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)

USENIX ATC 17, July 12-14, Santa Clara, CA
OUTLINE

• The problem: Bufferbloat (and airtime fairness)
• Our solution: Queueing structure (and scheduler)
• Evaluation results
• Summary and questions
Bufferbloat is **pathological and persistent** queueing latency.

Solutions exist to **eliminate bufferbloat** in wired networks:

- The FQ-CoDel hybrid AQM/fairness queueing algorithm
- PIE and CoDel AQMs
- Others appearing

However, on WiFi there we still see **100s of milliseconds** of extra buffering in the stack.
WHAT DID WE ACHIEVE?

- WiFi bufferbloat reduced by an order of magnitude
- (Almost perfect airtime fairness in most cases)
- Light-weight deployable solution, accepted in mainline Linux
CONSTRAINTS

1. We must handle aggregation per Traffic ID (TID)
2. We must handle re-injection of packets for retransmission
3. We must be able to keep the hardware busy
4. We must support low-power access points (down to 32MB of RAM)
5. We cannot modify clients

1 & 2 means we can't use solutions for wired networks as-is.
OUR SOLUTION

We design a queueing structure (and an airtime fairness scheduler).

QUEUEING STRUCTURE

- Per-flow queueing based on FQ-CoDel
- Shared pool of queues to avoid memory explosion
- Supports per-TID dequeueing and scheduling

AIRTIME SCHEDULER

- Measure actual airtime usage of each station
- Run a DRR-based scheduler to even them out
- Optimise for sparse stations
Linux kernel queueing structure before and after our modifications.

Per HW queue (x4)  
Qdisc layer  
MAC layer  
ath9k driver

*Can be replaced with an arbitrary configuration

FIFO  
buf_q  
retry_q  
Retries

TID  
buf_q  
retry_q  
RR  
123  
Prio  

TID  
buf_q  
retry_q  
RR  
123  
Prio  

Qdisc layer (bypassed)  
MAC layer  
HW queue (x4)  
ath9k driver

Assign TID  
Split flows  
Retries

TID  
Split flows

TID  
Split flows

FQ-CoDel  
Split flows  
Global limit

FQ-CoDel  
Split flows  
Global limit

To hardware

Toke Høiland-Jørgensen <toke.hoiland-jorgensen@kau.se>
EVALUATION

Four scenarios:

- **FIFO**: Default before modifications
- **FQ-CoDel**: FQ-CoDel qdisc on WiFi interface
- **FQ-MAC**: Our restructured MAC layer queues
- **Airtime fairness**: FQ-MAC + airtime fairness scheduler
APPLICATION IMPACT

We evaluate:

- HTTP page load time performance
- VoIP performance (MOS values)
HTTP PAGE LOAD TIMES

Mean download time (s)

Small page

10^1

FIFO
FQ-CoDel
FQ-MAC
Airtime

10^0

Large page

FIFO
FQ-CoDel
FQ-MAC
Airtime
### VOIP TEST

<table>
<thead>
<tr>
<th></th>
<th>QoS</th>
<th>MOS</th>
<th>Thrp</th>
<th>MOS</th>
<th>Thrp</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FIFO</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VO</td>
<td>4.17</td>
<td>27.5</td>
<td>4.13</td>
<td>21.6</td>
<td></td>
</tr>
<tr>
<td>BE</td>
<td>1.00</td>
<td>28.3</td>
<td>1.00</td>
<td>22.0</td>
<td></td>
</tr>
<tr>
<td><strong>FQ-CoDel</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VO</td>
<td>4.17</td>
<td>25.5</td>
<td>4.08</td>
<td>15.2</td>
<td></td>
</tr>
<tr>
<td>BE</td>
<td>1.24</td>
<td>23.6</td>
<td>1.21</td>
<td>15.1</td>
<td></td>
</tr>
<tr>
<td><strong>FQ-MAQ</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VO</td>
<td>4.41</td>
<td>39.1</td>
<td>4.38</td>
<td>28.5</td>
<td></td>
</tr>
<tr>
<td>BE</td>
<td>4.39</td>
<td>43.8</td>
<td>4.37</td>
<td>34.0</td>
<td></td>
</tr>
<tr>
<td><strong>Airtime</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VO</td>
<td>4.41</td>
<td>56.3</td>
<td>4.38</td>
<td>49.8</td>
<td></td>
</tr>
<tr>
<td>BE</td>
<td>4.39</td>
<td>57.0</td>
<td>4.37</td>
<td>49.7</td>
<td></td>
</tr>
</tbody>
</table>

Synthetic MOS values calculated from the **ITU-T G.107 E-model**.
SUMMARY

We have:
- Reduced WiFi bufferbloat by an order of magnitude
- Achieved almost perfect airtime fairness in most cases
- Created a light-weight, deployable solution, accepted in mainline Linux

Code, data and more graphs available at:
https://www.cs.kau.se/tohojo/airtime-fairness/

Many thanks to Sven Eckelmann, Simon Wunderlich, Felix Fietkau, Tim Shepard, Eric Dumazet, Johannes Berg, and the numerous other contributors to the Make-Wifi-Fast and LEDE projects.
Effective transmission time $T(i)$ and rate $R(i)$ (for station $i \in I$):

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

$$R(i) = T(i)R_0(i)$$

Where $R_0(i) = \frac{L_i}{T_{data}(i)+T_{oh}}$ is the effective rate of a station transmitting without collisions.

Network throughput is determined by the slowest station.
function on_tx(pkt) {
    station = get_station(pkt)
    station.deficit -= pkt.duration
}

function on_rx(pkt) {
    station = get_station(pkt)
    station.deficit -= calc_dur(pkt)
}

function schedule(hwq) {
    if full(hwq) { return }

    begin:
    station = list_head(station_list)

    if station.deficit <= 0 {
        station.deficit += quantum
        list_move_end(station, station_list)
        goto begin
    }
    if !station.queue {
        list_del(station)
        goto begin
    }
    queue_aggregate(station)
}
AIRTIME FAIRNESS

![Graph showing fairness index for different protocols and data types](image-url)
SPARSE STATION OPTIMISATION

![Graph showing cumulative probability vs. latency for different conditions: Enabled (UDP), Disabled (UDP), Enabled (TCP), Disabled (TCP).]
30 STATIONS TEST

- We cooperated with another lab to evaluate our solution
- 30 station testbed, one slow station (1 Mbps)
30 STATIONS - LATENCY

Cumulative probability

Latency (ms)

Slow - FQ-CoDel
Fast - FQ-CoDel
Slow - FQ-MAC
Fast - FQ-MAC
Slow - Airtime
Fast - Airtime