Reducing Latency in Linux Wireless Network Drivers

Tim Shepard

shep@alum.mit.edu

netdev 1.1 — Sevilla, España — 10-12 Febrero 2016

Background:

"Bufferbloat" has been an issue for awhile.

Much progress made several years ago.

Mostly fixable now with:

```
tc qdisc replace dev eth0 root fq_codel
```

(Note: do that on the queue that is feeding the bottleneck link on the path. Not needed on non-bottleneck links.)

My friend Andrew McGregor's story...

Found Nexus 5 wlan driver badly bloated.

Succeeded in explaining to Android folk.

Failed to get the ball out of his court.

.... until he talked me into working on this.

I spent most of 2015 confused, until early December

Do we need to fix wireless drivers?

Or do we need to fix how they are fed?

Review:
How was the latency problem solved for non-wireless drivers?

```
fq_codel
```

and... what else?

Drivers pull packets out of the linux qdisc before they are sent for good reason.

This was and is true for non-wireless drivers too.

Key is to pull enough (performance), but not too much (bloat).

4+ years ago, BQL and DQL landed in mainline Linux

DQL provides a library that does auto-tuning, and BQL hooks it up to control the flow from the Linux qdiscs to the device transmit buffers. Watches the completion events at runtime and figures out how much to let the device have. (Just a bit more than what is needed to avoid starvation.)

What I realized in early December was that this is what is needed for wireless drivers.

But: DQL's assumptions won't work in the case of wireless.

Imagine an AP sending to two clients... one client distant (going to be low rate) and a nearby client (highest rates working robustly).

BQL/DQL assumes it is trying to tune the system's response to completion events to match *the* rate at which packets are transmitted to the system's ability to respond to a completion and hand more packets to the device for transmission.

(BQL's unit of work is bytes. B is bytes.)

Actually, what matters is not how many bytes of transmit work you've handed to a device. What matters is how long you expect that to take.

For normal (wired) network devices, bytes makes a pretty good proxy measure of the time it is going to take. (Proportionality is all that's needed. DQL will autotune the constant scaling factor out.)

OK, so what to do for wireless?

It's not going to be so simple, and I don't have the full answer yet, but here are some ideas...

Main idea: for wireless interfaces use units of time hooked up to something DQL-like in mac80211.

If using mac80211 rate control, use rate control information to convert bytes to units of time. (If not... it is going to be complicated. Maybe a Baysian estimator per destination station.)

Take the packets in the order being fed to us by the Linux qdisc (where policy is) and feed just enough of them to the lower level wireless drivers to avoid starving the transmitter's DMA engine.

Hope DQL's auto-tuning will take care of varying channel conditions.

802.11 specifies multi-queue with a priority scheme. Most or all modern wireless interfaces and their Linux drivers do implement this. Need DQL that understands these queues share the transmission capacity.

Mapping from length rate info to expected time will need to be aware of aggregation, either explicitly, or via some guesstimate.

The new intermediate transmit queues in mac80211 (in Linux since April 2015) are part of the solution. Should allow us to share the solution between multiple devices.

This patch moves the flow control (from qdisc to device driver) out of the device driver and into mac80211.

I've got a patch for ath9k to use these new intermediate queues (instead of its own internal per-tid per station queues). Seems to work. Needs more testing (1) by me, and (2) let me know if you want to help.

Will eventually want to cut other wireless drivers over to use the new intermediate queues.

I have a very crude kludge of a patch to mac80211 which hooks up a DQL instance per ac and uses it instead of the existing (fixed constant limit) flow control IFF the driver uses the new intermediate queues AND driver uses mac80211 RC.

This patch is for experimentation and demo purposes only.

And it doesn't work yet—locks up the driver. Should work any day now. :-)

Even when I get it working, I don't expect it will work right in general. But hand-tuning it should let us see how good we can get.

What I can demo today:

An AP and two associated client stations.

Power turned down to 3mW on AP to make things more interesting.

One client near the AP (on the same table, a few inches away).

Other client in next room, about 10 meteres away.

Ping (once per second) from nearby client. After 5 seconds of pinging, start a bulk download from server on the LAN.

Note how the bulk download from the second client intereferes (adds latency) to the nearby client's pings.

```
64 bytes from 192.168.5.187: seq=0 ttl=64 time=1.805 ms
64 bytes from 192.168.5.187: seq=1 ttl=64 time=1.456 ms
64 bytes from 192.168.5.187: seq=2 ttl=64 time=1.772 ms
64 bytes from 192.168.5.187: seq=3 ttl=64 time=1.518 ms
64 bytes from 192.168.5.187: seq=4 ttl=64 time=1.120 ms
64 bytes from 192.168.5.187: seq=5 ttl=64 time=1.443 ms
64 bytes from 192.168.5.187: seq=6 ttl=64 time=8.927 ms
64 bytes from 192.168.5.187: seq=7 ttl=64 time=48.938 ms
64 bytes from 192.168.5.187: seq=8 ttl=64 time=47.640 ms
64 bytes from 192.168.5.187: seq=9 ttl=64 time=137.210 ms
64 bytes from 192.168.5.187: seq=10 ttl=64 time=100.463 ms
64 bytes from 192.168.5.187: seq=11 ttl=64 time=130.380 ms
64 bytes from 192.168.5.187: seq=12 ttl=64 time=143.099 ms
64 bytes from 192.168.5.187: seq=13 ttl=64 time=153.504 ms
64 bytes from 192.168.5.187: seq=14 ttl=64 time=141.456 ms
64 bytes from 192.168.5.187: seq=15 ttl=64 time=172.497 ms
64 bytes from 192.168.5.187: seq=16 ttl=64 time=144.905 ms
64 bytes from 192.168.5.187: seq=17 ttl=64 time=173.305 ms
64 bytes from 192.168.5.187: seq=18 ttl=64 time=119.317 ms
64 bytes from 192.168.5.187: seq=19 ttl=64 time=128.221 ms
```

Oops, that was without fq_codel.

Much of that bloat was in the Linux default qdisc.

```
tc qdisc replace dev wlan0 root fq_codel
```

on the AP and try again.

```
64 bytes from 192.168.5.187: seq=0 ttl=64 time=1.533 ms
64 bytes from 192.168.5.187: seq=1 ttl=64 time=1.429 ms
64 bytes from 192.168.5.187: seq=2 ttl=64 time=1.438 ms
64 bytes from 192.168.5.187: seq=3 ttl=64 time=1.775 ms
64 bytes from 192.168.5.187: seq=4 ttl=64 time=1.426 ms
64 bytes from 192.168.5.187: seq=5 ttl=64 time=1.459 ms
64 bytes from 192.168.5.187: seq=6 ttl=64 time=23.243 ms
64 bytes from 192.168.5.187: seq=7 ttl=64 time=7.401 ms
64 bytes from 192.168.5.187: seq=8 ttl=64 time=2.281 ms
64 bytes from 192.168.5.187: seq=9 ttl=64 time=13.135 ms
64 bytes from 192.168.5.187: seq=10 ttl=64 time=18.423 ms
64 bytes from 192.168.5.187: seq=11 ttl=64 time=19.316 ms
64 bytes from 192.168.5.187: seq=12 ttl=64 time=10.165 ms
64 bytes from 192.168.5.187: seq=13 ttl=64 time=17.567 ms
64 bytes from 192.168.5.187: seq=14 ttl=64 time=21.018 ms
64 bytes from 192.168.5.187: seq=15 ttl=64 time=59.292 ms
64 bytes from 192.168.5.187: seq=16 ttl=64 time=55.228 ms
64 bytes from 192.168.5.187: seq=17 ttl=64 time=33.936 ms
64 bytes from 192.168.5.187: seq=18 ttl=64 time=13.610 ms
```

and now again with fq_codel and ath9k using the mac80211 intermediate queues...

```
64 bytes from 192.168.5.187: seq=0 ttl=64 time=1.789 ms
64 bytes from 192.168.5.187: seq=1 ttl=64 time=1.452 ms
64 bytes from 192.168.5.187: seq=2 ttl=64 time=1.439 ms
64 bytes from 192.168.5.187: seq=3 ttl=64 time=1.456 ms
64 bytes from 192.168.5.187: seq=4 ttl=64 time=5.232 ms
64 bytes from 192.168.5.187: seq=5 ttl=64 time=1.431 ms
64 bytes from 192.168.5.187: seq=6 ttl=64 time=1.422 ms
64 bytes from 192.168.5.187: seq=7 ttl=64 time=1.429 ms
64 bytes from 192.168.5.187: seq=8 ttl=64 time=51.174 ms
64 bytes from 192.168.5.187: seq=9 ttl=64 time=1.515 ms
64 bytes from 192.168.5.187: seq=10 ttl=64 time=2.663 ms
64 bytes from 192.168.5.187: seq=11 ttl=64 time=17.367 ms
64 bytes from 192.168.5.187: seq=12 ttl=64 time=22.965 ms
64 bytes from 192.168.5.187: seq=13 ttl=64 time=9.485 ms
64 bytes from 192.168.5.187: seq=14 ttl=64 time=11.500 ms
64 bytes from 192.168.5.187: seq=15 ttl=64 time=19.749 ms
64 bytes from 192.168.5.187: seq=16 ttl=64 time=19.827 ms
64 bytes from 192.168.5.187: seq=17 ttl=64 time=16.181 ms
64 bytes from 192.168.5.187: seq=18 ttl=64 time=9.084 ms
64 bytes from 192.168.5.187: seq=19 ttl=64 time=30.826 ms
```

Please feel free to try and un-confuse me even further.

I might be more coherent tomorrow after getting some sleep.

I should be around through Friday. afternoon.

Thanks:

Andrew McGregor for starting me down this path, and continuing to listen to my confusion.

Avery Pennarun for regularly listening to me in my confusion multiple times per week for a couple of years.

Avery Pennarun and his employer Google Fiber for sending some funding my way to make this work possible.