Ending the Anomaly: Achieving Low Latency and Airtime Fairness in WiFi

Toke Høiland-Jørgensen, ***Karlstad University;*** **Michał Kazior,** ***Tieto Poland;***
Dave Täht, ***TekLibre;*** **Per Hurtig and Anna Brunstrom,** ***Karlstad University***

https://www.usenix.org/conference/atc17/technical-sessions/presentation/hoilan-jorgesen

This paper is included in the Proceedings of the 2017 USENIX Annual Technical Conference (USENIX ATC '17).

July 12–14, 2017 • Santa Clara, CA, USA

ISBN 978-1-931971-38-6

Open access to the Proceedings of the 2017 USENIX Annual Technical Conference is sponsored by USENIX.

Ending the Anomaly: Achieving Low Latency and Airtime Fairness in WiFi

Toke Høiland-Jørgensen
Karlstad University

Michał Kazior
Tieto Poland

Dave Täht
TekLibre

Per Hurtig
Karlstad University

Anna Brunstrom
Karlstad University

Abstract

With more devices connected, delays and jitter at the WiFi hop become more prevalent, and correct functioning during network congestion becomes more important. However, two important performance issues prevent modern WiFi from reaching its potential: increased latency under load caused by excessive queueing (i.e. *bufferbloat*) and the 802.11 performance anomaly.

To remedy these issues, we present a novel two-part solution. We design a new queueing scheme that eliminates bufferbloat in the wireless setting. Leveraging this queueing scheme, we then design an airtime fairness scheduler that operates at the access point and doesn't require any changes to clients.

We evaluate our solution using both a theoretical model and experiments in a testbed environment, formulating a suitable analytical model in the process. We show that our solution achieves an order of magnitude reduction in latency under load, large improvements in multi-station throughput, and nearly perfect airtime fairness for both TCP and downstream UDP traffic. Further experiments with application traffic confirm that the solution provides significant performance gains for real-world traffic.We develop a production quality implementation of our solution in the Linux kernel, the platform powering most access points outside of the managed enterprise setting. The implementation has been accepted into the mainline kernel distribution, making it available for deployment on billions of devices running Linux today.

1 Introduction

As more mobile devices connect to the internet, and internet connections increase in capacity, WiFi is increasingly the bottleneck for users of the internet. This means that congestion at the WiFi hop becomes more common, which in turn increases the potential for bufferbloat at the WiFi link, severely degrading performance [10].

The 802.11 performance anomaly [9] also negatively affects the performance of WiFi bottleneck links. This is a well-known property of WiFi networks: if devices on the network operate at different rates, the MAC protocol will ensure *throughput fairness* between them, meaning that all stations will effectively transmit at the lowest rate. The anomaly was first described in 2003, and several mitigation strategies have been proposed in the literature (e.g., [13, 26]), so one would expect the problem to be solved. However, none of the proposed solutions have seen widespread real-world deployment.

Recognising that the solutions to these two problems are complementary, we design a novel queue management scheme that innovates upon previous solutions to the bufferbloat problem by adapting it to support the 802.11 suite of WiFi protocols. With this queueing structure in place, eliminating the performance anomaly becomes possible by scheduling the queues appropriately. We develop a deficit-based airtime fairness scheduler to achieve this.

We implement our solution in the WiFi stack of the Linux kernel. Linux is perhaps the most widespread platform for commercial off-the-shelf routers and access points outside the managed enterprise, and hundreds of millions of users connect to the internet through a Linux-based gateway or access point on a daily basis. Thus, while our solution is generally applicable to any platform that needs to support WiFi, using Linux as our example platform makes it possible to validate that our solution is of production quality, and in addition gives valuable insights into the practical difficulties of implementing these concepts in a real system.

The rest of this paper describes our solution in detail, and is structured as follows: Section 2 describes the bufferbloat problem in the context of WiFi and the WiFi performance anomaly, and shows the potential performance improvement from resolving them. Section 3 describes our proposed solution in detail and Section 4 presents our experimental evaluation. Finally, Section 5 summarises related work and Section 6 concludes.

2 Background

In this section we describe the two performance issues we are trying to solve – Bufferbloat in the WiFi stack and the 802.11 performance anomaly. We explain why these matter, and show the potential benefits from solving them.

2.1 Bufferbloat in the context of WiFi

Previous work on eliminating bufferbloat has shown that the default buffer sizing in many devices causes large delays and degrades performance. It also shows that this can be rectified by introducing modern queue management to the bottleneck link [10, 15, 29]. However, this does not work as well for WiFi; prior work has shown that neither decreasing buffer sizes [23] nor applying queue management algorithms to the WiFi interface [10] can provide the same reduction in latency under load as for wired links.

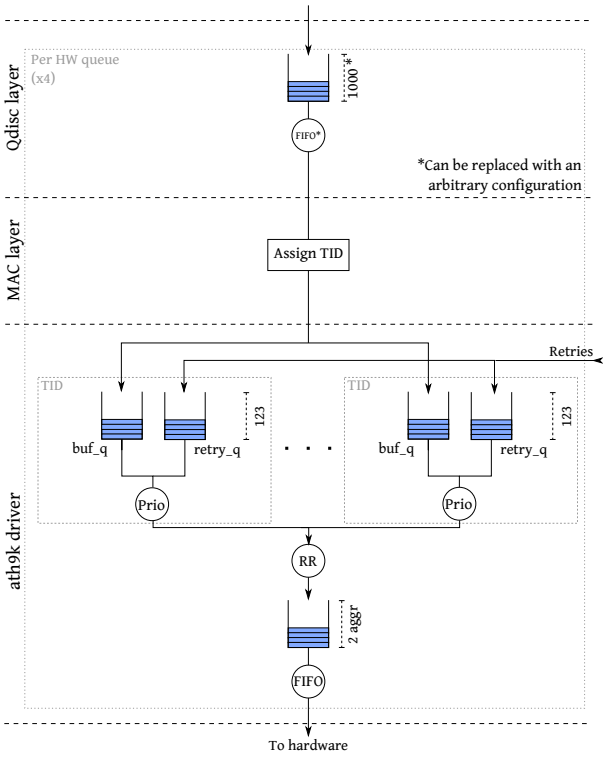


Figure 1: The queueing structure of the Linux WiFi stack.

The reason for the limited effect of prior solutions is queueing in the lower layers of the wireless network stack. For Linux, this is clearly seen in the queueing structure, depicted in Figure 1. The upper queue discipline ("qdisc") layer, which is where the advanced queue management schemes can be installed, sits above both the mac80211 subsystem (which implements the base 802.11 protocol) and the driver. As the diagram shows, there is significant unmanaged queueing in these lower layers, limiting the efficacy of the queue management schemes and leading to increased delay. Such a design is typical for an environment where low-level protocol details impose a certain queueing structure (as opposed to a wired Ethernet network, where the protocol-specific processing performed by the driver does not necessitate queueing). In WiFi this queueing is needed to build aggregates (and to a lesser extent to keep the hardware busy within the time constrains imposed by the protocol), but a similar situation can be seen in, e.g., mobile broadband devices, DSL modem drivers, and even in some VPN protocols, where the encryption processing can require a separate layer of queueing.

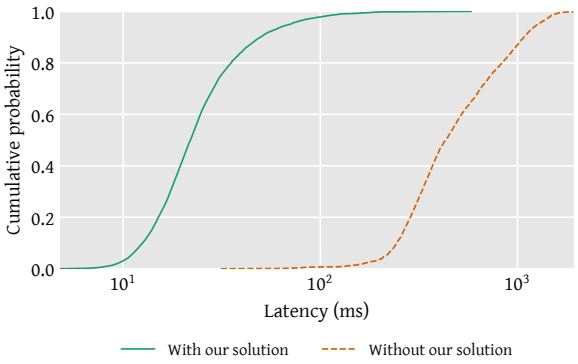


Figure 2: Latency of an ICMP ping flow with simultaneous TCP download traffic, before and after our modifications.

To solve this, an integrated queueing scheme is needed, that applies modern queue management to the protocol-specific queueing structures. In Section 3 we describe our design of such a solution for the WiFi domain. Figure 2 showcases the gain from applying our solution. The figure shows a latency measurement (ICMP ping) performed simultaneously with a simple TCP download to each of the stations on the network. The dashed line shows the state of the Linux kernel before we applied our solution, with several hundred milliseconds of added latency. The solid line shows the effects of applying the solution we propose in this paper – a latency reduction of an order of magnitude.

2.2 Airtime fairness

The 802.11 performance anomaly was first described for the 802.11b standard in [9], which showed that in a wireless network with differing rates, each station would achieve the same effective throughput even when their rates were different. Later work has shown both analytically and experimentally that time-based fairness improves the aggregate performance of the network [26], and

that the traditional notion of proportional fairness [18] translates to airtime fairness when applied to a WiFi network [12].

This latter point is an important part of why airtime fairness is desirable – proportional fairness strikes a balance between network efficiency and allowing all users a minimal level of service. Since a wireless network operates over a shared medium (the airwaves), access to this medium is the scarce resource that needs to be regulated. Achieving airtime fairness also has the desirable property that it makes a station's performance dependent on the *number* of active stations in the network, and not on the performance of each of those other stations.

The addition of packet aggregation to WiFi (introduced in 802.11n and also present in 802.11ac) adds some complexity to the picture. To quantify the expected gains of airtime fairness in the context of these newer revisions of 802.11, the following section develops an analytical model to predict throughput and airtime usage.

2.2.1 An analytical model for 802.11 with aggregation

The models in [9] and [26] give analytical expressions for expected throughput and airtime share for 802.11b (the latter also under the assumption of airtime fairness). Later work [16] updates this by developing analytical expressions for packet sizes and transmission times for a single station using 802.11n. However, this work does not provide expressions for predicting throughput and airtime usage. In this section we expand on the work of [16] to provide such an expression. While we focus on 802.11n here, the 802.11ac standard is backwards-compatible with 802.11n as far as the aggregation format is concerned, so these calculations apply to the newer standard as well.

For the following exposition, we assume a set of active stations, I. Each station, i, transmits aggregates of a fixed size of L_i bytes. In practice, the aggregates are composed of data packets, plus overhead and padding. The 802.11n standard permits two types of aggregation (known as A-MPDU and A-MSDU), which differ in how they combine packets into MAC-layer aggregates. For A-MPDU aggregation (which is the most common in use in 802.11n devices), the size of an aggregate consisting of n_i packets of size l_i is given by:

$$L_i = n_i(l_i + L_{delim} + L_{mac} + L_{FCS} + L_{pad}) \quad (1)$$

where $L_{delim}, L_{mac}, L_{FCS}, L_{pad}$ are, respectively, the frame delimiter, MAC header, frame check sequence and frame padding. However, these details are not strictly necessary for our exposition, so we leave them out in the following and instead refer to [16] for a nice overview of the details of aggregate composition.

A station transmits data over the air at a particular data rate r_i (measured in bits per second). So the time to transmit the data portion of an aggregate is simply:

$$T_{data}(i) = \frac{8L_i}{r_i} \quad (2)$$

From this we can compute the expected effective station rate, assuming no errors or collisions, and no other active stations:

$$R_0(i) = \frac{L_i}{T_{data}(i) + T_{oh}} \quad (3)$$

where T_{oh} is the per-transmission overhead, which consists of the frame header, the inter-frame spacing, the average block acknowledgement time, and the average back-off time before transmission. We again leave out the details and point interested readers to [2, 16].

Turning to airtime fairness, we borrow two insights from the analysis in [26]:

1. The rate achieved by station i is simply given by the baseline rate it can achieve when no other stations are present (i.e., $R_0(i)$) multiplied by the share of airtime available to the station.

2. When airtime fairness is enforced, the airtime is divided equally among the stations (by assumption). When it is not, the airtime share of station i is the ratio between the time that station spends on a single transmission (i.e., $T_{data}(i)$) and the total time all stations spend doing one

Aggr size	$T(i)$	Rates (Mbps)			
		PHY	Base	$R(i)$	Exp
Baseline (FIFO queue)[1]					
6892	10%	144.4	97.3	9.7	7.1
7833	11%	144.4	101.1	11.4	6.3
2914	79%	7.2	6.5	5.1	5.3
Total				26.4	18.7
Airtime Fairness					
28434	33%	144.4	126.7	42.2	38.8
28557	33%	144.4	126.8	42.3	35.6
2914	33%	7.2	6.5	2.2	2.0
Total				86.8	76.4

Table 1: Calculated airtime, calculated rate and measured rate for the three stations (two fast and one slow) in our experimental setup. The aggregation size is the measured mean aggregation size (in bytes) from our experiments and the measured rates (Exp column) are mean UDP throughput values.

[1] The aggregation size and throughput values vary quite a bit for this test, because of the randomness of the FIFO queue emptying and filling. We use the median value over all repetitions of the per-test mean throughput and aggregation size; see the online appendix for graphs with error bars.

transmission each.

With these points in mind, we express the expected airtime share $T(i)$ and rate $R(i)$ as:

$$T(i) = \begin{cases} \frac{1}{|I|} & \text{with fairness} \\ \frac{T_{data}(i)}{\sum_{j \in I} T_{data}(j)} & \text{otherwise} \end{cases} \quad (4)$$

$$R(i) = T(i) R_0(i) \quad (5)$$

Using the above, we can calculate the expected airtime share and effective rate for each station in our experimental setup. The assumption of no contention holds because all data is transmitted from the access point. As the queueing structure affects the achievable aggregation level (and thus the predictions of the model), we use the measured average aggregation levels in our experiments as input to the model.

The model predictions, along with the actual measured throughput in our experiments, are shown in Table 1. The values will be discussed in more detail in Section 4, so for now we will just remark that this clearly shows the potential of eliminating the performance anomaly: An increase in total throughput by up to a factor of five.

3 Our solution

We focus on the access point scenario in formulating our solution, since a solution that only requires modifying the access point makes deployment easier as there are fewer devices to upgrade. However, WiFi client devices can also benefit from the proposed queueing structure. And while we have focused on 802.11n here, the principles apply equally to both earlier (802.11abg) and newer (802.11ac) standards. The rest of this section describes the two parts of our solution, and outlines the current implementation status in Linux.

3.1 A bloat-free queueing structure for 802.11

An operating system networking stack has many layers of intermediate queueing between different subsystems, each of which can add latency. For specialised systems, it is possible to remove those queues entirely, which achieves significant latency reductions [1]. While such a radical restructuring of the operating system is not always possible, the general principle of collapsing multiple layers of queues can be applied to the problem of reducing bufferbloat in WiFi.

As mentioned in Section 2.1, an integrated queueing structure is needed to deal with protocol-specific constraints while still eliminating bufferbloat. What we propose here is such an integrated structure that is specifically suited to the 802.11 MAC. The components we use to build this structure already exists in various forms; the novelty of our solution lies in their integration, and some algorithmic innovations to make the implementation feasible, even on small devices.

There are three main constraints we must take into account when designing our queueing scheme. First, we must be able to handle aggregation; the 802.11e standard specifies that packets can be assigned different Traffic Identifiers (TIDs) (typically based on their DiffServ markings [25]), and the 802.11n standard specifies that aggregation be performed on a per-TID basis. Second, we must have enough data processed and ready to go when the hardware wins a transmit opportunity; there is not enough time to do a lot of processing at that time. Third, we must be able to handle packets that are re-injected from the hardware after a failed transmission; these must be retransmitted ahead of other queued packets, as transmission can otherwise stall due to a full Block Acknowledgement Window.

The need to support aggregation, in particular, has influenced our proposed design. A generic packet queueing mechanism, such as that in the Linux qdisc layer (see Section 2.1), does not have the protocol-specific knowledge to support the splitting of packets into separate queues, as is required for aggregation. And introducing an API to communicate this knowledge to the qdisc layer would impose a large complexity cost on this layer, to the detriment of network interfaces that do not have the protocol-specific requirements. So rather than modifying the generic queueing layer, we bypass it completely, and instead incorporate the smart queue management directly into the 802.11 protocol-specific subsystem. The main drawback of doing this is, of course, a loss of flexibility. With this design, there is no longer a way to turn off the smart queue management completely; and it does add some overhead to the packet processing. However, as we will see in the evaluation section, the benefits by far outweigh the costs.

We build our smart queue management solution on the FQ-CoDel queue management scheme, which has been shown to be a best-in-class bufferbloat mitigation technique [10, 15, 29]. The original FQ-Codel algorithm is a hybrid fairness queueing and AQM algorithm [11]. It functions as a Deficit Round-Robin (DRR) scheduler [24] between flows, hashing packets into queues based on their transport protocol flows, and applying the CoDel AQM separately to each queue, in order to keep the latency experienced by each flow under control. FQ-CoDel also adds an optimisation for sparse flows to the basic DRR algorithm. This optimisation allows flows that use less than their fair share of traffic to gain scheduling priority, reducing the time their packets spend in the queue. For a full explanation of FQ-CoDel, see [11].

FQ-CoDel allocates a number of sub-queues that are used for per-flow scheduling, and so simply assigning a full instance of FQ-CoDel to each TID is impractical. In-

Algorithm 1 802.11 queue management algorithm - enqueue.

```
1:  function enqueue(pkt, tid)
2:      if queue_limit_reached() then                ▷ Global limit
3:          drop_queue ← find_longest_queue()
4:          drop(drop_queue.head_pkt)
5:      queue ← hash(pkt)
6:      if queue.tid ≠ NULL and queue.tid ≠ tid then
7:          queue ← tid.overflow_queue               ▷ Hash collision
8:      queue.tid ← tid
9:      timestamp(pkt)                   ▷ Used by CoDel at dequeue
10:     append(pkt, queue)
11:     if queue is not active then
12:         list_add(queue, tid.new_queues)
```

stead, we innovate on the FQ-CoDel design by having it operate on a fixed total number of queues, and group queues based on which TID they are associated with. So when a packet is hashed and assigned to a queue, that queue is in turn assigned to the TID the packet is destined for. In case that queue is already active and assigned to another TID (which means that a hash collision has occurred), the packet is instead queued to a TID-specific overflow queue.[2] A global queue size limit is kept, and when this is exceeded, packets are dropped from the globally longest queue, which prevents a single flow from locking out other flows on overload. The full enqueue logic is shown in Algorithm 1.

The lists of active queues are kept in a per-TID structure, and when a TID needs to dequeue a packet, the FQ-CoDel scheduler is applied to the TID-specific lists of active queues. This is shown in Algorithm 2.

The obvious way to handle the two other constraints mentioned above (keeping the hardware busy, and handling retries), is, respectively, to add a small queue of preprocessed aggregates, and to add a separate priority queue for packets that need to be retried. And indeed, this is how the ath9k driver already handled these issues, so we simply keep this. The resulting queueing structure is depicted in Figure 3.

3.2 Airtime fairness scheduling

Given the above queueing structure, achieving airtime fairness becomes a matter of measuring the airtime used by each station, and appropriately scheduling the order in which stations are served. For each packet sent or received, the packet duration can either be extracted directly from a hardware register, or it can be calculated from the packet length and the rate at which it was sent (including any retries). Each packet's duration is subtracted

[2] A hash collision can of course also mean that two flows assigned to the same TID end up in the same queue. In this case, no special handling is needed, and the two flows will simply share a queue like in any other hash-based fairness queueing scheme.

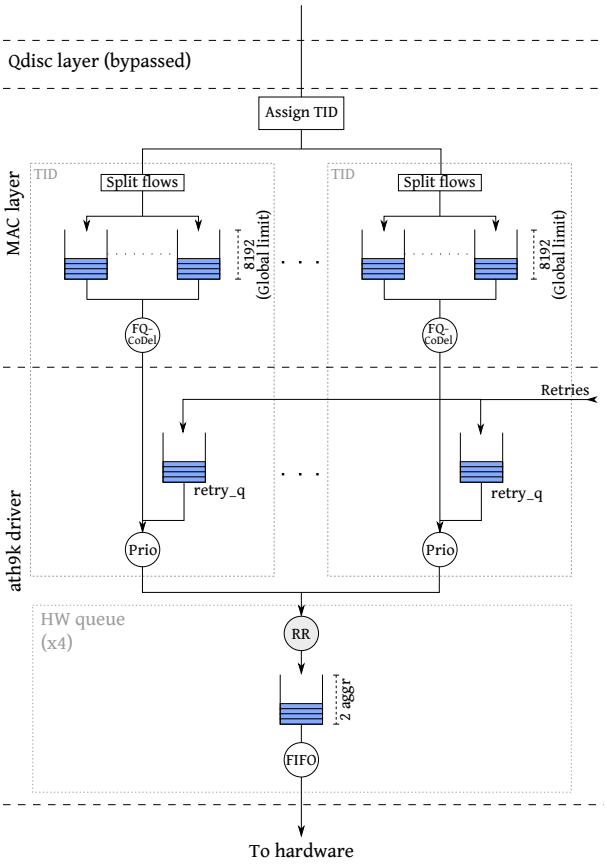


Figure 3: Our 802.11-specific queueing structure, as it looks when applied to the Linux WiFi stack.

from a per-station airtime deficit which is used by a deficit scheduler, modelled after FQ-CoDel, to decide the destination station ahead of each transmission. The decision to keep the deficit per station instead of per TID follows from the fact that the goal of airtime fairness is to even out differences in the physical signal conditions, which is a per-station property. However, because the four 802.11 QoS precedence markings (VO, VI, BE and BK) are commonly scheduled independently down to the hardware level, we actually keep four deficits per station, corresponding to the four precedence levels, to simplify the scheduler implementation.

The resulting airtime fairness scheduler is shown in Algorithm 3. It is similar to the the FQ-CoDel dequeue algorithm, with stations taking the place of flows, and the deficit being accounted in microseconds instead of bytes. The two main differences are (1) that the scheduler function loops until the hardware queue becomes full (at two queued aggregates), rather than just dequeueing a single packet; and (2) that when a station is chosen to be scheduled, it gets to build a full aggregate rather than a single packet.

Compared to the closest previously proposed solu-

Algorithm 2 802.11 queue management algorithm - dequeue.

```
1:  function dequeue(tid)
2:      if tid.new_queues is non-empty then
3:          queue ← list_first(tid.new_queues)
4:      else if tid.old_queues is non-empty then
5:          queue ← list_first(tid.old_queues)
6:      else
7:          return NULL
8:      if queue.deficit ≤ 0 then
9:          queue.deficit ← queue.deficit + quantum
10:         list_move(queue, tid.old_queues)
11:         restart
12:     pkt ← codel_dequeue(queue)
13:     if pkt is NULL then                    ▷ queue empty
14:         if queue ∈ tid.new_queues then
15:             list_move(queue, tid.old_queues)
16:         else
17:             list_del(queue)
18:             queue.tid ← NULL
19:         restart
20:     queue.deficit ← queue.deficit − pkt.length
21:     return pkt
```

Algorithm 3 Airtime fairness scheduler. The schedule function is called on packet arrival and on transmission completion.

```
1:  function schedule
2:      while hardware queue is not full do
3:          if new_stations is non-empty then
4:              station ← list_first(new_stations)
5:          else if old_stations is non-empty then
6:              station ← list_first(old_stations)
7:          else
8:              return
9:          deficit ← station.deficit[pkt.qoslvl]
10:         if deficit ≤ 0 then
11:             station.deficit[pkt.qoslvl] ← deficit + quantum
12:             list_move(station, old_stations)
13:             restart
14:         if station's queue is empty then
15:             if station ∈ new_stations then
16:                 list_move(station, old_stations)
17:             else
18:                 list_del(station)
19:             restart
20:         build_aggregate(station)
```

tion [6], our scheme has several advantages:

1. We lower implementation complexity by leveraging existing information on per-aggregate transmission rates and time, and by using a per-station deficit instead of token buckets, which means that no token bucket accounting needs to be performed at TX and RX completion time.

2. [6] measures time from an aggregate is submitted to the hardware until it is sent, which risks including time spent waiting for other stations to transmit. We increase accuracy by measuring the actual time spent transmitting, and by also accounting the airtime from received frames to each station's deficit.

3. We improve on the basic scheduler design by adding an optimisation for sparse stations, analogous to FQ-CoDel's sparse flow optimisation. This improves latency for stations that only transmit occasionally, by giving them temporary priority for one round of scheduling. We apply the same protection against gaming this mechanism that FQ-CoDel does to its sparse flow mechanism [11].

3.3 Implementation

We have implemented our proposed queueing scheme in the Linux kernel, modifying the mac80211 subsystem to include the queueing structure itself, and modifying the ath9k and ath10k drivers for Qualcomm Atheros 802.11n and 802.11ac chipsets to use the new queueing structure. The airtime fairness scheduler implementation is limited to the ath9k driver, as the ath10k driver lacks the required scheduling hooks.

Our modifications have been accepted into the mainline Linux kernel, different parts going into kernel releases 4.8 through 4.11, and is included in the LEDE open source router firmware from release 17.01. The implementation is available online, as well as details about our test environment and the full evaluation dataset.[3]

4 Evaluation

We evaluate our modifications in a testbed setup consisting of five PCs: Three wireless clients, an access point, and a server located one Gigabit Ethernet hop from the access point, which serves as source and sink for the test flows. All the wireless nodes are regular x86 PCs equipped with PCI-Express Qualcomm Atheros AR9580 adapters (which use the ath9k driver). Two of the test clients are placed in close proximity to the access point (and are referred to as fast nodes), while the last (referred to as the slow node) is placed further away and configured to only support the MCS0 rate, giving a maximum throughput to that station of 7.2 Mbps at the PHY layer. A fourth virtual station is added as an additional fast node to evaluate the sparse station optimisation (see Section 4.1.4 below). All tests are run in HT20 mode on an otherwise unused channel in the 5Ghz band. We use 30 test repetitions of 30 seconds each unless noted otherwise.

The wireless nodes run an unmodified Ubuntu 16.04 distribution. The access point has had its kernel replaced with a version 4.6 kernel from kernel.org on top of which we apply our modifications. We run all experiments with four queue management schemes, as follows:

[3] See `http://www.cs.kau.se/tohojo/airtime-fairness/` for the online appendix that contains additional material, as well as the full experimental dataset and links to the relevant Linux code.

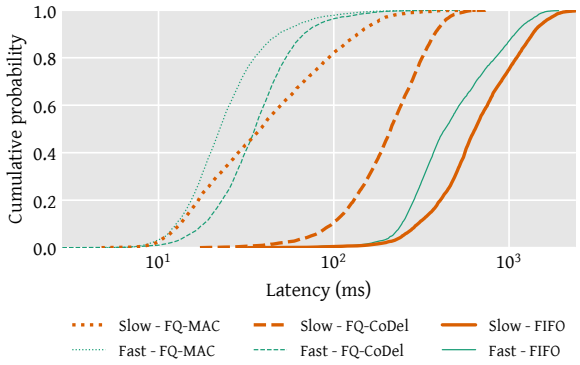


Figure 4: Latency (ICMP ping) with simultaneous TCP download traffic.

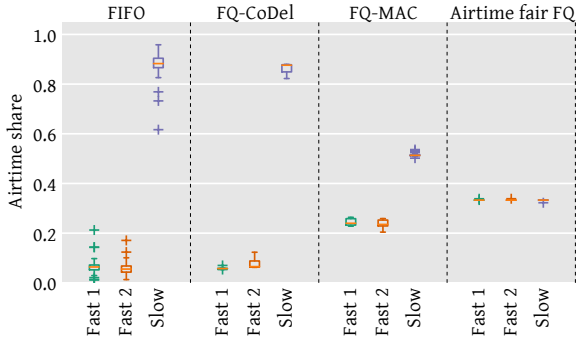


Figure 5: Airtime usage for one-way UDP traffic. Each column shows the relative airtime usage of one of the three stations, with the four sections corresponding to the four queue management schemes.

FIFO: The default 4.6 kernel from kernel.org modified only to collect the airtime used by stations, running with the default PFIFO queueing discipline installed on the wireless interface.

FQ-CoDel: As above, but using the FQ-CoDel qdisc on the wireless interface.

FQ-MAC: Kernel patched to include the FQ-CoDel based intermediate queues in the MAC layer (patching the mac80211 subsystem and the ath9k driver).

Airtime fair FQ: As FQ-MAC, but additionally including our airtime fairness scheduler in the ath9k driver.

Our evaluation is split into two parts. First, we validate the effects of the modifications in simple scenarios using synthetic benchmark traffic. Second, we evaluate the effect of our modifications on two application traffic scenarios, to verify that they provide a real-world benefit.

4.1 Validation of effects

In this section we present the evaluation of our modifications in simple synthetic scenarios designed to validate the correct functioning of the algorithms and to demonstrate various aspects of their performance.

4.1.1 Latency reductions

Figure 4 is the full set of results for our ICMP latency measurements with simultaneous TCP download traffic (of which a subset was shown earlier in Figure 2). Here, the FIFO case shows several hundred milliseconds of latency when the link is saturated by a TCP download. FQ-CoDel alleviates this somewhat, but the slow station still sees latencies of more than 200 ms in the median, and the fast stations around 35 ms. With the FQ-MAC queue restructuring, this is reduced so that the slow station now has the same median latency as the fast one does in the FQ-CoDel case, while the fast stations get their latency reduced by another 45%. The airtime scheduler doesn't improve further upon this, other than to alter the shape of the distribution slightly for the slow station (but retaining the same median). For this reason, we have omitted it from the figure to make it more readable.

For simultaneous upload and download the effect is similar, except that in this case the airtime scheduler slightly worsens the latency to the slow station, because it is scheduled less often to compensate for its increased airtime usage in the upstream direction. The graph of this case can be found in the online appendix.

4.1.2 Airtime usage

Figure 5 shows the airtime usage of the three active stations for one-way UDP traffic going to the stations. There is no reverse traffic and no contention between stations, since only the access point is transmitting data. This is the simplest case to reason about and measure, and it clearly shows the performance anomaly is present in the current Linux kernel (left half of the figure): The third station (which transmits at the lowest rate) takes up around 80% of the available airtime, while the two other stations share the remaining 20%.

The differences between the first two columns and the third column are due to changes in aggregation caused by the change to the queueing structure. In the FIFO and FQ-CoDel cases, there is a single FIFO queue with no mechanism to ensure fair sharing of that queue space between stations. So because the slow station has a lower egress rate, it will build more queue until it takes up the entire queueing space. This means that there are not enough packets queued to build sufficiently large aggregates for the fast stations to use the airtime effectively. The FQ-MAC queueing scheme drops packets from the largest queue on overflow, which ensures that the available queueing space is shared between stations, which improves aggregation for the fast stations and thus changes

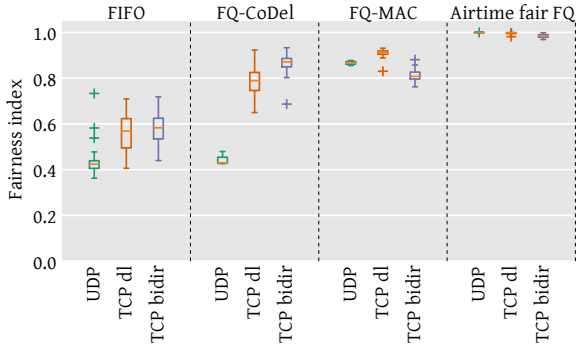


Figure 6: Jain's fairness index (computed over the airtime usage of the three stations) for UDP traffic, TCP download, and simultaneous TCP upload and download traffic.

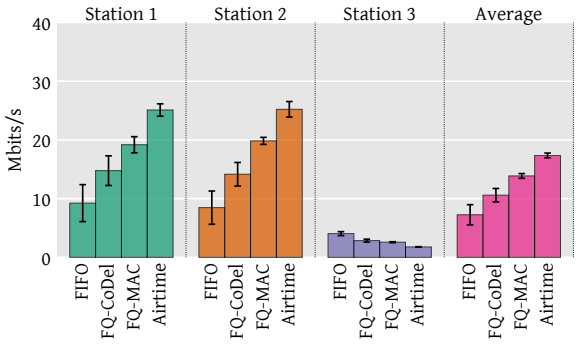


Figure 7: Throughput for TCP download traffic (to clients).

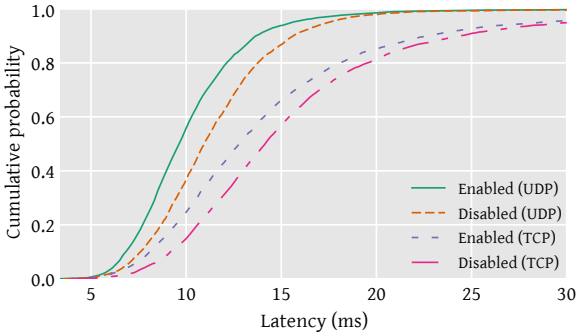


Figure 8: The effects of the sparse station optimisation.

the airtime shares. Referring back to Table 1, the values correspond well to those predicted by the analytical model. The fourth column shows the airtime fairness scheduler operating correctly – each station receives exactly the same amount of airtime in this simple one-way test.

Going beyond the simple UDP case, Figure 6 shows Jain's fairness index for the airtime of the four different schemes for UDP (for comparison) and both unidirectional (to the clients) and bidirectional (simultaneous up and down) TCP traffic. The same general pattern is seen with TCP as with UDP traffic: The performance anomaly is clear for the FIFO case, but somewhat lessened for the FQ-CoDel and FQ-MAC cases. The airtime fairness scheduler achieves close to perfect sharing of airtime in the case of uni-directional traffic, with a slight dip for bidirectional traffic. The latter is because the scheduler only exerts indirect control over the traffic sent from the clients, and so cannot enforce perfect fairness as with the other traffic types. However, because airtime is also accounted for received packets, the scheduler can partially compensate, which is why the difference between the unidirectional and bidirectional cases is not larger than it is.

4.1.3 Effects on throughput

As was already shown in Table 1, fixing the performance anomaly improves the efficiency of the network for unidirectional UDP traffic. Figure 7 shows the throughput for downstream TCP traffic. For this case, the fast stations increase their throughput as fairness goes up, and the slow station decreases its throughput. The total effect is a net increase in throughput. The increase from the FIFO case to FQ-CoDel and FQ-MAC is due to better aggregation for the fast stations. This was observed for UDP as well in the case of FQ-MAC, but for FQ-CoDel the slow station would occupy all the queue space in the driver, preventing the fast station from achieving full aggregation. With the TCP feedback loop in place, this lock-out behaviour is lessened, and so fast stations increase their throughput.

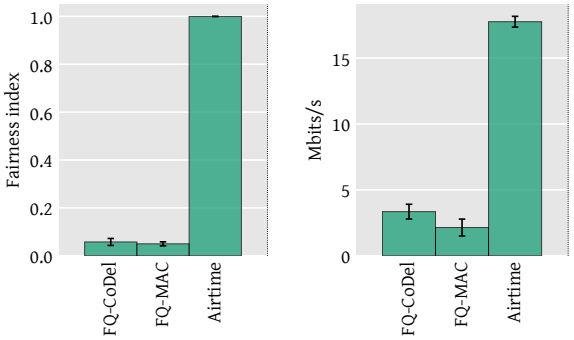
When traffic is flowing in both directions simultaneously, the pattern is similar, but with a slightly higher variance. The graph for the bidirectional case can be found in the online appendix.

4.1.4 The sparse station optimisation

To evaluate the impact of the sparse station optimisation, we add a fourth station to our experiments which receives only a ping flow, but no other traffic, while the other stations receive bulk traffic as above. We measure the latency to this extra station both with and without the sparse station optimisation. The results of this are shown in Figure 8. For both UDP and TCP download traffic, the optimisation achieves a small, but consistent, improvement: The round-trip latency to the fourth station is reduced by 10 to 15% (in the median) when the optimisation is in place.

4.1.5 Scaling to more stations

While the evaluations presented in the previous sections have shown that our modifications work as planned, and that they provide a substantial benefit in a variety of scenarios, one question is left unanswered – does the solution scale to more stations? To answer this, we arranged for



(a) Airtime usage fairness **(b)** Aggregate throughput

Figure 9: Aggregate results for the 30 stations TCP test.

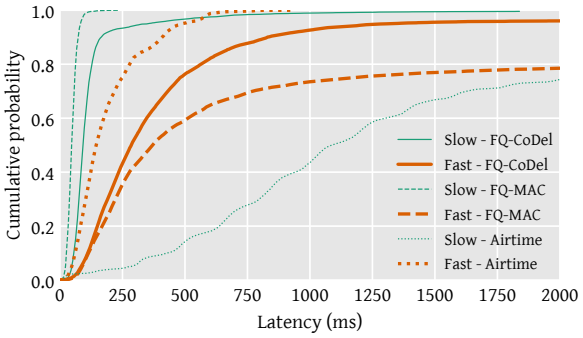


Figure 10: Latency for the 30 stations TCP test.

an independent third party to repeat a subset of our tests in their testbed, which features an access point and 30 clients. The nodes are all embedded wireless devices from a commercial vendor that bases its products on the OpenWrt/LEDE open-source router platform, running a LEDE firmware development snapshot from November 2016.

In this setup, one of the 30 clients is artificially limited to only transmit at the lowest possible rate (1 Mbps, i.e. disabling HT mode), while the others are configured to select their rate in the usual way, on a HT20 channel in the 2.4 Ghz band. One of the 29 "fast" clients only receives ping traffic, leaving 28 stations to contend with the slow 1 Mbps station for airtime and bandwidth.

In this environment, our downstream TCP experiment presented above was repeated, with the difference that each test was run for five minutes, but with only five repetitions, and without the FIFO test case. A subset of these results are shown in figures 9 and 10. From this experiment, we make several observations:

1. When the slow station is at this very low rate, it manages to grab around two thirds of the available airtime, even with 28 other stations to compete with. However, our airtime fairness scheduler manages to achieve completely fair sharing of airtime between all 29 stations. This is reflected in the fairness index as seen in Figure 9a.

2. As seen in Figure 9b, total throughput goes from a mean of 3.3 Mbps for the FQ-CoDel case to 17.7 Mbps with the airtime scheduler. That is, the relative throughput gain with airtime fairness is 5.4x in this scenario.

3. As can be expected, with the airtime fairness scheduler, the latency to the fast stations is improved with the increased throughput (Figure 10, green lines). However, the latency to the slow station increases by an order of magnitude in the median, as it is throttled to stay within its fair share of the airtime (Figure 10, dashed orange line). Overall, the *average* latency to all stations is improved by a factor of two (not shown on the figure).

4. With 30 stations, we see the sparse station optimisation being even more effective; in this scenario it reduces latency to the sparse station by a factor of two (not shown in the figures; see the online appendix).

Finally, we verify the in-kernel airtime measurement against a tool developed by the same third party that measures airtime from captures taken with a monitor device. We find that the two types of measurements agree to within 1.5%, on average.

4.2 Effects on real-world application performance

In the previous section we evaluated our solution in a number of scenarios that verify its correct functioning and quantify its benefits. In this section we expand on that validation by examining how our modifications affect performance of two important real-world applications – VoIP and web browsing.

4.2.1 VoIP

VoIP is an important latency-sensitive application which it is desirable to have working well over WiFi, since that gives mobile handsets the flexibility of switching between WiFi and cellular data as signal conditions change. To evaluate our modifications in the context of VoIP traffic, we measure VoIP performance when mixed with bulk traffic. As in Section 4.1.4 we include a virtual station as another fast station, and so these scenarios have three fast stations. Due to space constraints, we only include the case where the slow station receives both VoIP traffic *and* bulk traffic, while the fast stations receive bulk traffic. The other cases show similar relative performance between the different queue management schemes.

The QoS markings specified in the 802.11e standard can be used to improve the performance of VoIP traffic, and so we include this aspect in our evaluation. 802.11e specifies four different QoS levels, of which voice (VO) has the highest priority. Packets transmitted with this QoS marking gets both queueing priority and a shorter contention window, but cannot be aggregated. This difference

		5 ms		50 ms	
	QoS	MOS	Thrp	MOS	Thrp
FIFO	VO	4.17	27.5	4.13	21.6
	BE	1.00	28.3	1.00	22.0
FQ-CoDel	VO	4.17	25.5	4.08	15.2
	BE	1.24	23.6	1.21	15.1
FQ-MAC	VO	4.41	39.1	4.38	28.5
	BE	4.39	43.8	4.37	34.0
Airtime	VO	4.41	56.3	4.38	49.8
	BE	4.39	57.0	4.37	49.7

Table 2: MOS values and total throughput when using different QoS markings for VoIP traffic. Data for 5 ms and 50 ms baseline one-way delay.

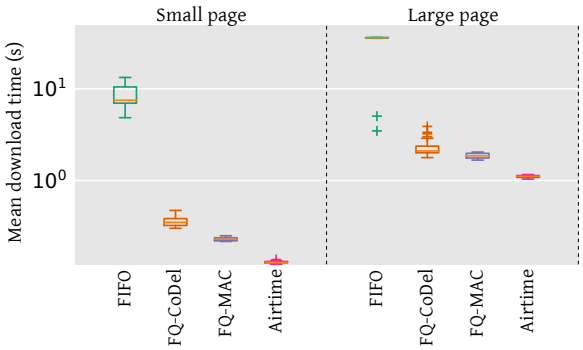


Figure 11: HTTP page fetch times for a fast station (while the slow station runs a bulk transfer). Note the log scale - the fetch time for the large page is 35 *seconds* for the FIFO case.

can dramatically reduce the latency of the traffic, at a cost in throughput because of the lack of aggregation. We repeat the voice experiments in two variants – one where the VoIP packets are sent as best effort (BE) traffic, and one where they are put into the high-priority VO queue. We also repeat the tests with a baseline one-way delay of both 5 ms and 50 ms.

To serve as a metric of voice quality, we calculate an estimate of the Mean Opinion Score (MOS) of the VoIP flow according to the E-model specified in the ITU-T G.107 recommendation [27]. This model can predict the MOS from a range of parameters, including the network conditions. We fix all audio and codec related parameters to their default values and calculate the MOS estimate based on the measured delay, jitter and packet loss. The model gives MOS values in the range from $1 - 4.5$.

Table 2 shows the results. This shows that throughput follows the trends shown in previous tests, as expected. Also as expected, the FIFO and FQ-CoDel cases have low MOS values when the voice traffic is marked as BE, and higher values when using the VO queue. However, both the FQ-MAC and airtime fairness schemes achieve *better* MOS values with best-effort traffic than the unmodified kernel does with VO-marked traffic. In the FQ-MAC and airtime cases, using the VO queue still gives a slightly better MOS score than using the BE queue does; but the difference is less than half a percent. This is an important improvement, because it means that with our modifications, applications can rely on excellent real-time performance even when unable to control DiffServ markings, as well as when the markings are removed in transit.

4.2.2 Web

Another important real-world application is web traffic. To investigate the impact of our modifications on this, we measure page load time (PLT) with emulated web traffic. Our test client mimics the common web browser behaviour of fetching multiple requests in parallel over four different TCP connections. We simply measure the total time to fetch a web site and all its attached resources (including the initial DNS lookup) for two different pages – a small page (56 KB data in three requests) and a large page (3 MB data in 110 requests). We run the experiments in two scenarios. One where a fast station fetches the web sites while the slow station runs a simultaneous bulk transfer, to emulate the impact of a slow station on the browsing performance of other users on the network. And another scenario where the slow station fetches the web sites while the fast stations run simultaneous bulk transfers, to emulate the browsing performance of a slow station on a busy network.

The results for the fast station are seen in Figure 11. Fetch times decrease from the FIFO case as the slowest to the airtime fair FQ case as the fastest. In particular, there is a an order-of-magnitude improvement when going from FIFO to FQ-CoDel, which we attribute to the corresponding significant reduction in latency seen earlier.

When the slow station is fetching the web page, adding airtime fairness increases page load time by $5 - 10\%$. This is as expected since in this case the slow station is being throttled. The graph for this can be found in the online appendix.

4.3 Summary

Our evaluation shows that our modifications achieve their design goal. We eliminate bufferbloat and the 802.11 performance anomaly, and achieve close to perfect airtime fairness even when station rates vary; and our solution scales successfully as more clients are added. We improve total throughput by up to a factor of five and reduce latency under load by up to an order of magnitude. We also achieve close to perfect airtime fairness in a scenario

where traffic is mixed between upstream and downstream flows from the different stations. Finally, the optimisation that prioritises sparse stations achieves a consistent improvement in latency to stations that only receive a small amount of traffic.

In addition, we show that our modifications give significant performance increases for two important real-world applications – VoIP and web traffic. In the case of VoIP, we manage to achieve better performance with best effort traffic than was achievable with traffic marked as Voice according to the 802.11e QoS standard. For web traffic we achieve significant reductions in page load times.

Finally, even though our evaluation scenario only features a limited number of stations, we have sought to represent a challenging scenario, by having high congestion rates and a large difference between the station rates. As such, we believe that it serves well as a validation of the effects. In addition, the feedback we have received from users of the code indicates that our solution works well in a variety of deployments.

5 Related work

There have been several previous studies on bufferbloat and its mitigations (e.g. [15, 29]), but only a few that deal with the problem in a WiFi-specific context. [10] and [15] both feature a WiFi component in a larger evaluation of bufferbloat mitigation techniques and show that while these techniques can help on a WiFi link, the lower-level queueing in the WiFi stack prevents a full solution of the problem in this space. [23] draws similar conclusions while looking at buffer sizing (but only mentions AQM-based solutions briefly). Finally, [4] touches upon congestion at the WiFi hop and uses different queueing schemes to address it, but in the context of a centralised solution that also seek to control fairness in the whole network. None of these works actually provide a solution for bufferbloat at the WiFi link itself, as we present here.

Several different schemes to achieve fairness based on modifying the contention behaviour of nodes are presented in [8, 12, 13, 19, 22, 30]. [12] and [19] both propose schemes that use either the 802.11e TXOP feature to allocate equal-length to all stations, or scaling of the contention window by the inverse of the transmission rate to achieve fairness. [13] develops an analytical model to predict the values to use for a similar scaling behaviour, which is also verified in simulation. [22] presents a modified contention behaviour that can lower the number of collisions experienced, but they do not verify the effect of their schemes on airtime fairness. [8] proposes a modification to the DCF based on sensing the idle time of the channel scaling CW_{min} with the rate to achieve fairness. Finally, [30] proposes a scheme for airtime fairness that runs several virtual DCF instances per node, scaling the number of instances with the rate and channel properties.

Achieving fairness by varying the transmission size is addressed in [5, 16, 20]. The former two predate the aggregation features of 802.11n and so [5] proposes to scale the packet size downwards by varying the MTU with the transmission rate. [20] goes the other way and proposes a scheme where a station will burst packets to match the total transmission length of the previous station that was heard on the network. Finally, [16] uses the two-level aggregation feature of 802.11n and proposes a scheme to dynamically select the optimal aggregation size so all transmissions take up the same amount of time.

Turning to schedulers, [7] and [6] both propose schedulers which work at the access point to achieve airtime fairness, the former estimating the packet transmission time from channel characteristics, and the latter measuring it after transmission has occurred. [21] proposes a solution for wireless mesh networks, which leverages routing metrics to schedule links in a way that ensures fairness. Finally, [17] proposes a scheme to scale the queue space at the access point based on the BDP of the flows going through the access point. Our solution is closest to [6], but we improve upon it by increasing accuracy and reducing implementation difficulty, while adding an important latency-reducing optimisation for sparse stations, as was described in Section 3.2.

A few proposals fall outside the categories above. [14] proposes a TCP congestion control algorithm that uses information about the wireless conditions to cap the TCP window size of clients to achieve fairness. Finally, there are schemes that sidestep the fairness problems of the 802.11 MAC and instead replace it entirely with TDMA scheduling. [3] proposes a scheme for TDMA scheduling in a mesh network that ensures fair bandwidth allocation to all connecting clients, and [28] implements a TDMA transmission scheme for infrastructure WiFi networks.

6 Conclusion

We have developed a novel two-part solution to two large performance problems affecting WiFi – bufferbloat and the 802.11 performance anomaly. The solution consists of a new integrated queueing scheme tailored specifically to eliminate bufferbloat in WiFi, which reduces latency under load by an order of magnitude. Leveraging the queueing structure, we have developed a deficit-based airtime fairness scheduler that works at the access point with no client modifications, and achieves close to perfect fairness in all the evaluated scenarios, increasing total throughput by up to a factor of 5.

Our solution reduces implementation complexity and increases accuracy compared to previous work, and has been accepted into the mainline Linux kernel, making it deployable on billions of Linux-based devices.

7 Acknowledgements

We would like to thank Sven Eckelmann and Simon Wunderlich for their work on independently verifying our implementation. Their work was funded by Open Mesh Inc, who also supplied their test hardware. We would also like to thank Felix Fietkau, Tim Shepard, Eric Dumazet, Johannes Berg, and the numerous other contributors to the Make-Wifi-Fast and LEDE projects for their insights, review and contributions to many different iterations of the implementation.

Portions of this work were funded by Google Fiber and by the Comcast Innovation Fund, and parts of the infrastructure was sponsored by Lupin Lodge.

8 References

[1] Adam Belay et al. 'IX: A Protected Dataplane Operating System for High Throughput and Low Latency'. In: *OSDI*. 11th USENIX Symposium on Operating Systems Design and Implementation. Oct. 2014, pp. 49–65.

[2] Teuku Yuliar Arif and Riri Fitri Sari. 'Throughput Estimates for A-MPDU and Block ACK Schemes Using HT-PHY Layer'. In: *Journal of Computers* 9.3 (Mar. 2014). doi: 10.4304/jcp.9.3.678-687.

[3] Naouel Ben Salem and Jean-Pierre Hubaux. 'A fair scheduling for wireless mesh networks'. In: *WiMesh*. 2005.

[4] Kan Cai et al. 'Wireless Unfairness: Alleviate MAC Congestion First!' In: *Proceedings of the Second ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization*. WinTECH '07. New York, NY, USA: ACM, 2007, pp. 43–50. doi: 10.1145/1287767.1287777.

[5] Joseph Dunn et al. 'A practical cross-layer mechanism for fairness in 802.11 networks'. In: *Broadband Networks, 2004. BroadNets 2004. Proceedings. First International Conference on*. IEEE, 2004, pp. 355–364.

[6] Rosario G. Garroppo et al. 'Providing air-time usage fairness in IEEE 802.11 networks with the deficit transmission time (DTT) scheduler'. In: *Wireless Networks* 13.4 (Aug. 2007), pp. 481–495. doi: 10.1007/s11276-006-9201-7.

[7] K. Gomez et al. 'On efficient airtime-based fair link scheduling in IEEE 802.11-based wireless networks'. In: *2011 IEEE 22nd International Symposium on Personal, Indoor and Mobile Radio Communications*. Sept. 2011, pp. 930–934. doi: 10.1109/PIMRC.2011.6140105.

[8] Martin Heusse et al. 'Idle Sense: An Optimal Access Method for High Throughput and Fairness in Rate Diverse Wireless LANs'. In: *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM '05. New York, NY, USA: ACM, 2005, pp. 121–132. doi: 10.1145/1080091.1080107.

[9] Martin Heusse et al. 'Performance anomaly of 802.11 b'. In: *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*. Vol. 2. IEEE, 2003, pp. 836–843.

[10] Toke Høiland-Jørgensen, Per Hurtig, and Anna Brunstrom. 'The Good, the Bad and the WiFi: Modern AQMs in a residential setting'. In: *Computer Networks* 89 (Oct. 2015), pp. 90–106. doi: 10.1016/j.comnet.2015.07.014.

[11] T. Høiland-Jørgensen et al. *FlowQueue-Codel*. Internet Draft (informational). Mar. 2016.

[12] Li Bin Jiang and Soung Chang Liew. 'Proportional fairness in wireless LANs and ad hoc networks'. In: *Wireless Communications and Networking Conference, 2005 IEEE*. Vol. 3. IEEE, 2005, pp. 1551–1556.

[13] T. Joshi et al. 'Airtime Fairness for IEEE 802.11 Multirate Networks'. In: *IEEE Transactions on Mobile Computing* 7.4 (Apr. 2008), pp. 513–527. doi: 10.1109/TMC.2007.70740.

[14] K. Kashibuchi, A. Jamalipour, and N. Kato. 'Channel Occupancy Time Based TCP Rate Control for Improving Fairness in IEEE 802.11 DCF'. In: *IEEE Transactions on Vehicular Technology* 59.6 (July 2010), pp. 2974–2985. doi: 10.1109/TVT.2010.2048931.

[15] Naeem Khademi, David Ros, and Michael Welzl. 'The New AQM Kids on the Block: Much Ado About Nothing?' In: *Technical Report, Oslo University* (2013).

[16] Minho Kim, Eun-Chan Park, and Chong-Ho Choi. 'Adaptive Two-Level Frame Aggregation for Fairness and Efficiency in IEEE 802.11n Wireless LANs'. In: *Mobile Information Systems* 2015 (2015), pp. 1–14. doi: 10.1155/2015/548109.

[17] Dzmitry Kliazovich et al. 'Queue Management Mechanism for 802.11 Base Stations'. In: *IEEE Communications Letters* 15.7 (July 2011), pp. 728–730. doi: 10.1109/LCOMM.2011.051911.110642.

[18] M. Laddomada et al. ‘On the throughput performance of multirate IEEE 802.11 networks with variable-loaded stations: analysis, modeling, and a novel proportional fairness criterion’. In: *IEEE Transactions on Wireless Communications* 9.5 (May 2010), pp. 1594–1607. doi: 10.1109/TWC.2010.05.081191.

[19] Pochiang Lin, Wu-I. Chou, and Tsungnan Lin. ‘Achieving airtime fairness of delay-sensitive applications in multirate IEEE 802.11 wireless LANs’. In: *Communications Magazine, IEEE* 49.9 (2011), pp. 169–175.

[20] Tahiry Razafindralambo et al. ‘Dynamic packet aggregation to solve performance anomaly in 802.11 wireless networks’. In: *Proceedings of the 9th ACM international symposium on Modeling analysis and simulation of wireless and mobile systems*. ACM, 2006, pp. 247–254.

[21] R. Riggio, D. Miorandi, and I. Chlamtac. ‘Airtime Deficit Round Robin (ADRR) packet scheduling algorithm’. In: *2008 5th IEEE International Conference on Mobile Ad Hoc and Sensor Systems*. Sept. 2008, pp. 647–652. doi: 10.1109/MAHSS.2008.4660101.

[22] Luis Sanabria-Russo et al. ‘Future evolution of CSMA protocols for the IEEE 802.11 standard’. In: *Communications Workshops (ICC), 2013 IEEE International Conference on*. IEEE, 2013, pp. 1274–1279.

[23] A. Showail, K. Jamshaid, and B. Shihada. ‘Buffer sizing in wireless networks: challenges, solutions, and opportunities’. In: *IEEE Communications Magazine* 54.4 (Apr. 2016), pp. 130–137. doi: 10.1109/MCOM.2016.7452277.

[24] M. Shreedhar and G. Varghese. ‘Efficient fair queuing using deficit round-robin’. In: *IEEE/ACM Transactions on Networking* 4.3 (June 1996), pp. 375–385. doi: 10.1109/90.502236.

[25] T. Szigeti and F. Baker. *DiffServ to IEEE 802.11 Mapping*. Internet Draft (standards track). Nov. 2016.

[26] Godfrey Tan and John V. Guttag. ‘Time-based Fairness Improves Performance in Multi-Rate WLANs.’ In: *USENIX Annual Technical Conference, General Track*. 2004, pp. 269–282.

[27] *The E-model: a computational model for use in transmission planning*. Tech. rep. G.107. ITU-T, June 2015.

[28] Wim Torfs and Chris Blondia. ‘TDMA on commercial of-the-shelf hardware: Fact and fiction revealed’. In: *AEU-International Journal of Electronics and Communications* 69.5 (2015), pp. 800–813.

[29] Greg White. *Active Queue Management Algorithms for DOCSIS 3.0: A Simulation Study of CoDel, SFQ-CoDel and PIE in DOCSIS 3.0 Networks*. Tech. rep. Cable Television Laboratories, Inc., 2013.

[30] M. A. Yazici and N. Akar. ‘Running Multiple Instances of the Distributed Coordination Function for Air-Time Fairness in Multi-Rate WLANs’. In: *IEEE Transactions on Communications* 61.12 (Dec. 2013), pp. 5067–5076. doi: 10.1109/TCOMM.2013.111113.130120.