Airtime Based Queue Limit for FQ-CoDel in Wireless Interfaces

kyan@google.com
Reviewed by: kevinhayes@google.com, *hayesr@google.com*

Objective

Background
- FQ-CoDel
  - FQ-CoDel on wireless interfaces
- Airtime Fairness (ATF)

Design Overview
- Implementation

Test results

Objective

To improve Wireless Quality of Service (QoS) by providing an Airtime Fairness (ATF) based Active Queue Management (AQM) layer that enforces queue limit in airtime, reducing overall packet delivery latency and significantly mitigating bufferbloat in wireless interface.

More specifically:

- Reduce Wi-Fi packet transit latency.
  - Mitigate “bufferbloat” and reduce transmit queue depth when the link is oversubscribed.
  - Reduce head of line blocking.
- Improve overall wireless network performance by enforcing airtime fairness.
  - Improve airtime usage fairness by preventing stations with low data rate/weak signal (e.g. from long range) from obtaining a disproportionately large share of airtime as compared to stations with high data rate/stronger signal.
  - Improve aggregated throughput and minimize adverse impact on peak throughput while reducing latency.
- Behave robustly under diverse link conditions.

Background

Bufferbloat is a well known issue where excessive buffering of packets forms long-term standing queues, which only adds delays without providing the benefit of queueing, i.e, absorbing jitter and improving throughput. As shown in Figure 1, once the packets in flight exceed the Bandwidth-Delay Product (BDP), it only increases latency without improving throughput. Bufferbloat is mostly caused by a mismatch between TCP's windows size and link bandwidth, and the large memory buffers on some network edge devices.

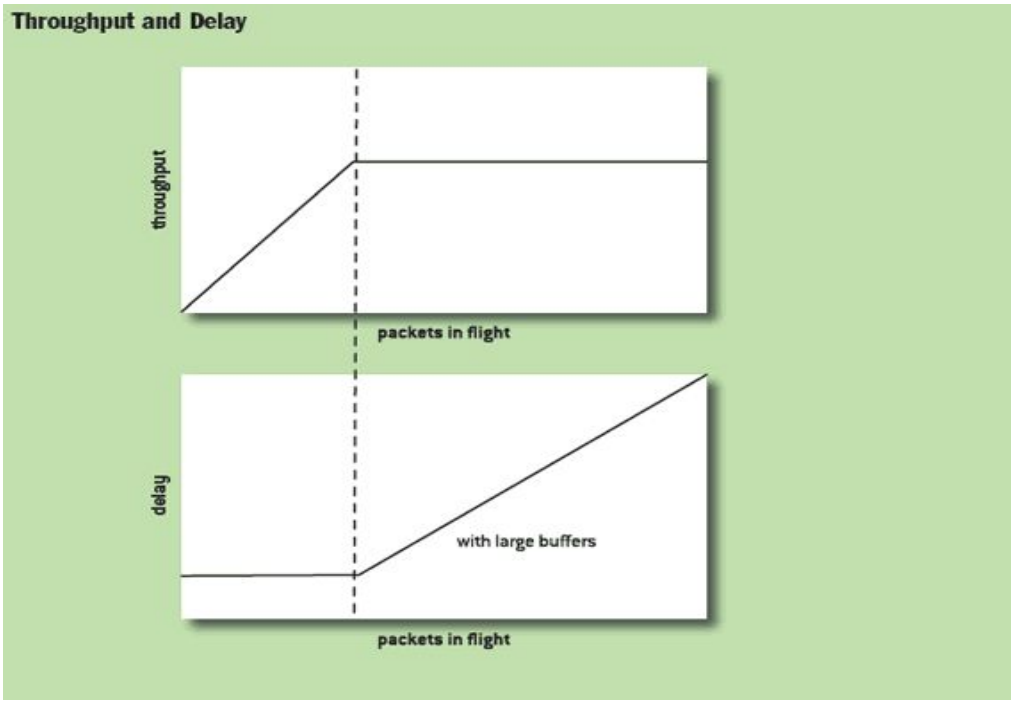

Figure 1. Latency and throughput vs packets in flight/queue size.
From: Bufferbloat: Dark Buffers in the Internet

Bufferbloat can cause severe packet latency when the link is oversubscribed. For example, a far away client with weak signal and low data rate doing heavy downloading can queue excessive amount of packets at the AP, causing all stations to experience long latency, including those stations with latency-sensitive traffic such as voice and real time video gaming.

FQ-CoDel

FlowQueue-Codel (FQ-CoDel) is a powerful tool for fighting bufferbloat and reducing latency. It is a hybrid packet scheduler/Active Queue Management (AQM) algorithm built as a combination of Controlled Delay (CoDel), an AQM algorithm, and flow based queuing (FQ).

CoDel is an AQM algorithm to control queuing behavior, developed by Van Jacobson and Kathleen Nichols, and implemented by Dave Täht and Eric Dumazet in the Linux kernel.
Its key innovation is its use of dwell (aka sojourn) time during which a packet stays in the queue instead of using queue depth as the indication of congestion. CoDel operates by monitoring the sojourn time. Once the sojourn time is above the "target" for a time greater than its "interval" parameter, the CoDel state machine enters the "dropping" state, and starts to drop packets from the head of the queue to (hopefully) signal the remote end to slow down to shrink queue size. The dropping interval is governed by CoDel's control law, which is inversely proportional to the square root of number of packets being dropped. E.g., if the scheduling interval is 100ms, then the dropping interval for n packets are: 100, 100/sqrt(2), 100/sqrt(3), … 100/sqrt(n). CoDel continues to monitor the sojourn time of queued packets during the dropping state. It will exit the dropping state and reset the dropping interval once the sojourn time goes under the target again.

FQ-CoDel, invented by Eric Dumazet, combines flow based queueing with CoDel. Traffic is classified into flows stochastically by hashing the 5-tuple of IP protocol number, source and destination IP addresses and port numbers. Each flow is managed by the CoDel AQM algorithm independently, and has its own queue and CoDel state variables. FQ-CoDel provides both fairness and isolation among queues, so low bandwidth (and yet important) flows like DNS queries are less likely to be blocked by flows with heavy traffic.

FQ-CoDel on wireless interfaces

Applying FQ-CoDel on a wired network has been a great success since its appearance in Linux kernel 3.5. However, adapting FQ-CoDel to wireless interface has been approved to be challenging. In order to support the wireless protocol's MAC layer operation such as frame aggregation, subframe retries and power saving mode, there are multiple layers of queues in the wireless networking stack: in mac80211, in the host driver, and in firmware for some architectures. Unless queues in all layers are properly managed, they suffer from long latency due to bloated queues in one or more layers.

Direct apply FQ-CoDel as a qdisc on top of wireless network interface doesn't help much, because it cannot manage the queues in mac80211 layer and host wireless driver. This issue was solved by FQ-CoDel based intermediate queues in the MAC layer, in the upstream Linux kernel 4.8. We will refer to the version that mac80211 driver integrated with FQ-CoDel as FQ-MAC. By integrating FQ-CoDel into mac80211 layer and providing Tx de-queue API for the host driver, FQ-MAC can effectively manage queues in the mac80211 layer and the host driver. It showed great results in reducing latency for 11n wireless drivers like ath9k.

However, the newer 802.11ac chipset trend to offload a large portion of data processing tasks, such as transmit scheduling and frame aggregation, to the firmware running in the microprocessor inside the wireless chipset. As a result, deep queue often builds up in firmware,

and this additional layer of unmanaged queue become the new source of the bufferbloat problem.

CoDel requires the lower layers to have limited queues in order to get a good measure of sojourn time; recall that sojourn time is intended to represent the total time the packet spends in the queue. Wired links tend to have relatively moderate queuing in the driver, and since Linux 3.4, manage these queues via the Byte Queue Limits algorithm. Unfortunately, the opposite is usually true for Wi-Fi systems. When the combination of the host-based wireless driver and the chip-based firmware queues are considered, deep firmware queues are typically used in order to achieve good link utilization and to manage frame aggregation, which are critical to improving MAC layer efficiency and achieving good single station throughput. The deep firmware queues result in little flow control being applied by the host driver, resulting in most packets flowing directly through the queues in mac80211 layer and host driver. Therefore, the sojourn time, as measured by CoDel in mac80211 layer, is usually lower than the CoDel algorithm's threshold (the "target" parameter) and very few packet drops take place even when the link is backlogged. Bufferbloat is therefore not mitigated in this scenario, but we will revisit this below.

As we can see, just FQ-MAC alone is insufficient to fix the bufferbloat problem for architecture that utilized firmware queue, which is prevalent for 802.11ac based chipset such as the ath10k driver used by Google Wifi. The queues in firmware must also be managed. A common method to manage lower layer queue for FQ-CoDel is to enable BQL in the driver. However, this doesn't work for a wireless interface, as explained in the next section.

Airtime Fairness (ATF)

One characteristic of wireless interfaces is that the link speed varies greatly among clients, due to the different capability of stations (e.g. some stations may use older 802.11b/g/n chipset) and/or the station's distance to the AP. For example, when a 3x3 11ac AP operates on the 5GHz band, the transmit PHY rate may vary between 6 Mbps to 1300 Mbps. At the MAC level, the CSMA/CA mechanism for media access is designed to ensure each station wishing to transmit with a given AC (access class) receives a stochastically equal opportunity for transmission. Releasing an equal number of bytes (encapsulated in packets) to all stations transmit queues for every round of scheduling will cause the slowest station to dominate the airtime. Previous research (Performance anomaly of 802.11b) shows that if a device on the wireless network operates at different rates, ensuring throughput fairness will result in all stations effectively transmitting at the *lowest rate* in an 802.11b/g network in the worst case.

If the mac80211 layer and the wireless drivers try to maintain byte-based transmit throughput fairness among all stations, stations using a poor link rate can therefore get a disproportionately large share of airtime, causing overall degradation of performance to the BSS. Besides this, long queues may be formed to the station with poor links and cause head of line blocking to all stations, again adversely affecting the BSS. Maintaining fairness among stations in terms of *airtime* instead of bytes is the solution for the above mentioned issue.

The first version of airtime fairness implementation for ath9k first appeared in Linux Kernel 4.11. It combines airtime fairness scheduling into FQ-MAC. However, it was only for the 11n based driver architecture, namely ath9k. For 11ac based chipset that utilize firmware queues, a few challenges still need to be addressed:

- Manage the firmware queue length for 802.11ac based wireless drivers that use firmware offloading. The current upstream mac80211 scheduler (FQ-MAC) releases packets to firmware too aggressively for it to be effective. The recent proposed airtime-based TxQ scheduler in the upstream mac80211 wireless driver only enforces fairness among stations, but does not enforce an airtime limit on the number of in-flight packets released to lower layer, resulting in ineffective CoDEL operation and long latency due to bloated firmware queue when the link is oversubscribed.
- Provision to adapt to dynamic firmware queue backpressure behavior (host-push transitioning to firmware-pull in ath10k)

Overview

Here is a brief description of queue architecture in current Linux wireless networking stack for 11ac chipset that uses firmware offloading, using ath10k as example.

In the FQ-MAC mac80211, there is a pool of flow queues. Packets are first classified into flows using a 5 tuple hash (Src/Dest IP, Port and Protocol), then queued into the flow's queue. Another layer of queueing, the intermediate transmit queue (TxQ), is used to interface with the vendor-specific wireless driver. The TxQ is a per station, per AC, per TID (Traffic Identifier) queue. The intermediate TxQ shares the pool of flow queues. The TxQ itself is a container of flow queues and it has a linked list of flows which contains the packet queue for each flow. Each TxQ also has its own FQ-CoDel data structure, which contains the state variables for its state machine instance. Fq_CoDel works as the dequeue function of the TxQ to schedule packets from its list of flows for transmission.

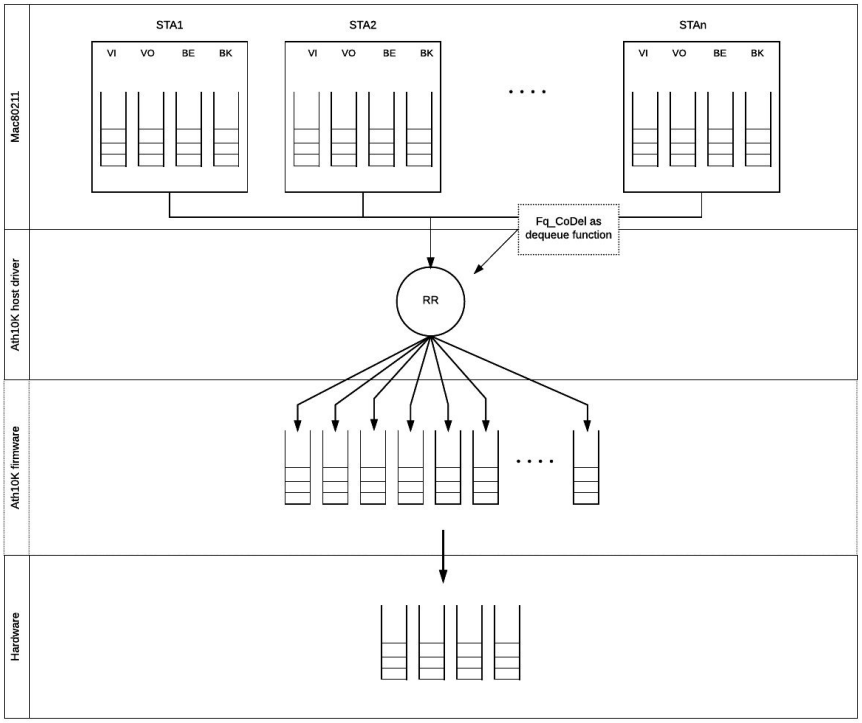

Figure 2. mac80211/ath10k transmit queue architecture

The wireless driver (Ath10k) schedules packets across all destination stations in a round robin fashion for transmitting. When a TxQ is selected for transmit, the FQ-MAC's dequeue function tries to dequeue packets fairly from its list of flows using a byte-based quantum. The CoDel algorithm is applied on flows at dequeue time, and, as discussed, packets could potentially be dropped if the sojourn time is over the limit in order to shrink the queue size and signal the remote end to slow down.

However, FQ-MAC is not effective in reducing latency when there is additional layer of deep queue in firmware. Packets only stay in the intermediate queues in the mac80211 layer transiently since there is always room in the lower layer's queue. As a result, the CoDel portion of the algorithm is mostly inactive since the sojourn time measured in the mac80211 layer is almost always lower than the target. Tests show (as in Figure 4) that the ath10k driver regularly keeps more than 1000 packets in firmware queues when the link is oversubscribed and operates past the critical point of the delay-bandwidth product.

To make FQ-CoDel work on the wireless driver with deep lower layer queue, the lower layer queue depth needs to actively managed. However, queueing in the wireless driver is essential in order to get good link utilization and form large aggregation, which is critical to improve MAC layer efficiency and get good throughput. The key is releasing just enough packets to the wireless driver/firmware to keep the hardware busy, and keep most packets in the mac80211 layer's intermediate queue that is managed by FQ-CoDel.

If a byte based queue limit is used, it is very challenging to find a suitable limit that can both prevent starvation on a fast link while preventing bufferbloat for traffic to a low speed station. With PHY rates ranging from 1.3 Gbps down to 6 Mbps for a typical 3x3 802.11ac wireless system, byte based limits aren't able to adapt to such wide dynamic ranges of PHY rates.

Byte-based fairness also means that a low speed station may, in the worst case, use hundreds of times more airtime than a fast station to transmit the same amount of data. A moderate number of packets in queue for normal stations will become excessive for slow stations, causing head of line blocking and introducing long delays for every station in the same AC. The result is the CoDel AQM either does nothing, or it starts dropping packets from all flows for every station when the delay is eventually signalled to the mac80211 layer.

Airtime based Firmware Queue Limit

The design described herein specifies a new airtime fairness-based packet scheduler that enforces Airtime Queue Limit (AQL), integrated with FQ-CoDel at mac80211 layer. We will refer it as AQL-FQ-CoDel.

AQL-FQ-CoDel not only provides an airtime based packet scheduler, but also uses airtime accounting to manage the lower layer queue depth. It makes FQ-CoDel works effectively with wireless driver. In some sense, it is similar to applying BQL in firmware, but use airtime in place of bytes.

It manages the firmware queue depth by limiting the total pending airtime for each queue. It only releasing a moderately amount of packets in order to efficiently utilize the hardware, and holding the rest of the packets in upper layer queues managed by CoDel. In this way, the CoDel algorithm in the mac80211 layer can get an accurate measurement of sojourn time, and then act properly to control the latency.

By using *airtime* as the dequeue quantum, there can be dynamic adaptations to stations with differing link speeds. The scheduler optimizes the link with high speed stations by dequeuing packets at a faster rate, since every airtime quantum translates to a relatively large number of packets in bytes, to prevent link starvation. For low speed stations, the same airtime quantum translates to fewer packets in bytes, thus reducing delay, and preventing head of line blocking; as long as the low level queue has slightly more packets than the link can sustain, throughput is not degraded. Packets stay in the intermediate queue longer for the low speed station. The CoDel algorithm can react to the longer sojourn times and take appropriate action to drop packets to signal the remote end to slow down if the minimum sojourn time is over target.

One important difference in this design versus the upstream linux version of ATF scheduler for ath9k (as of June, 2017) is how the airtime accounting is implemented. In the upstream version, airtime accounting is only used to achieve airtime-wise fairness among stations in tx scheduling,

but not used for the purpose of limiting lower layer queue depth. On the other hand, the primary goal of airtime accounting in this design is to manage queue depth in the lower layer, by estimating the pending queue size in airtime. The leads to the key difference in how airtime accounting is implemented; the airtime value is not “past” airtime spent for completed transitions in the previous scheduling period, but “future” airtime estimated using current transmit rate for frames pending in each queue (per station, per tid) in the firmware.

Another major difference between wired and wireless networks is wireless networks operate in half-duplex mode. Only one station can transmit at any given instant in a wireless network due to the shared medium. This could cause a downstream/upstream imbalance for TCP flows. When the AP applies FQ-CoDel, it regulates the sending rate when congestion is detected by occasionally drops packets or using ECN. On the other hand, wireless clients typically don’t apply FQ-CoDel, and therefore try to transmit as fast as they can, resulting in the upstream dominating the airtime. This problem occurs mostly between upstream and downstream flow competition within the same connection to a given client. The contention-based media access mechanism in 802.11 helps maintain fairness among stations.

To mitigate the upstream/downstream imbalance problem, FQ-CoDel is modified to use a frame-based quantum rather than a byte-based quantum. For a given station, FQ_MAC dequeues packets from its list of flows in a round robin way, using a byte-based quantum to enforce fairness among flows. In AQL-FQ-CoDel, by changing this byte-based quantum to a frame-based quantum, a small TCP ACK packet in reply of the upstream traffic is treated the same as a large data packet for downstream flow, thus helping to keep the upstream and downstream bandwidth utilization in balance.

The reference implementation is done for ath10k wireless driver, however, the method may be adopted to drivers for other chipset. Tests on Google Wifi prove it is very effective. The AQL-FQ-CoDel solution archives more than an order of magnitude reduction in latency when the link is oversubscribed as shown in figure 3. It can significantly improve user experience for latency sensitive applications such as voice and real time gaming when the link is overloaded.

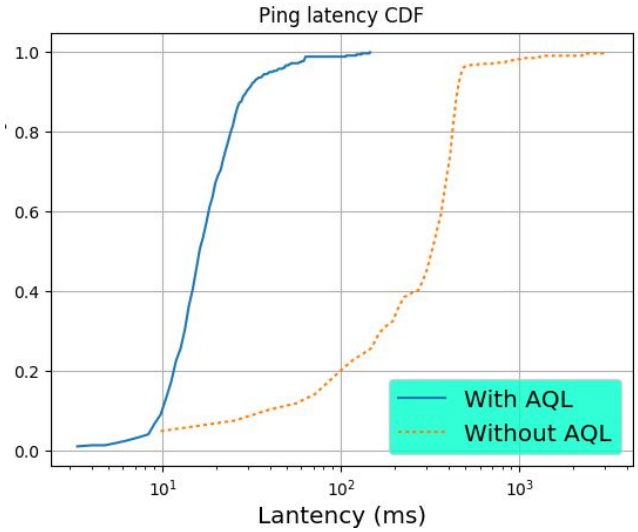

Figure 3. ICMP Ping latency with simultaneous TCP downstream traffic, with and without AQL-FQ-CoDel

In summary, there are two levels of scheduling in AQL-FQ-CoDel. At the outer layer, a Deficit Round Robin (DRR) scheduler, using airtime as the token, schedules packets from the list of active queues (station's per AC TxQ) to transmit. The selected queue is then allowed to release a fixed amount (in airtime) of packets, depending on its available tokens and the current queue depth. The queue will be skipped for the scheduling round if the estimated total airtime for all its packets exceed the airtime limit. The inner scheduler running FQ-CoDel then releases packets from the queue to firmware. The FQ-CoDel scheduler dequeues packets from the station's list of flow queues in a round robin fashion, using frames count as the quota. The outer scheduler loop maintains airtime fairness among stations and the airtime time based queue limit for each queue, and the inner loop maintains fairness among flows belonging to a given station.

The AQL-FQ-CoDel scheduler maintains airtime fairness among all stations within each 802.11 access category (AC). It may be enhanced in the future by the application of different parameters (airtime quantum, latency target and scheduling interval) to each AC/Station to further differentiate traffic, or reserve bandwidth for certain configured class/device/flows.

Implementation

Google has designed and Implemented an airtime fairness based scheduler in the ath10k wireless driver in Google Wifi's ChromeOS Linux Kernel, which is released in Feb., 2018. As described above it is a deficit round robin (DRR) scheduler with two loops. The outer loop traverses the list of active queues from all stations in a round robin fashion, allowing each queue to transmit if it has enough airtime deficit to do so. The inner loop dequeues multiple packets from the selected queue until its airtime quota becomes depleted. The queue that the inner loop operates on is the FQ-CoDel flow queue from the upper mac80211 layer. The dequeue function

is *ieee80211_tx_dequeue()*, a callback from the mac80211 layer, which calls FQ_Codel's dequeue function *fq_tin_dequeue_frame*(), and applies CoDel's control algorithm in the process.

To enable airtime fairness, the ath10k wireless driver is also modified to calculate the airtime for each frame and the overall pending airtime for each transmit queue. When packets are released to firmware, the airtime is calculated using the last reported transmit rate, and the airtime deficit is adjusted accordingly for the queue. A new callback, *ieee80211_tx_upd_rateinfo(),* was added to the mac80211 driver to record the bitrate of last successful transmission for each station. The *sk_buff* structure in kernel is modified to add a record for airtime. A new field *airtime_est* is added to the *cb* field of the *sk_buff* structure. At dequeue time, airtime is calculated, and recorded in the *airtime_est* field in the *sk_buff*. Total airtime in flight for each TxQ and the entire wireless interface also gets updated. When a frame transmission is done, either successfully acked or failed all retry attempts, the estimated airtime is retrieved from the packet when the driver processes tx completion events and in-flight airtime gets updated accordingly.

The scheduler tries to keep ~4 ms worth of packets in flight for each active transmit queue, or 8 ms if it is the only queue that is active. 4 ms is used as the queue limit because it is about the airtime of a maximal sized aggregation in an 802.11 wireless network, due to the TxOP limit. The goal is to keep the hardware busy enough to get good link utilization rate, while avoiding queueing an excessive amount of packets in the firmware queues. If there are many stations active simultaneously, the total in-flight airtime of the packets accumulated in the firmware queue could still be substantial. However, the design's goal is not to control the latency by limiting the total in-flight airtime as seen by the wireless driver/firmware, but rather to control latency using the CoDel algorithm, by letting that algorithm operate on the packet's sojourn time in the queue at the mac802111 layer. In this case, with more active stations, it will take longer for a station to get another chance to dequeue more packets. It must wait until its current tx frame has completed before its airtime quota can be refilled. Hence, the sojourn time will get longer. Once the sojourn time is above the predefined target, CoDel starts dropping packets using its control laws, and regulates the lower layer queue size from getting too big. It is similar to applying BQL to the wireless driver, except that airtime, instead of bytes is used as the queue limit.

The airtime limit for TxQs are tunable parameters. CoDel algorithm's *target* and scheduling *interval* parameters are also tunable through debugfs. The *target* parameter is CoDel's threshold of sojourn time that triggers packet drops. It is the time that packets stay in the mac80211 queue. Our testing shows CoDeL does a great job controlling the latency at mac80211 layer to the set value of *target*. But as described, there is another layer of queueing in the wireless firmware, and hence another component of latency. By employing an airtime fairness scheduler, the host driver is able to limit the queue size (in airtime) in firmware to appropriately 4-8 ms per active client. Due to frame aggregation, the dequeue behavior triggered by firmware's tx completion event is very bursty. If the target value is too small, e.g. ~10 ms, it could cause the host driver/mac80211 to drop packets unnecessarily and hurt performance. Currently, we have

tuned the default value of FQ-CoDel's *target* parameter to 35 ms and the scheduling interval to 150 ms. These are very conservative settings that behave well for all tested link conditions, and yet are still very effective. Testing shows the actual average latency correlates nearly linearly with the target value when the link is oversubscribed, if the target value is not too small. The average latency is normally about 10 - 20 ms above *target*, due to additional queueing time in the wireless driver's firmware.

This design can be further improved by dynamically adjusting those parameters according to real time network traffic conditions, especially for applications that need tighter control for latency, such as voice or gaming. Smaller targets and airtime limits can be applied to achieve tighter bounds of latency, at the cost of slightly reduced peak throughput, mostly due to smaller aggregation sizes and slight degradation of link utilization.

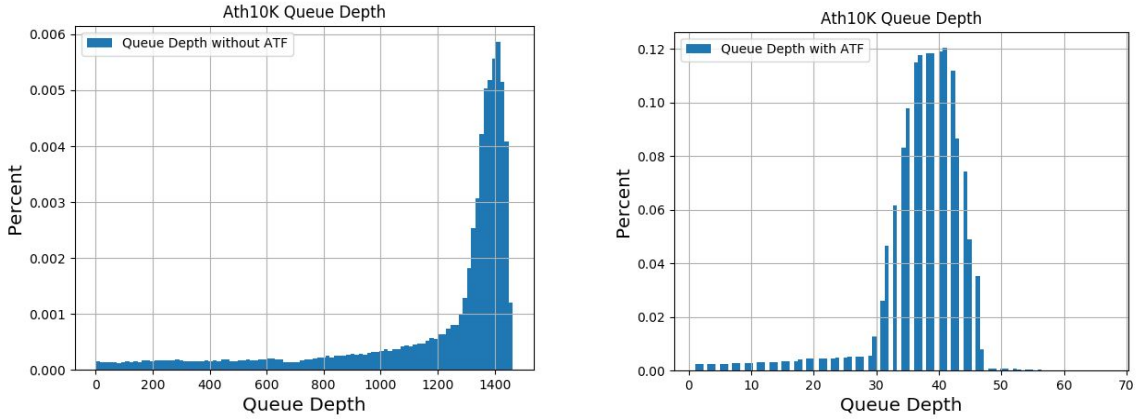

Figure 4. PDF of queue depth in ath10k firmware

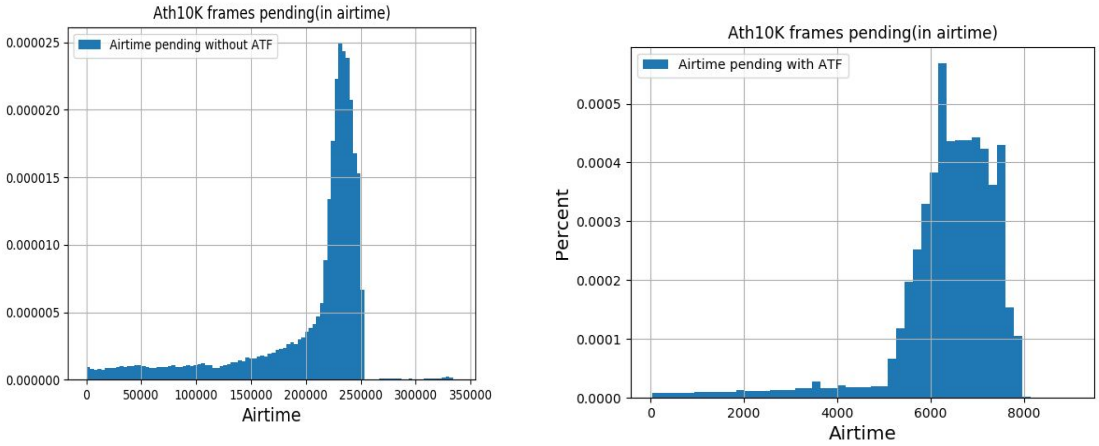

Figure 5. PDF of queue depth, in airtime (us), in ath10k firmware

Figures 4 and 5 show histograms of transmit queue depth and airtime, respectively, resident in ath10k firmware queue with FQ-MAC and with AQL-FQ-CoDel. The queue depth and airtime

are observed during a single stream TCP throughput tests using iperf. The test is done on the 5 GHz interface with moderate attenuation (~40 dB) to emulate a typical use case (about 60 Mbps actual throughput). With this AQL-FQ-CoDel enabled, the mean queue depth dropped to 37 frames from 1197 frames, and total airtime in-flight dropped to 6.3 ms from 203 ms, while maintaining about the same throughput.

The histogram shows the design achieves the goal, only releasing just enough packets to the wireless firmware to get good link utilization, while keeping the rest of the packets queued in the mac80211 layer to let FQ-CoDel control latency using the sojourn time.

The main patches to ChromeOS 3.18 Linux kernel:

CHROMIUM: net: **mac80211**: Recording last tx bit rate.
https://chromium-review.googlesource.com/c/chromiumos/third_party/kernel/+/588189

CHROMIUM: **ath10k**: Implementing airtime fairness based TX scheduler.
https://chromium-review.googlesource.com/c/chromiumos/third_party/kernel/+/588190/13

CHROMIUM: **ath10k**: use frame count as deficit for fq_codel.
https://chromium-review.googlesource.com/c/chromiumos/third_party/kernel/+/588192/13

Test results

Tests show AQL-FQ-CoDel is very effective in reducing latency for a congested link as shown earlier in figure 3. It shows ping latency comparison in the above mentioned test setup. Without AQL-FQ-CoDel, the 98th percentile ping latency is about 1000ms. With AQL-FQ-CoDel enabled, the latency is an order of magnitude smaller -- 99th percentile of packets with latency less than 100ms while maintaining comparable throughput. This performance improvement has meaningful positive impact to latency sensitive use cases such as gaming or video/voice conferencing.

Rate vs Range (RvR) test:

Figures 6 and 7 are the Rate vs Range test results for 2.4 GHz and 5 GHz downlink tests. The RvR curves are overlays of tests done with and without AQL-FQ-CoDel. It shows AQL-FQ-CoDel is able to adapt to all link conditions in this test and archive near identical throughput as without this feature. In addition, latency is dramatically reduced. The figures show a sharp increase in latency when the attenuation is getting higher (i.e. station is farther away) without AQL-FQ-CoDel. On the other hand, the latency is still almost flat when AQL-FQ-CoDel is enabled even as the attenuation (distance of the client from the AP) increases. In fact, there is more than 5x reduction in latency at the 30 dB attenuation point in the 5 GHz test, from 100 ms down to 12.5 ms. At this point, the client still has relatively good

throughput at 163 Mbps, and in the real world clients often operate at a location with even weaker signal than this point. At higher attenuation, the improvement is even more significant; there is more than 10x latency reduction at the 40 dB attenuation point.

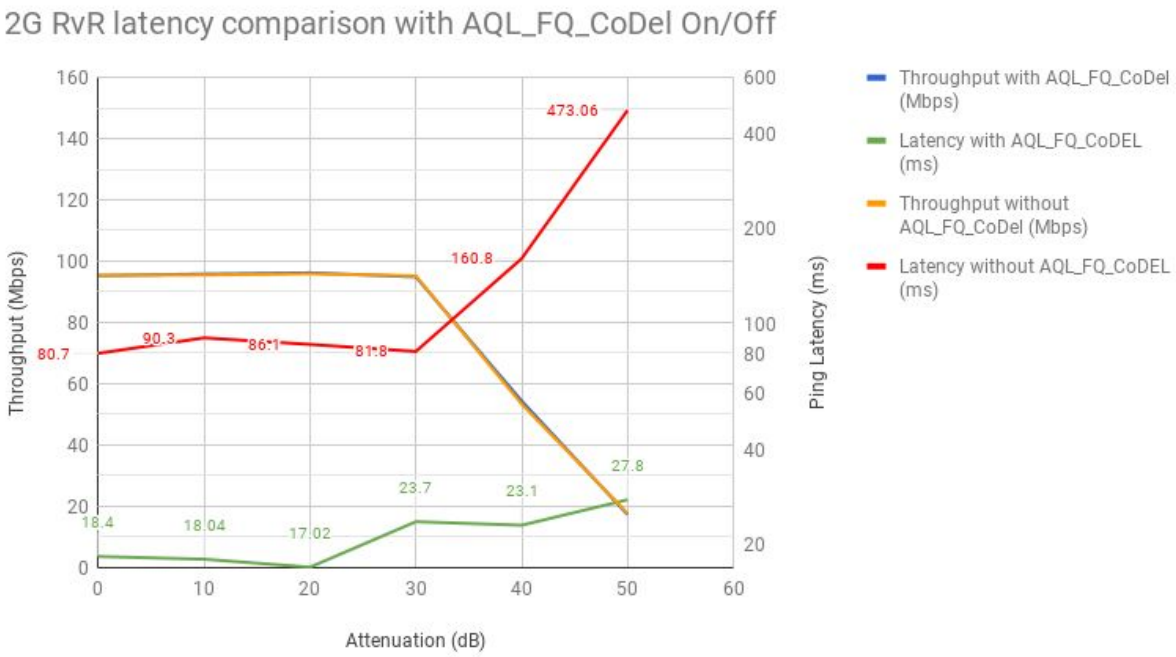

Figure 6. RvR and latency (in log scale) for 2G Downlink, with and without AQL-FQ-CoDel

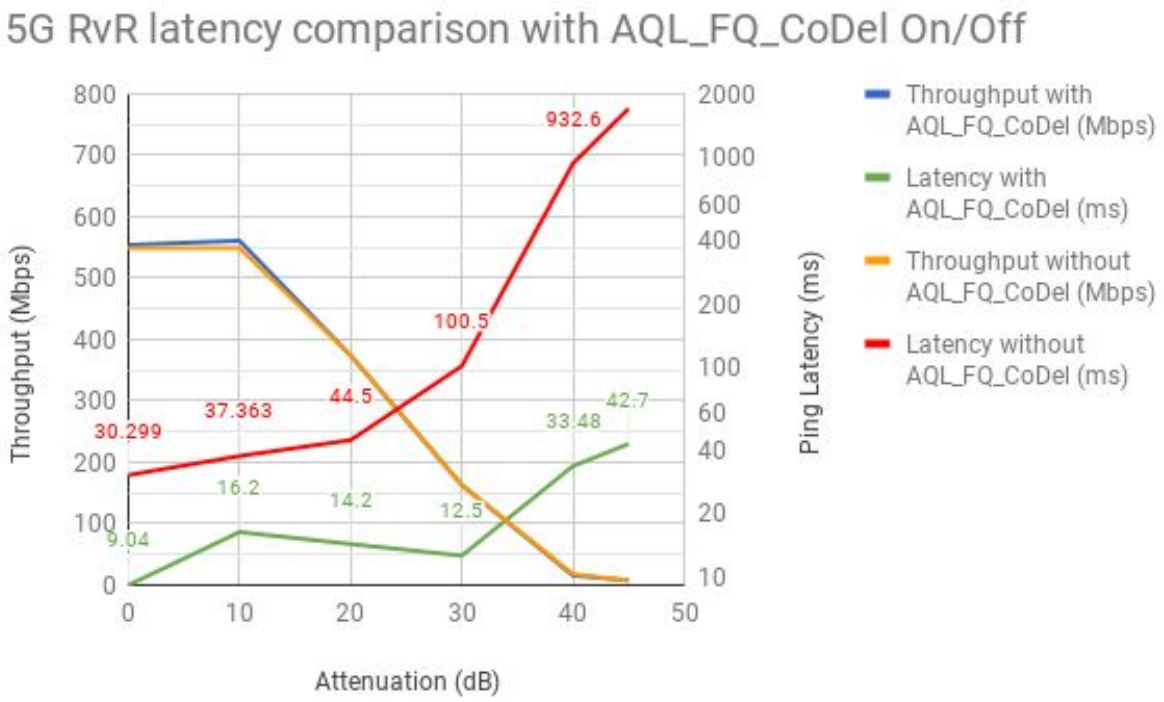

Figure 7. RvR and latency (in log scale) for 5G Downlink, with and without AQL-FQ-CoDel

In conclusion, the Airtime Fairness based AQL-FQ-CoDel implemented in Google Wifi is very effective in mitigating bufferbloat. By adding an airtime fairness scheduler on top of FQ-CoDel, in addition to achieving airtime fairness, it enforces the queue limits in airtime, thoroughly solving the challenge of adapting FQ-CoDel effectively for wireless interfaces.