# Analysing the Latency of Sparse Flows in the FQ-CoDel Queue Management Algorithm

Toke Høiland-Jørgensen, *Student Member, IEEE*

***Abstract*—The FQ-CoDel queue management algorithm was recently published as an IETF RFC. It achieves low latency especially for low-volume (or *sparse*) traffic flows competing with bulk flows. However, the exact conditions for when a particular flow is considered to be sparse has not been well-explored.**

**In this work, we analyse the performance characteristics of the sparse flow optimisation of FQ-CoDel, formulating the constraints that flows must satisfy to be considered sparse in a given scenario. We also formulate expressions for the expected queueing latency for sparse flows.**

**Then, using a numerical example, we show that for a given link and a given type of sparse flows (VoIP traffic), the number of sparse flows that a given bottleneck can service with low sparse flow latency is only dependent on the number of backlogged bulk flows at the bottleneck. Furthermore, as long as the maximum number of sparse flows is not exceeded, all sparse flows can expect a very low queueing latency through the bottleneck.**

***Index Terms*—FQ-CoDel, Queueing Latency, Bufferbloat**

## I. Introduction

THE FQ-CoDel queue management algorithm, which was recently published as an IETF RFC [1], is a hybrid AQM and packet scheduling algorithm that has been shown to be an excellent remedy for the bufferbloat problem of excessive queueing on a congested link [2]. In particular, FQ-CoDel achieves very low latency for low-volume traffic competing with the bulk flows causing the congestion. This is due to the *sparse flow optimisation* employed in the flow scheduler.

However, while FQ-CoDel has been shown to achieve very low latency for such sparse flows, the exact conditions for when a particular flow is considered to be sparse has not been well-explored, as noted in the RFC [1, Section 1.3].

The contribution of this work is an analysis of what exactly constitutes a sparse flow in FQ-CoDel. We achieve this by formulating analytical expressions for the constraints flows must satisfy to be treated as sparse by the FQ-CoDel scheduler, and supplement with a numeric example and simulation for a typical example of real-world traffic (real-time VoIP traffic).

The rest of this paper is structured as follows: Section II first summarises related work and Section III explains how the sparse flow optimisation in FQ-CoDel works. Section IV then presents our analytical framework and results and Section V shows the real-world examples. Finally, Section VI concludes.

## II. Related work

While several studies have measured the performance of FQ-CoDel (e.g., [2]–[4]), none deal specifically with the sparse flow optimisation, and none offer any analytical expressions for the performance of the algorithm. However, similar algorithms have been subject to analysis, as summarised below.

FQ-CoDel is based on the deficit round-robin (DRR) scheduler [5]. The authors of DRR propose an extension called DRR+ where "latency-sensitive flows" are given priority as long as such a flow never sends more than $x$ bytes every time period $T$, which can be said to be an a priori analytical expression for the constraints of a sparse flow. This mechanism is expanded upon in the DRR++ [6] algorithm, which adds an extension to the mechanism to better deal with bursty latency-sensitive flows. The scheduling of DRR++, in particular, is identical to that of FQ-CoDel, except that DRR++ requires flows to be explicitly classified as latency-sensitive (without specifying any mechanism to do so), whereas FQ-CoDel applies the same scheduling to all flows, which means that latency-sensitive flows are only classified implicitly. However, since the authors of DRR++ assume an a priori classification of latency-sensitive flows, there is no analysis of their constraints.

The implicit classification mechanism of FQ-CoDel is similar to that used by the Shortest Queue First (SQF) queueing scheme [7], which works by simply dequeueing packets from the shortest queue available at the time of dequeue. This gives implicit priority to flows that do not build a queue, such as voice flows and low-bandwidth video streams. However, since SQF does not use a round-robin scheduler, it gives no service guarantees to the backlogged bulk flows. The authors provide both analytical and experimental evaluations of the algorithm performance characteristics in [8].

The Quick Fair Queueing (QFQ) algorithm [9] is an O(1) scheduling algorithm that implements fairness queueing between flows in a way that approximates a fluid model of the flows with high accuracy. The paper provides an extensive analysis of its performance characteristics.

Finally, a comprehensive analysis of the number of active flows in a fairness queueing system is provided in [10]. This does not treat queueing latency, nor does it distinguish between types of traffic, such as sparse or bulk flows.

## III. The sparse flow optimisation

The FQ-CoDel sparse flow optimisation works as follows: When a packet arrives at the router, it is hashed on its transport layer 5-tuple (source and destination IP, IP protocol number and source and destination ports). The result of the hash, modulo the number of configured queues, is the queue number of that packet, and the packet is enqueued to that queue. If this queue is already active, no further action is taken.

Toke Høiland-Jørgensen is with the Dept. of Computer Science at Karlstad University, Sweden.
E-mail: toke.hoiland-jorgensen@kau.se

However, if the queue is **not** already active, it is made active by being added to the end of the list of *new queues*.

When dequeueing a packet, FQ-CoDel first finds a queue to dequeue from. This is done by first looking at the list of new queues, which gives priority to queues that recently transitioned from inactive to active. If the list of new queues is empty, a queue is selected from the list of *old queues* (which is every queue that is not a new queue). Having selected the appropriate queue (either new or old), that queue gets to dequeue packets at most totalling *quantum* bytes (which is configurable, but defaults to one MTU), and afterwards the queue is moved to the end of the list of old queues. When a queue becomes empty, it is removed (and so transitions to the inactive state) as long as it has transitioned through the list of old queues at least once.

Since empty queues return to the inactive state, it is possible for a flow to have all its packets trigger the re-activation of the queue when they arrive at the router, which will give the flow effective priority for its entire duration. In the following, we explore what it takes for a flow to achieve this.

## IV. Analytical framework

Consider an FQ-CoDel instance managing a bottleneck with transmission rate $R$ bytes per second, with $N$ backlogged flows sharing the bottleneck (and so each achieving a rate of $R/N$ bytes per second). We do not concern ourselves with the performance of the $N$ flows, and we assume no hash collisions occur. We furthermore assume all flows transmit packets of equal size $L$ bytes and that the FQ-CoDel quantum $Q = L$.

### A. One sparse flow

Consider a sparse flow $S$ transmitting packets of size $L_S \leq L$ bytes. What is the maximum transmission rate that permits this flow to be prioritised by the sparse flow mechanism? We first assume that the packets of $S$ are equally spaced with inter-arrival time $I_S$ seconds.

When a packet from flow $S$ arrives at the bottleneck, it will have to wait for the packet currently being serviced to complete transmission. After this, the queue of flow $S$ will be activated as a new queue (i.e., get priority) and be serviced immediately. Once the packet has been transmitted, the queue will be moved to the end of the list of old queues, and if it is still empty after the scheduler has cycled through all the backlogged flows, it will be removed.

This means that to get treated as sparse, the next packet from $S$ has to arrive after the queue has been removed from the scheduler. I.e., after the transmission of the previous packet in $S$, plus the bulk packet being serviced on arrival, and one additional packet from each backlogged flow. This translates to the following constraints on $S$:

$$I_S > \frac{L(N+1) + L_S}{R} \Rightarrow R_S < \frac{R}{\frac{L}{L_S}(N+1)+1} \tag{1}$$

Where $R_S$ is the rate of flow $S$.

Next, we consider what happens if the packets of $S$ are not equally spaced (i.e., that $I_S$ varies between subsequent packets), but still obeys the rate restriction in (1). There are two cases to consider: The case where the packets of $S$ are sent in *bursts* of several back-to-back packets with longer spaces between them, and the case where the inter-arrival time simply varies so that, say, every other packet pair obeys (1) and every other pair does not.

In the case of bursts we assume that the bursts themselves are equally spaced over the lifetime of the flow. If the total burst size is less than the quantum size (i.e., $Q >= nL_S$ for bursts of $n$ packets), all packets in the burst will be dequeued at the same time, and we can simply consider the behaviour equivalent to the case where the flow consists of single equally spaced packets of size $nL_S$. If the burst is larger than the quantum size, the first $Q$ bytes of each burst will be dequeued immediately, while the rest will be queued until the next round of the scheduler[1].

For the non-burst case, we consider the packets $p_1, \ldots, p_n$ of flow $S$ with inter-arrival times $i_1, \ldots, i_{n-1}$ since the previous packet. By assumption, the average inter-arrival time is $I_S$ and obeys (1). This means that inter-arrival times will alternate between being less than or more than $I_S$. I.e., every sequence of consecutive packet arrivals with inter-arrival times $i_0^-, \ldots, i_j^- < I_S$ will be followed by a sequence of packet arrivals with inter-arrival times $i_0^+, \ldots, i_k^+ > I_S$ (otherwise (1) wouldn't hold). We label the $i$'th sequence of packets with inter-arrival times $< I_S$ as $I_i^-$, and the (ordered) set of all such sequences as $\mathbf{I}^-$. Similarly, the $j$'th sequence of packets with inter-arrival times $>= I_S$ are labelled $I_j^+$, with the set of all such sequences given as $\mathbf{I}^+$. We furthermore impose a *regularity constraint* on the flow:

$$\forall I_i^- \in \mathbf{I}^- : \frac{\overline{I_i^-} + \overline{I_i^+}}{2} \geq I_S \tag{2}$$

where $\overline{I_i^-}$ is the average value of $i_k \in I_i^-$. I.e., (2) states that every sequence of packets with inter-arrival times smaller than $I_S$ must be followed by a sequence of packets with inter-arrival times larger than $I_S$, such that the average inter-arrival time satisfies (1) when looking only at those two sub-sequences.

Given these constraints, packets in $I^+$ will receive the low latency performance from the sparse flow optimisations, while packets in $I^-$ will arrive while the queue is already being scheduled, and so will experience higher queueing latency. The actual queueing latency experienced by packets in $I^-$ depends on the distribution of packets; exploring this is out of scope for this analysis.

### B. Multiple sparse flows

If $M$ sparse flows go through the bottleneck, and we assume that all sparse flows have the same packet inter-arrival time $I_S$, this inter-arrival time will have to satisfy:

$$I_S > \frac{L(N+1) + L_S M}{R} \tag{3}$$

[1]Since we assume that the average rate of the flow obeys (1), the queue has to be cleared out before the next burst.

In the worst case scenario, the sparse flows synchronise (so packets from all flows arrive at the same time). In this case, the expected queueing latency for all sparse flows will be:

$$\frac{L_S(M-1)+L}{2R} \tag{4}$$

Where the $L$ is due to the bulk flows not being preempted.

However, this worst-case latency is only seen if the sparse flows synchronise so that their packets arrive at the same time (and have to queue behind one another). We can express expected queueing latency of a sparse flow in the average case by modelling the arrivals of sparse flows as follows.

Since we have bounded the inter-arrival time for each flow by (3), all flows are by assumption sparse themselves. This means that when a packet on a given sparse flow arrives at the bottleneck, it will not queue behind any other packets from the same flow, but only behind other sparse flows. Since the sparse flows are served in round-robin order, this becomes equivalent to a FIFO queue of flows waiting to be serviced (each of which has a single packet queued), and so we are really expressing the distribution of flow start times. Assuming Poisson arrivals for the flows, this system can be expressed as an $M/D/1$ queue (as link capacity and packet sizes are fixed). This will allow us to express an upper bound on the expected queueing latency of the sparse flows (since the flow arrival distribution with a fixed number of flows would be a right-truncated exponential distribution, rather than the exponential distribution assumed in an $M/D/1$ setting).

This $M/D/1$ queueing system has the following values for arrival rate ($\lambda$), service rate ($\mu$) and utilisation ($\rho$):

$$\lambda = M/I_S, \quad \mu = R/L_S, \quad \rho = \frac{ML_S}{RI_S} \tag{5}$$

From this, we can straight-forwardly express the expected queueing time $\omega_q$ as a function of the number of sparse flows, the packet size and inter-arrival times and the link rate:

$$\begin{aligned}\omega_q = \frac{\rho}{2\mu(1-\rho)} &= \frac{\frac{ML_S}{RI_S}}{2\frac{R}{L_S}\left(1-\frac{ML_S}{RI_S}\right)} \\ &= \frac{ML_S}{2\frac{R}{L_S}\left(RI_S - ML_S\right)}\end{aligned} \tag{6}$$

This is useful for predicting and upper bound the expected queueing time for any concrete flow type (where these values are known), as we will see in Section V. Note that in the case where there are also bulk flows present, we need to add $L/2R$ to the expected queueing time, to account for the packet that is being processed when a packet from a sparse flow arrives.

### C. Impact on bulk flows

Since the sparse flow optimisation simply corresponds to inserting new queues at the head of the round-robin list instead of at the tail, the steady-state performance impact on bulk flows is the same as for DRR; i.e., given two flows flow $i$ and $j$, for each dequeue opportunity afforded to $i$, $j$ has at least one dequeue opportunity. As such, the expected service given to each flow scales with the total number of bulk and sparse flows (i.e., it is proportional to $N+M$), in the worst case. In practice, many sparse flows will have rates significantly lower than the bulk flows, in which case the DRR scheduler will divide the spare capacity between the backlogged bulk flows.

### D. Impact of changing the quantum

We initially assumed that the quantum $Q=L$, which means that a bulk flow can dequeue a full packet every time it is scheduled. If $Q>L$, every bulk flow is still serviced every scheduling round, but may dequeue more than one packet. Whereas, if $Q<L$, each bulk flow will get a dequeue opportunity every $L/Q$ scheduling rounds and, conversely, only $QN/L$ bulk flows will dequeue a packet each round. In the case where only bulk flows are present, these two effects cancel each other out. However, in the presence of sparse flows, the quantum impacts the bounds on sparse flow inter-arrival time given in (3). Assuming $Q >= L_S$ so sparse flows always dequeue a full packet when they are scheduled, this becomes:

$$I_S > \frac{Q(N+1)+L_S M}{R} \tag{7}$$

## V. Real-world examples

Using (6) and (7) we can compute two useful properties: The maximum number of sparse flows a given link can sustain as a function of the number of backlogged bulk flows at the bottleneck, and the expected queueing time for each such sparse flow. To do this, we can rewrite (7) as follows, while also including the case where there are no bulk flows:

$$M_{max} < \begin{cases} \frac{I_S R}{L_S}, & \text{if } N=0 \\ \frac{I_S R - Q(N+1)}{L_S}, & \text{if } N>0 \end{cases} \tag{8}$$

We also need to assign some values to the variables in the equations. For our example, we consider a 10 Mbps Ethernet link where bulk flows transmit full-size (1518 bytes) packets and the quantum is set to coincide with this full packet size (as is the default in FQ-CoDel), and the sparse flows consist of a number of G.711 VoIP flows at the highest (64 Kbps) data rate, which transmits packets of 218 bytes (160 bytes payload + RTP, UDP, IP and Ethernet headers) at a fixed 20 ms interval. These values are summarised in Table I.

TABLE I: Values used in the numerical example

| Variable | Value |
|---|---|
| $Q$ | $1518B$ |
| $L_S$ | $218B$ |
| $I_S$ | $0.02s$ |
| $R$ | 1.25 MB/s (10 Mbps) |

With these values, (8) tells us that the maximum number of VoIP flows the bottleneck can handle while still treating them as sparse flows, is a linear function of the number of bulk flows backlogged at the bottleneck. With no bulk flows, 114 sparse flows can be serviced, which correspond to the number of VoIP flows the bottleneck link has capacity for (each flow transmits at a link-level rate of 87.2 Kbps). With 15 bulk flows backlogged, only three simultaneous VoIP flows can traverse the bottleneck link as sparse flows, and with more backlogged flows, the VoIP flows will no longer be treated as sparse. The number of sparse flows per bulk flow is related to the ratio between the quantum and the packet size of the VoIP flows.

Turning to the expected queueing time of the sparse flows themselves, Figure 1 shows this as a function of the number of sparse flows. To verify that the model accurately predicts an upper bound on the queueing latency, we have also created a simulation of FQ-CoDel in the Salabim event-driven simulator[2]. The results from the simulation are also included in the figure.[3] Note that the expected queueing time does not depend on the number of bulk flows, other than to limit the number of sparse flows that can be supported. In fact, a sparse flow can experience lower latency when competing against bulk flows, than when competing against a large number of other sparse flows. This limiting is illustrated by the two graphs, where Figure 1a shows the case of no bulk flows and Figure 1b shows the case of a single bulk flow.

The thing to note here is that the expected queueing latency is kept very low for all the sparse flows, and that adding bulk flows does not change this, other than to add a constant to the queueing time corresponding to the packet being processed when a sparse flow packet arrives. In fact, as Figure 1b shows, the expected queueing latency even for the maximum number of sparse flows that the link can handle, is less than two milliseconds with one or more bulk flows limiting the number of sparse flows. So as long as an operator is using (8) to calculate the max number of sparse flows the link can support, she can be confident that the sparse flows themselves will achieve very low queueing latency at the bottleneck.

## VI. Conclusion

We have analysed the performance characteristics of the sparse flow optimisation of FQ-CoDel. This analysis shows the constraints that flows must satisfy to be considered sparse in a given scenario, which is dependent on the number of flows (both bulk and sparse) and the link rate. We also formulate expressions for the expected queueing latency for sparse flows.

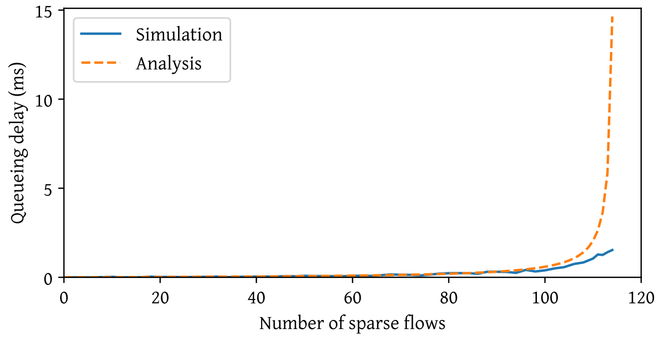
Using a numerical example, we also show that for a given link and a given type of sparse flows (VoIP traffic), the number of sparse flows that a given bottleneck can service with the low sparse flow latency is only dependent on the number of backlogged bulk flows at the bottleneck. And that as long as the maximum number of sparse flows is not exceeded, all sparse flows can expect a very low queueing latency.

## VII. Acknowledgements

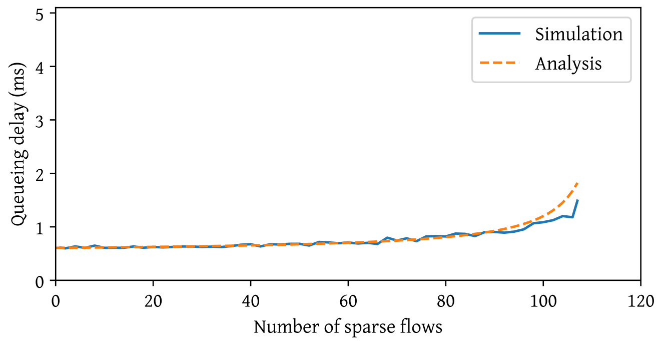
Many thanks to Daniel Larsson and Martin Wahlberg for implementing and running the simulation and to Dave Taht, Johan Garcia, Per Hurtig and Anna Brunstrom for feedback on the ideas and for reviewing various versions of the manuscript.

[2]http://www.salabim.org

[3]The bulk flows used in the simulation are fixed rate UDP flows. Further details of the simulation runs are omitted here due to space constrains, but are available (along with the full simulation source code) in [11].



(a) No bulk flows



(b) One bulk flow

Fig. 1: Expected queueing delay as a function of the number of sparse flows at the bottleneck.

## References

[1] T. Høiland-Jørgensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet, “The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm,” RFC 8290 (Experimental), RFC Editor, Jan. 2018. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8290.txt

[2] T. Høiland-Jørgensen, P. Hurtig, and A. Brunstrom, “The Good, the Bad and the WiFi: Modern AQMs in a residential setting,” *Computer Networks*, vol. 89, pp. 90–106, Oct. 2015.

[3] N. Khademi, D. Ros, and M. Welzl, “The new AQM kids on the block: Much ado about nothing?” *Technical Report, Oslo Univ.*, 2013.

[4] J. Kua, G. Armitage, and P. Branch, “The impact of active queue management on dash-based content delivery,” in *IEEE 41st Conference on Local Computer Networks (LCN)*, Nov 2016, pp. 121–128.

[5] M. Shreedhar and G. Varghese, “Efficient fair queuing using deficit round-robin,” *IEEE/ACM Transactions on Networking*, Jun. 1996.

[6] M. H. MacGregor and W. Shi, “Deficits for bursty latency-critical flows: DRR++,” in *IEEE International Conference on Networks, 2000*. IEEE, 2000, pp. 287–293.

[7] T. Bonald, L. Muscariello, and N. Ostallo, “Self-prioritization of audio and video traffic,” in *2011 IEEE International Conference on Communications (ICC)*. IEEE, 2011, pp. 1–6.

[8] G. Carofiglio and L. Muscariello, “On the impact of TCP and per-flow scheduling on internet performance,” *IEEE/ACM Trans. Netw.*, vol. 20, no. 2, pp. 620–633, 2012.

[9] F. Checconi, L. Rizzo, and P. Valente, “QFQ: Efficient packet scheduling with tight guarantees,” *IEEE/ACM Transactions on Networking (TON)*, vol. 21, no. 3, pp. 802–816, 2013.

[10] A. Kortebi, L. Muscariello, S. Oueslati, and J. Roberts, “Evaluating the number of active flows in a scheduler realizing fair statistical bandwidth sharing,” in *ACM SIGMETRICS Performance Evaluation Review*, vol. 33. ACM, pp. 217–228.

[11] T. Høiland-Jørgensen, D. Larsson, and M. Wahlberg, “FQ-CoDel Queue Management Algorithm - Sparse Flow Analysis Software,” Sep. 2018. [Online]. Available: https://doi.org/10.5281/zenodo.1420467