

About bufferbloat.net

We are a loose group of engineers, theorists, sysadmins, and academics that **really hate network latency**.

- Working to reduce network lag (in linux and BSD) since 2011
  - Old problems, new algorithms, new ideas, new code
  - We’ve worked hard to not patent anything
  - Mostly everything is open source, dual BSD and GPL licensed, and well described in the literature in open access publications, IETF RFCs, upstreamed to Linux & BSD, and so on.
  - Backed by the open inventions network (OIN) & NLNET.
  - Lots of help from OpenWrt, Google, Redhat, academia, and the open source community.
- Reference source for codel, fq_codel, sch_cake, sqm-scripts, flent, and irtt

  for linux, ns2, ns3.

About me

- I’ve been on the Internet, doing neat stuff, since 1985.
- In the bufferbloat project we’ve given away all the good stuff (who will pay for 7 years of R&D anymore?), my living comes from helping companies turn it into product or into production and... sometimes solving thorny theory problems.
- I’m finishing up a project adding better wifi emulations to netem at google at the moment. Also just finished shipping “sch_cake” in openwrt and linux mainline.
- I’m mostly interested in filling the gap between theory and reality, between papers and product.
- Lab notebook and rants: http://blog.cerowrt.org/post/
- I’d **dearly** like to rip the inessential latency out of **everything**.
- This talk is not about the technical details of fq, codel, fq_codel, bql, nor all the theory behind them. It’s about the APIs to reduce latency.
- There’s plenty of other talks on this link.

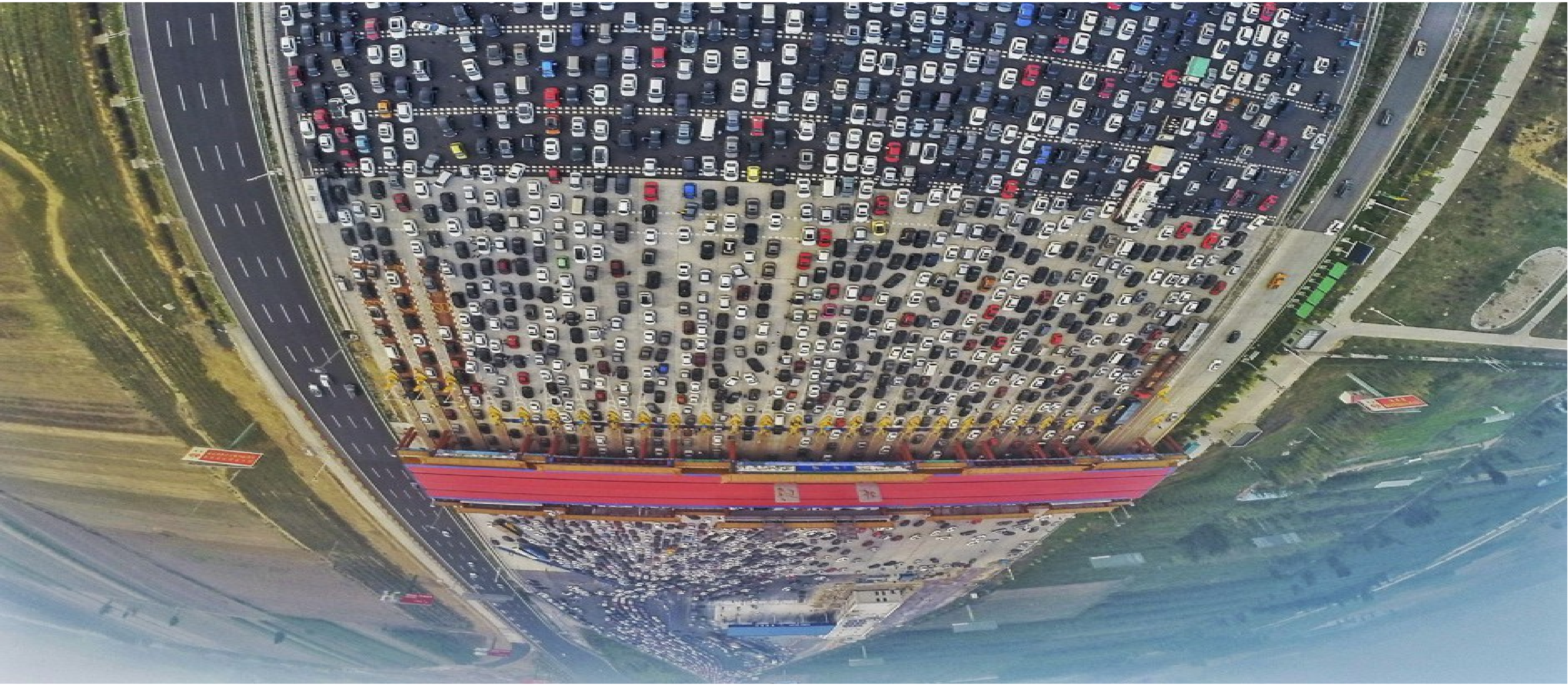
Bad Latency is everywhere

- Keyboard latency – Dan luu
- Network latency – what this talk is about
- Video latency – 16ms frames are worse than scanlines were in some applications
- HDMI and up/down conversion latency
- Audio latency
- Fly by wire latency – in any form of R/T control
- GPU latency
- Videoconferencing encoder/decoder latency
- I hate latency, generally. I’d rather drive a sportscar than an RV.

What’s bufferbloat?

- Buffering is what do you do when offered load exceeds capacity. Bufferbloat is what happens when you mistakenly think that alone can solve reducing the load.
- See wikipedia for more.
- Excess buffering is everywhere:
  - http://www.dslreports.com/speedtest/results/bufferbloat
  - http://www.dslreports.com/speedtest/results/bufferbloat?up
  - But new techniques have emerged to combat it.
  - Excess buffering makes network congestion worse.

Carmageddon



50 lanes squeezed into 20. 2.5x1 ratio of input to output led to hours of delay, for cars.

The car traffic analogy is flawed

- Packets aren’t cars.
- With FQ, bicycles, pedestrians, motorcycles all slip through.
- With dropping or AQM we blow up 60% of the cars on this road so traffic flows smoothly. Or with ECN paint them and hope the tcp on the other side notices the graffiti and slows down.
- With TCP congestion control, cameras send signals back to people on the on-ramps miles behind to stay off the road as a function of the number of crashes.
- This is all stuff that humans do, but networks don’t.
- The vehicle traffic analogy remains seriously flawed. For example, there is actually a 50,000x1 ratio between fastest and slowest links in networking.
- Van Jacobson uses an analogy of a fountain nowadays. We tend to use bottles of water going into funnels. They’re still just analogies!

Quick Bufferbloat Demo

- Lesson: Always measure what’s received, not what’s sent.
- “Carmageddon” - 50 lanes to 20. A 2.5x1 ratio of input to output led to hours of delay, for cars.
- The internet is worse: 50,000x1, from 100gbit to 128kbit. How do we deal with a rate change like that?

What Does bad latency mean?

- Both the up and down latency numbers are important. Round Trip Time (RTT) matters most.
  - Latency happens on every packet
  - A typical TCP connection takes 5 RTTs to first usable byte.
  - You click, at a 50ms RTT, and won’t see a result for 250ms.
- Lots of work (like QUIC, TCP FAST OPEN) “out there” to reduce the number of RTTs.

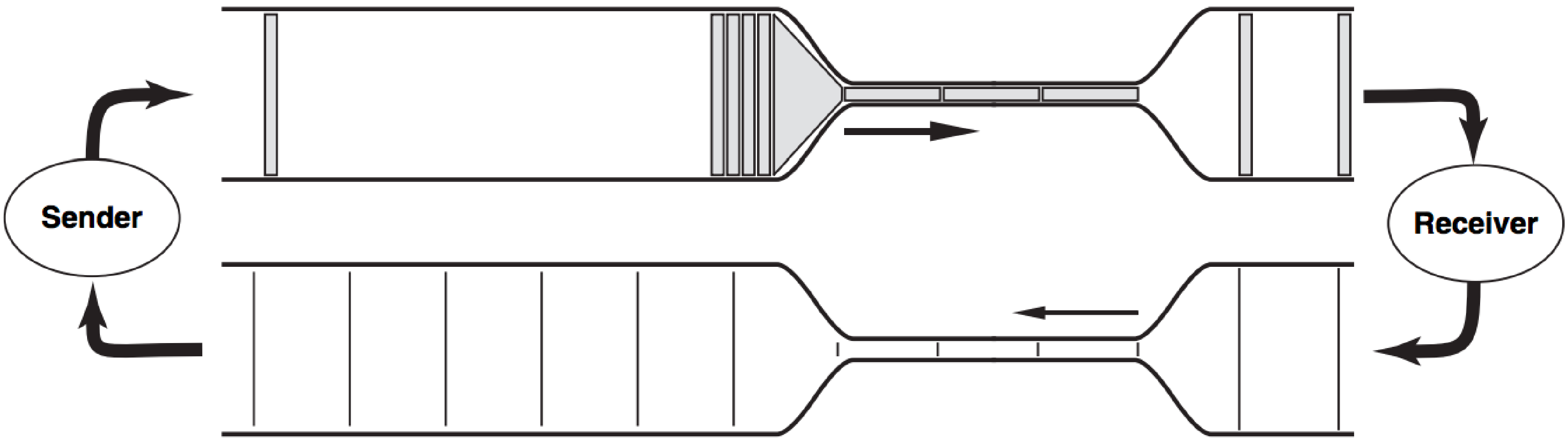
The Bandwidth Fallacy

- Bandwidth = data/interval
- Humans and marketing folk tend to use mbits/second
- But: Humans can feel intervals as low as 4ms.
- If it takes 250ms from clicking on a web page, until the first useful byte… what's the bandwidth for the first 249ms?

ZERO

- A better measure might be kbit/ms or kbit/us or bit/ns, and measuring time to first byte. Even 10ms is a LOT.

We care about the RTT



Typical Cablemodem latencies
UNLOADED UP: 16ms   DOWN: 22ms
LOADED   UP: 280ms DOWN: 680ms

Some advice for driver writers

- Keep track of bytes outstanding, not packets
- Enable FQ_codel
- Test for web page load time (PLT), VOIP MOS, Videoconferencing, while under load…
  - And weight those benchmarks higher than raw bandwidth you can achieve with tcp and udp flooding tests.
  - Measure latency under load.
  - Measure under small (20ms or less) intervals

Byte Queue Limits (BQL)

- Measures the necessary latency required for interrupt service time in the OS, based on workload, to manage a packet ring buffer to have “just enough” bytes outstanding
  - Bytes = ~time
  - At gigE, on intel, BQL is typically 40-128k per hardware queue.
  - At 10Mbit – 3k.
  - Packet ring buffers typically 512 entries up to 64k each
  - BQL Autoscales from 1 packet to thousands, from 0kbit to 50+Gbit
- BQL, as implemented, is MIAD (multiplicative increase, Additive decrease), not AIMD.
  - In theory, this is unstable
  - In practice, it hasn’t mattered much
  - It probably matters with lots of hardware queues
- I have no idea what other Oses do.
- It’s a good idea that should also end up in more firmware.
- BQL was the fundamental building block that the debloating effort was built on.
- It’s in 36 ethernet drivers including all of broadcom’s.

BQL was invented by Tom Herbert in 2011

Adding BQL is easy

- On the sent path
  - Use netdev_tx_sent_queue
- On the receive path
  - Use netdev_tx_completed_queue
- In the reset routine(s)
  - netdev_reset_queue(dev)

BQL: Sent and completed hooks

```
diff --git a/drivers/net/ethernet/broadcom/genet/bcmgenet.c b/drivers/net/ethernet/broadcom/genet/bcmgenet.c
index 49f132c..8150c74 100644
--- a/drivers/net/ethernet/broadcom/genet/bcmgenet.c
+++ b/drivers/net/ethernet/broadcom/genet/bcmgenet.c
@@ -1221,8 +1221,10 @@ static unsigned int __bcmgenet_tx_reclaim(struct net_device *dev,
        dev->stats.tx_packets += pkts_compl;
        dev->stats.tx_bytes += bytes_compl;

+       txq = netdev_get_tx_queue(dev, ring->queue);
+       netdev_tx_completed_queue(txq, pkts_compl, bytes_compl);
+
        if (ring->free_bds > (MAX_SKB_FRAGS + 1)) {
-               txq = netdev_get_tx_queue(dev, ring->queue);
                if (netif_tx_queue_stopped(txq))
                        netif_tx_wake_queue(txq);
        }
@@ -1516,6 +1518,8 @@ static netdev_tx_t bcmgenet_xmit(struct sk_buff *skb, struct net_device *dev)
        ring->prod_index += nr_frags + 1;
        ring->prod_index &= DMA_P_INDEX_MASK;

+       netdev_tx_sent_queue(txq, GENET_CB(skb)->bytes_sent);
+
        if (ring->free_bds <= (MAX_SKB_FRAGS + 1))
                netif_tx_stop_queue(txq);
```

Cleanup handlers

```
@@ -2364,6 +2368,7 @@ static int bcmgenet_dma_teardown(struct bcmgenet_priv *priv)
 static void bcmgenet_fini_dma(struct bcmgenet_priv *priv)
 {
        int i;
+       struct netdev_queue *txq;

        bcmgenet_fini_rx_napi(priv);
        bcmgenet_fini_tx_napi(priv);
@@ -2378,6 +2383,14 @@ static void bcmgenet_fini_dma(struct bcmgenet_priv *priv)
                }
        }

+       for (i = 0; i < priv->hw_params->tx_queues; i++) {
+               txq = netdev_get_tx_queue(priv->dev, priv->tx_rings[i].queue);
+               netdev_tx_reset_queue(txq);
+       }
+
+       txq = netdev_get_tx_queue(priv->dev, priv->tx_rings[DESC_INDEX].queue);
+       netdev_tx_reset_queue(txq);
+
        bcmgenet_free_rx_buffers(priv);
        kfree(priv->rx_cbs);
        kfree(priv->tx_cbs);
--
2.7.4
```

Adding BQL can be tricky

- You can’t just blindly add it.
  - You have the hardware in front of you
  - Devices & drivers tend to “pad” the sent vs received stats and they need to match exactly
  - Finding all the places where a reset can happen can take time – and a lot of resets!
  - Getting it wrong almost immediately hangs the driver
  - “Git log” will show you where others went wrong
- Still, it’s only ~6 lines of new code in your driver!

BQL @ broadcom

- In Linux 4.17 BQL is supported by all the broadcom Sky2, bnx2x, b44, and bgmac, bnxt ethernet drivers. CONGRATS!
- But: BQL is ALSO useful for **any device driver with a ringbuffer**
  - Try it there. DSL, Cable modems, fiber ONTs, etc
- If you don’t think BQL’s useful, try disabling it in one of your ethernet drivers
  - For I in /sys/class/net/your_device/queues/tx*/byte_queue_limits/limit_min
  - do echo 10000000 > $I
  - Done
- And measure latency under load at 10, 100, 1gbit, 10gbit.
- And then get it into everything else!
- (I’ll show what that looks like later in the talk)

Fair queuing

- Is generally implicit...
  - You multiplex all the ports on a switch
    - Use VOQs to further break up flows
    - Have multiple hardware queues in an ethernet device
    - More entropic than “fair”
    - Big Dumb Buffers elsewhere.
- FQ fits more flows into a RTT
- FQ balances demand faster

In 2012, Van Jacobson sayeth:

“FQ_Codel provides great isolation... if you have 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.”

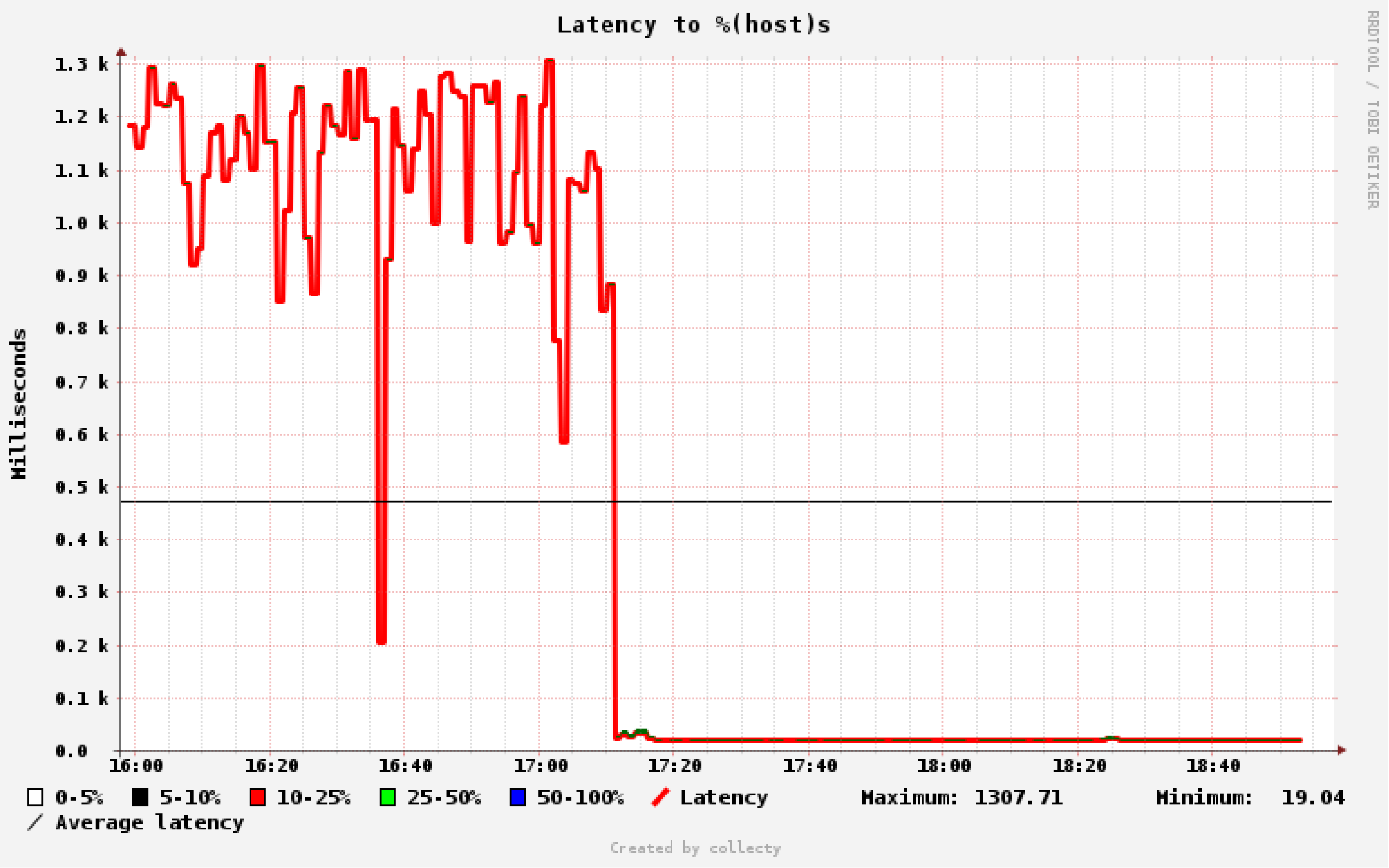
- Van Jacobson IETF 84 Talk

So that’s what we did!

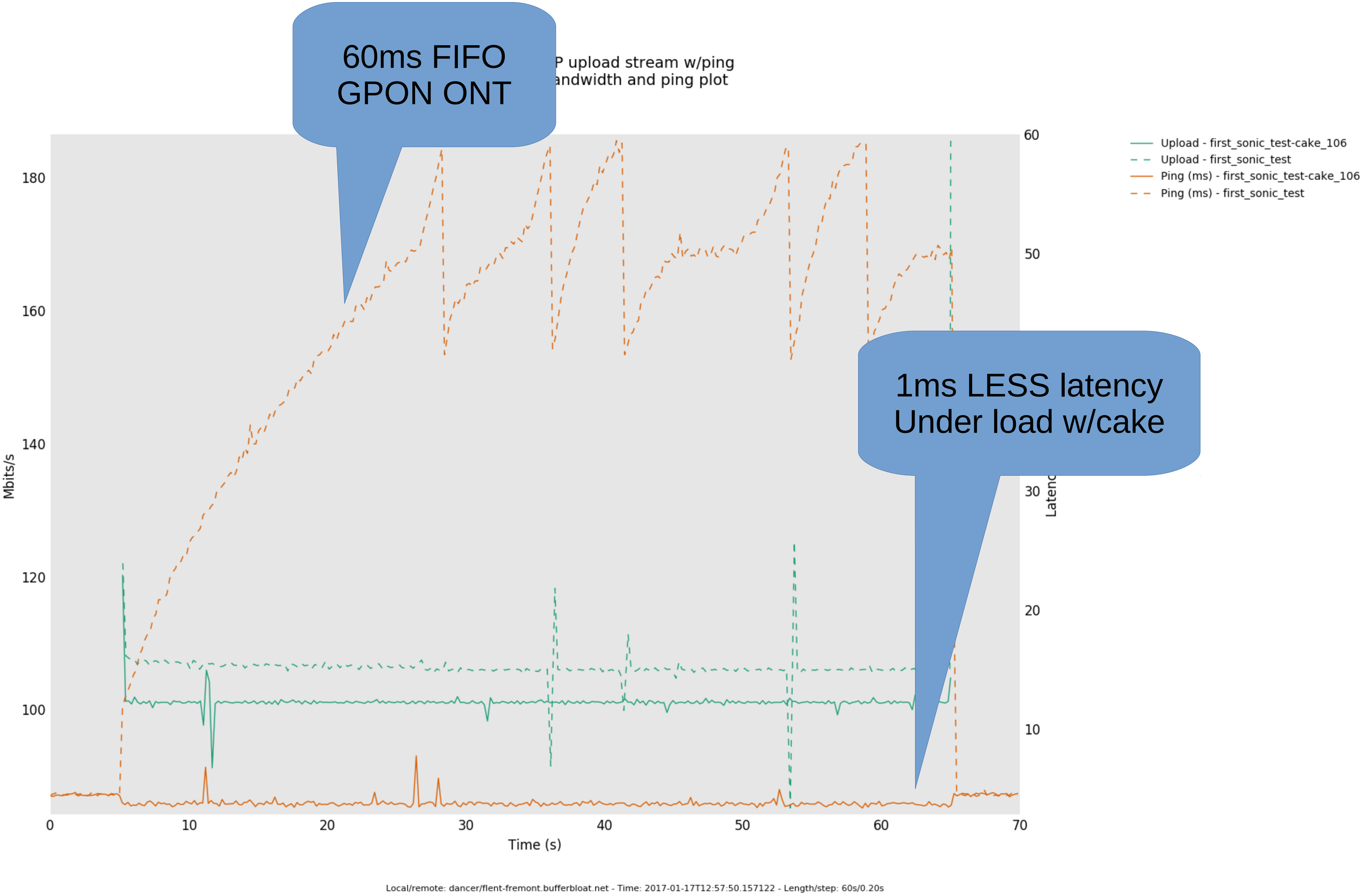
Some form of FQ for networks has swept the linux world.

- fq_codel default in openwrt, most third party router firmwares
- sch_fq used at google.
- fq_codel default for linux distros using systemd
- fq_codel/cake of the QoS system in a zillion products in many rebranded ways creating “smart queue management” (SQM)
- fq_codel part of linux wifi now for 3 chipsets
- Available in freebsd, pfsense, also
- Some success stories to follow:
  - DSL with BQL + fq_codel:
  - Cable
  - Fiber
  - Wifi

DSL w/hw flow control, bql, and fq_codel



Cake v Sonic Fiber @100Mbit



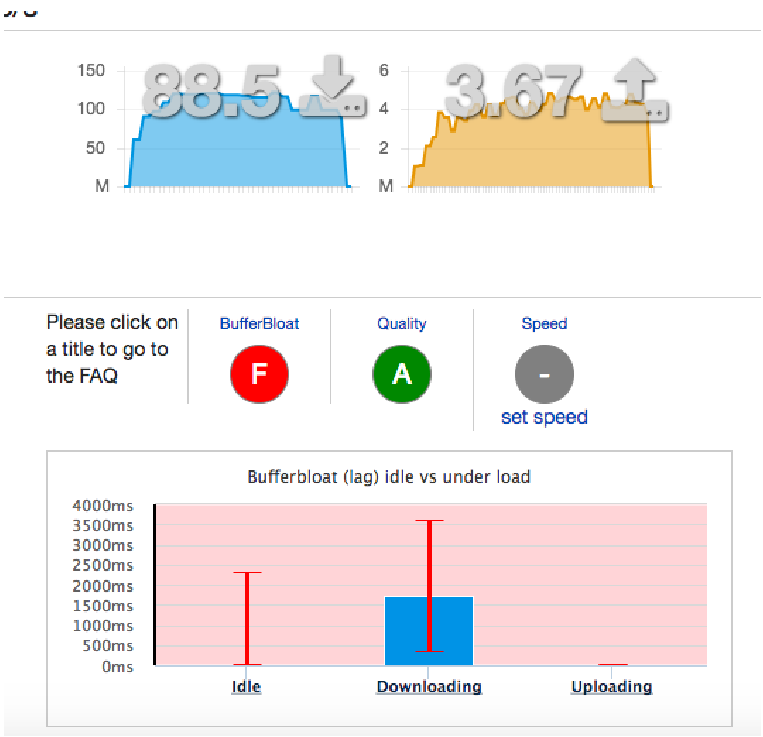
60ms FIFO
GPON ONT

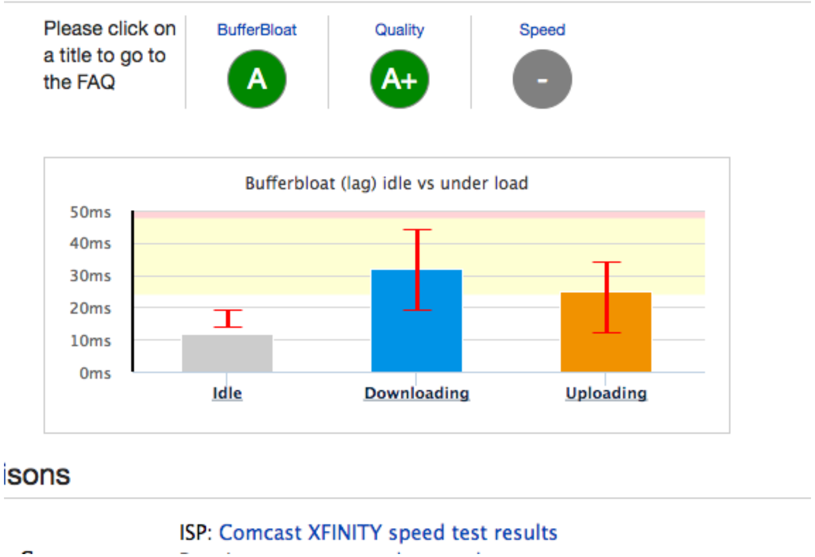
1ms LESS latency
Under load w/cake

Smart Queue Management on cablemodems

Before: http://www.dslreports.com/speedtest/36930164

After: http://www.dslreports.com/speedtest/36209937




Please click on a title to go to the FAQ
BufferBloat
A
Quality
A+
Speed
-
Bufferbloat (lag) idle vs under load
50ms
40ms
30ms
20ms
10ms
0ms
Idle
Downloading
Uploading
isons
ISP: Comcast XFINITY speed test results

FQ_Codel for WiFi

- See: Ending the Anomaly
  - 10x reduction in latency across all rates, 5x improvement in multi-station throughput
  - Code for Qualcomm’s ath9k and ath10k chips
  - Code for MediaTek’s Mt62 chip
  - Nothing else (yet) I’m aware of
  - Thousands of horrifically bad wifi designs to fix.
- Abstractions arriving to generalize this to make it work on more chips like broadcom’s
  - But you still need to do driver and firmware work to use it
- Still work to do (dynamically adjusting TXOP length)
- 802.11ax has some nice features, haven’t tried yet, would love to try
- See make-wifi-fast at lists.bufferbloat.net

Device Driver Writer Summary

- Add BQL everywhere!
- Default to fq_codel if you have BQL
- Or shape based on interface statistics using htb+fq_codel or sch_cake
- Weight PLT, VOIP, and interactive tests higher than pure bandwidth
- Profit!
- The stack above your driver is debloated… but you need co-operation from the firmware folk also.

Firmware writers

- Never eat more than 2x you can send in a ms
- If you must buffer, buffer smartly (FQ, AQM)
- Signal up the stack that “you’re full now”.
- Do the real time tasks, but punt to the OS for anything complicated.
- If you can’t buffer smartly, provide useful statistics about physical rate and queue occupancy so something smarter up the stack can get it.

Example: LTE USB modem

- It has 32k of buffer.
- I get 800kbit/sec out of it going up.
  - With 200ms totally unneeded latency

Another example
Ethernet over powerline... sucks

- Two really good examples of how to get network buffering wrong:
- https://pdfs.semanticscholar.org/271b/39469241f072a6a7fd594875e2a208ed6ce5.pdf
- http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.278.6149&rep=rep1&type=pdf
  - Horrifically bad, noisy environment… so, just add
  - tons of buffering
  - And only exert hw flow control after that fills up!
  - Homplug devices are poster children for bufferbloat
  - 2 second delays!
- Not sure if BQL would work, something more like fq_codel_for_wifi might
- I have no idea what OS runs on these things

While I’m at it, 5G bugs me

- Everybody’s slides brag about 1ms “mac” delay
  - And a glorious low latency world full of sensors, cameras, robots, games, and applications.
- Nobody talks about current LTE queuing delay
  - Frequently measured in SECONDS
- Some radicals “get it”
  - Shorter queues make handoffs easier
  - FQ makes more traffic responsive to congestion signals
  - “Managing radio networks in an encrypted world”
- Handsets, ENODE-Bs, backhauls are all bufferbloated

Some latency + bandwidth measurement tools

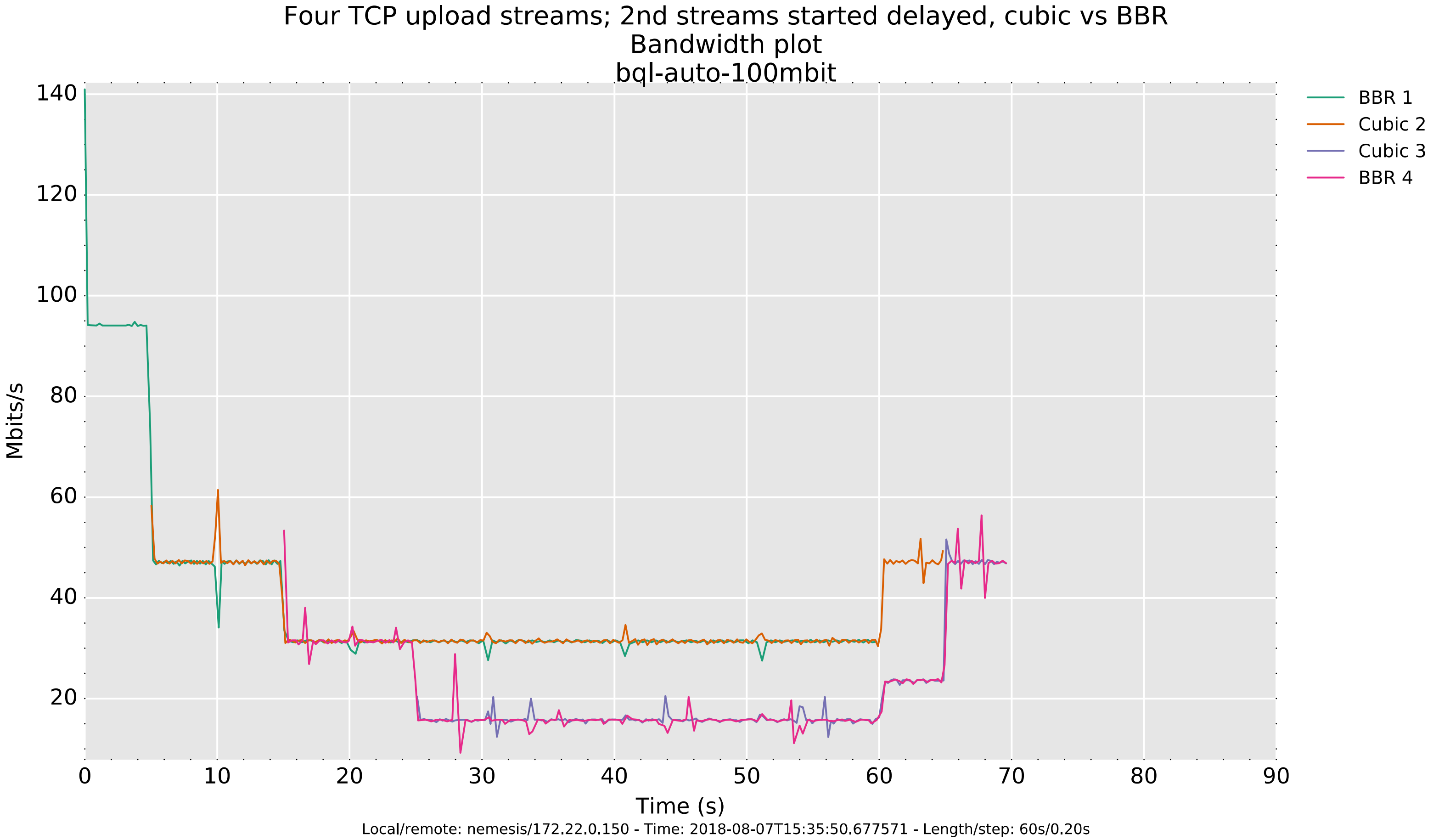
- Old fashioned test tools we use
  - TCPTRACE
  - XPLOT
  - We live and die by these
- New fangled tests
  - IRTT – Isochronous round trip tester (with OWD)
  - Flent – 100s of tests that repeatably create and plot many forms of network traffic
  - Both are **free software.**

IRTT one way delay measurements

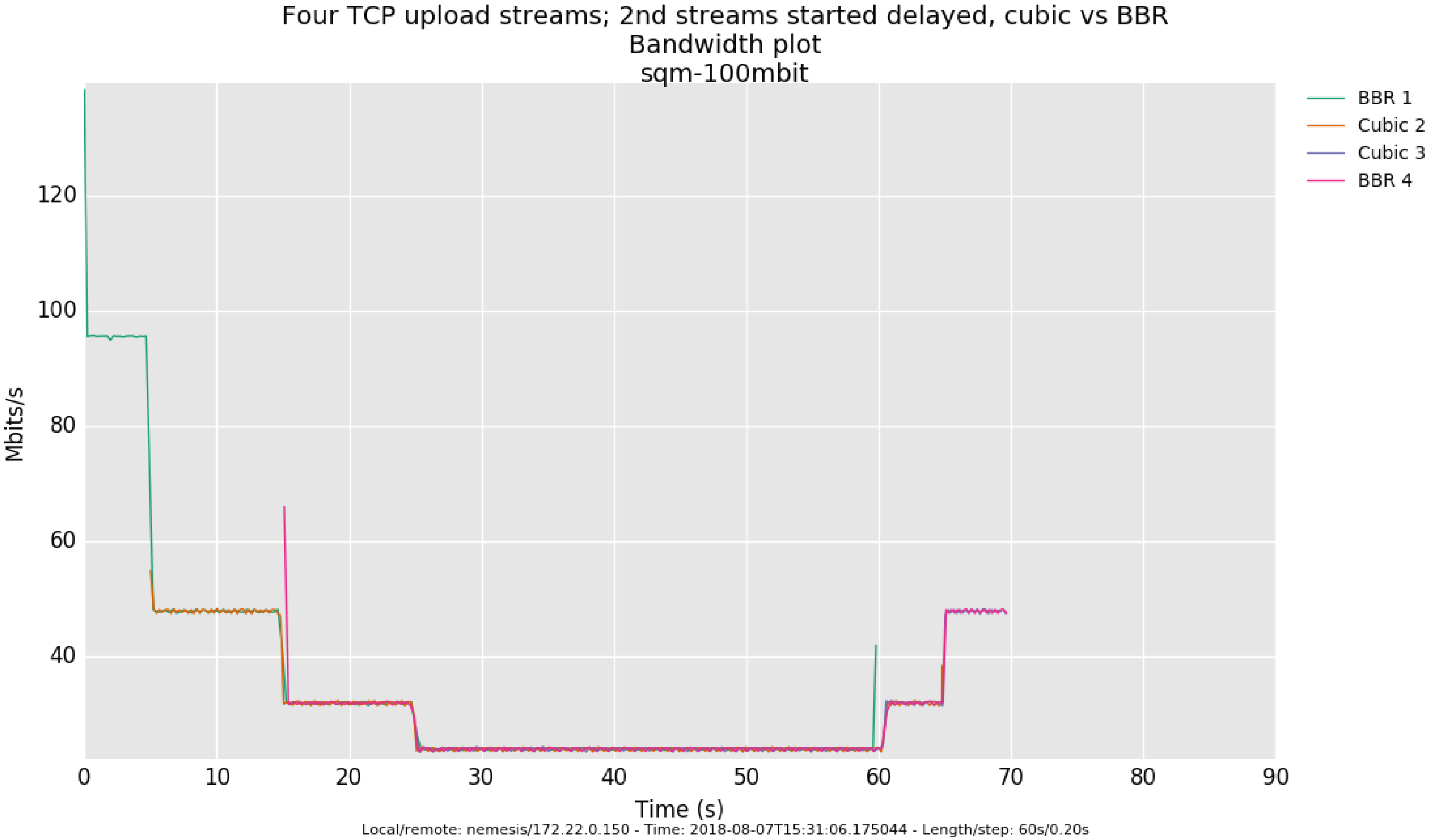
```
seq=10 rtt=12.13ms rd=8.14ms sd=3.99ms ipdv=1.13ms
^Cinterrupt

                        Min     Mean   Median      Max  Stddev
                        ---     ----   ------      ---  ------
                RTT  12.13ms  13.15ms  13.24ms  13.94ms   511µs
         send delay   3.99ms   4.67ms   4.44ms   6.09ms   609µs
      receive delay   7.85ms   8.48ms   8.46ms   9.06ms   380µs

      IPDV (jitter)   20.6µs    480µs    408µs   1.13ms   332µs
          send IPDV   69.5µs    590µs    437µs   1.37ms   464µs
       receive IPDV    262µs    520µs    483µs    791µs   191µs

     send call time   15.2µs   15.6µs             16.2µs   368ns
        timer error    3.1µs   30.6µs              177µs  51.7µs
  server proc. time   12.7µs     14µs             15.6µs  1.16µs

                duration: 10.7s (wait 0s)
   packets sent/received: 11/10 (9.09% loss)
 server packets received: 10/11 (9.09%/0.00% loss up/down)
     bytes sent/received: 660/600
       send/receive rate: 527 bps / 480 bps
           packet length: 60 bytes
             timer stats: 0/11 (0.00%) missed, 0.00% error
```

https://github.com/heistp/irtt

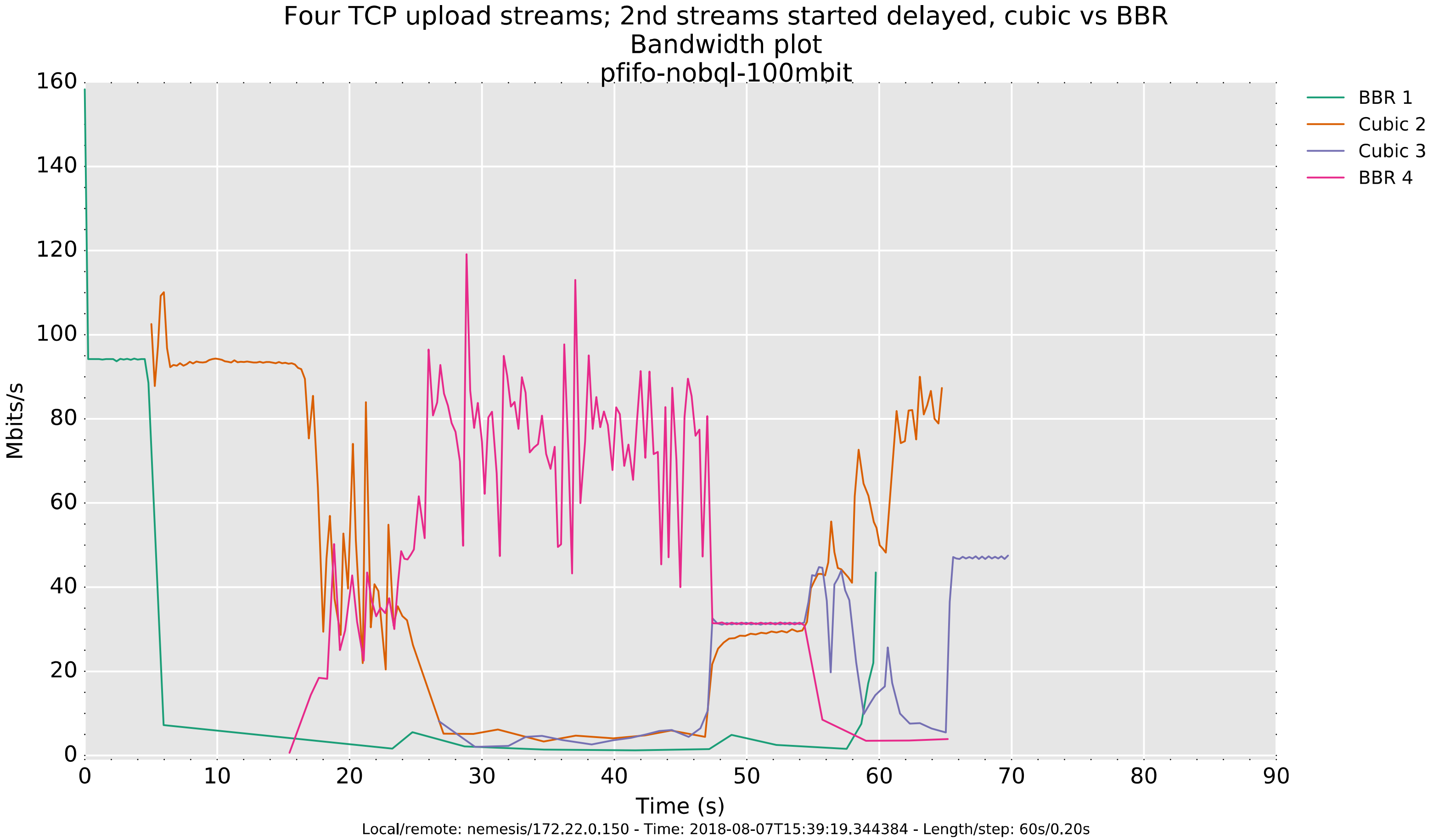
4 hw queues, bql – auto fq_codel



Flent’s tcp square wave test sch_cake shaped 100Mbit

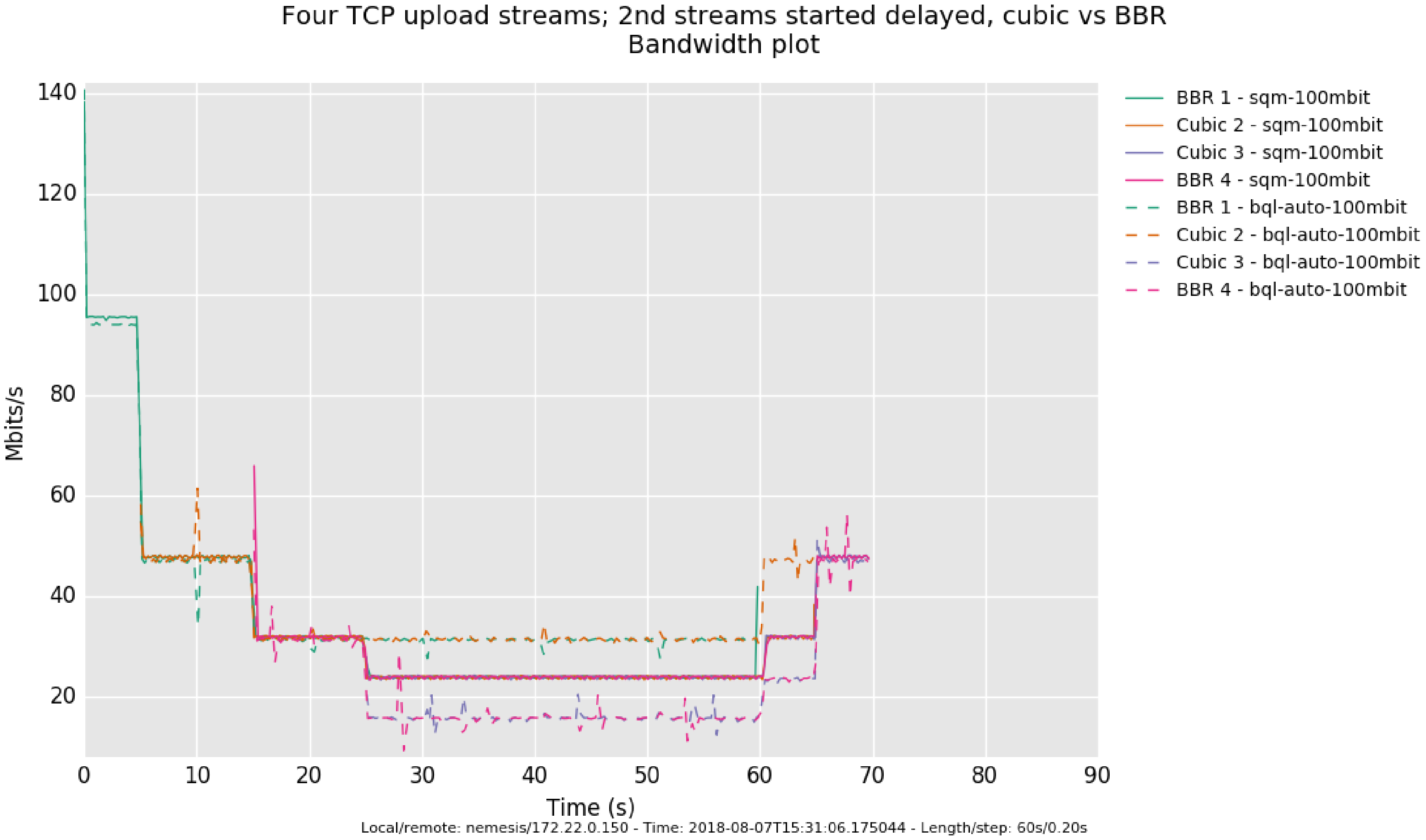


Flent’s tcp square wave test
No BQL, 1000 packet FIFO



This is basically today’s view of the Internet from the cloud

Comparing results



Advice for hardware designers

- Provide useful statistics (error rate, line rate),
- Allow for hardware flow control
- The world is full of great ideas for FQ & AQM
  - Explore some of them
- Offloads for IP hashes, timestamps, invsqrt (codel)
- Sigh: after all this work cutting queue delay in the stack, Soc’s are gaining on-board switches with more buffering
  - Please use small bufffers
  - Please build in better switches with VOQ/FQ/AQM

HW I dream of

- CPUs that can do 30 ops/cycle and context switch in 5 cycles like Mill Computer’s design
- Switches that do FQ by default
- CMTSes & DSLAMS that don’t have seconds of buffering
- Competently built DSL & Cablemodems
- Of citywide networks with 1ms edge to edge latency
- Of highly interactive VR and AR applications co-existing with fat film flows.

Next Steps

- Run back to your desk & see if BQL or fq_codel help you, your server or your chip
- Go home, try dslreports bufferbloat test.
- Try an OpenWrt router with sqm enabled.
- Drop in on the mailing list or irc channel!

“Evolving from as fast as possible - Teaching NICs about time”

- Van Jacobson’s great talk at netdevconf
  - Get rid of queues!
  - Switch to timer wheels!
  - Let the host schedule all the packets!
  - network scaling issues past 100GigE solved!
  - Film at 11.
- Please feel free to ask me technical details of various FQ and AQM algorithms, sch_cake, ecn, 802.11ax, the babel routing protocol, or opine on any other topic.
- The only public info on this is in Van’s talk.

Any questions?

Any questions?