# Evaluation of Priority Scheduling and Flow Starvation for Thin Streams with FQ-CoDel

Eduard Grigorescu, Chamil Kulatunga, Gorry Fairhurst
School of Engineering, University of Aberdeen, UK
{eduard, chamil, gorry}@erg.abdn.ac.uk

Nicolas Kuhn
Télécom Bretagne, IRISA
nicolas.kuhn@telecom-bretagne.eu

*Abstract*— **Bufferbloat is the result of oversized buffers and induced high end-to-end latency experienced by applications across the Internet. This additional delay can adversely impact thin streams that frequently exchange small amounts of data, but have stringent latency requirements. Active Queue Management (AQM) techniques, such as Controlled Delay (CoDel), can control the queuing delay in a network device to ensure low latency by dropping packets to indicate incipient congestion. FlowQueue-CoDel (FQ-CoDel) is a scheduling scheme that creates one sub-queue per flow and applies CoDel on each of them. FQ-CoDel features: (1) priority scheduling for low-rate traffic; (2) flow isolation; (3) queue management with CoDel. First, this paper fills a gap in the understanding of FQ-CoDel by analyzing what features are of interests for providing low latency for thin streams applications. Second, this paper provides the first analysis of the limits of the flow starvation mechanisms and show that FQ-CoDel is vulnerable to Denial of Service (DoS) attacks.**

***Keywords— Bufferbloat, AQM, Scheduling, CoDel, FQ-CoDel, Thin-streams, Flow Starvation***

## I. Introduction

Network devices require buffers to store bursts of incoming packets prior to forwarding. These buffers have traditionally implemented a FIFO/DropTail buffer policy, with passive queue management that drops packets only when a queue is full. While buffers are needed to achieve statistical multiplexing or to guarantee high bottleneck utilization when the available capacity fluctuates, excessive buffering can lead to large queues resulting in high network latency. This issue is known as Bufferbloat [1]. Although capacity between Internet core routers is often over-provisioned, this is rarely the case for access networks, using technologies such as ADSL, satellite and mobile broadband [8]. Latency has become a major issue in such access networks in the past decade where excessive queuing, affects application performance [9].

Active Queue Management (AQM) techniques, such as Random Early Detection (RED) [2], have been proposed for over a decade to appropriately manage the buffer by indicating impeding congestion to responsive transport protocols, to avoid building a standing queue. However, AQM schemes have been reported to be usually turned off, as they were hard to tune. More recent protocols, such as CoDel [3] or PIE [4], have been designed to especially tackle the Bufferbloat, but with RED deployment issues in mind. Recent IETF work recommends the deployment of AQM as one solution to reduce latency [6]. On top of AQM schemes, scheduling algorithms, managing packet scheduling and isolation/capacity allocation among flows, can be introduced. As one example of a scheme that mixes both classes, FlowQueue-CoDel (FQ-CoDel) [7] is a scheduling scheme that features prioritization and flow isolation. FQ-CoDel creates one sub-queue per flow and applies CoDel on each of them. The awareness of the latency resulting from over-provisioned buffers has been accompanied by an increase in real-time applications such as Voice over Internet Protocol, gaming or financial trading applications. As one example, the latency experienced by gamers can directly impact the perceived value of the network service [10]. These thin streams applications generally send sparse streams of small time-critical packets [14].

Since FQ-CoDel features a mechanism that prioritizes low-rate traffic, the benefits for the increasing number of thin streams that carry latency sensitive applications needs to be assessed. This paper fills the gap in research evaluating FQ-CODel when the traffic is a mix of thin streams and bulk flows and evaluates which part of FQ-CoDel (CoDel, prioritization, flow isolation) provides improvement. Because FQ-CoDel features flow prioritization, we also evaluate to what extent its flow starvation prevention mechanism works.

The remainder of this paper is organized as follows. Section II describes FQ-CoDel, by clearly identifying when each internal mechanism adds value. Section III presents the simulation setup used to assess the suitability of FQ-CoDel for carrying latency sensitive thin streams over a capacity limited path. Section IV discusses which part of FQ-CoDel might provide improvements in the queuing delay experienced by thin stream applications. Section V assesses the limits of the flow starvation prevention mechanism. Section VI concludes this work.

## II. FlowQueue-CoDel Algorithm

AQM algorithms seek to control network buffering level by sending messages to trigger transport congestion control, *i.e.*, by early dropping/marking packets.

### A. Over the Need for Flow Scheduling for Thin-Streams

AQM dropping techniques on their own may not be sufficient to satisfy the strict latency requirements for thin stream applications [5],[6]. Indeed, traffic such as file transfers or unresponsive constant rate streaming flows, with different time constraints, may share the bottleneck with thin streams.

This would result in a non-negligible queuing delay experienced by the thin streams.

Some form of flow isolation might be required to separate and protect the time sensitive small flows from larger and more aggressive flows. Scheduling algorithms can provide per-flow or per-class queuing to isolate traffic classes to guarantee the specific constraints of latency sensitive applications. As one example, if thin flows are assigned different subqueues to other flows, a scheduling scheme may protect the thin flows from the background traffic. This would avoid an increasing queuing delay for a latency sensitive application when the background flows build in a subqueue.

One simple isolation method is per-flow queuing [11] in which each flow is deterministically assigned its own virtual sub-queue. However, this requires network devices to classify each flow, which can be difficult when dealing with traffic aggregates or when encryption is used. Each of the subqueues is served in a round-robin manner, improving fairness between flows [5]. A more complex example is Stochastic Fair Queuing (SFQ) [5] that provides a statistical alternative in the way the subqueues are served.

### B. *FlowQueue scheduling in FQ-CoDel*

FQ-CoDel uses a modified Deficit Round Robin (DRR) scheduler. The default flow classifier of FQ-CoDel hashes incoming packets stochastically to a subqueue based on a 5-tuple classifier – IP source and destination address, protocol, and port numbers. The scheduler of FQ-CoDel can be applied on any flow isolation technique and is not limited to a 5-tuple classifier. A byte-based scheduler, rather than a packet-based scheduler, is used to select the next packet for transmission.

The scheduling in FQ-CoDel is based on three lists of subqueues that are represented in Figure 1, the "new" list, the "old" list and the "empty state" list. In the rest of this subsection, we will detail how these lists are managed. We focus on how they can prevent flow starvation and how they prioritize some classes of traffic.

At initialization, FQ-CoDel creates a set of subqueues (by default 1024 subqueues), all are placed in the "empty state" list. When there is at least one byte of data that enters a subqueue, the subqueue is defined as *active.* This subqueue is initially placed in the "new" list, but may later be moved to the "old" list.

When a packet is enqueued, it is added at the end of the subqueue corresponding to the 5-tuple classifier: if the 5-tuple of this packet does not correspond to any list, it is added to an existing "empty list", which becomes a "new" list.

We describe here how the *deficit* is decreased and how the decision to dequeue data is taken:

- The scheduler first cycles through the "new" list of subqueues, allowing each subqueue to dequeue up to one "quantum" of packets and updating the deficit value.
- For each subqueue, the scheduler checks the deficit. If the deficit becomes negative, the subqueue is moved to the end of the "old" list and its deficit is updated to the sum of the size of the quantum and the previous deficit.
- When a subqueue in the "new" list is found to be inactive (no packets in the buffer), it is placed at the end of the "old" list, and its deficit is re-initialized to the "quantum" size.

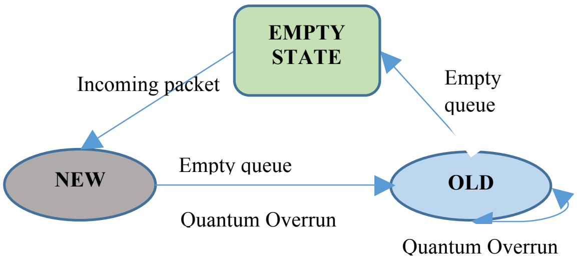

Figure 1. FQ-CoDel State Machine

- When the "new" list becomes empty, the scheduler examines the "old" list by repeating the algorithm used on the "new" list. When a subqueue from the "old" list becomes empty, it is removed.

### C. *Flow Starvation Prevention Mechanism in FQ-CoDel*

When the deficit of a given queue, $Q$, becomes negative, it is moved to the end of the "old" list. This ensures that when FQ-CoDel loops over the "old" list, a subqueue that had been previously pushed to the end of the "old" list would be given the opportunity to transmit a quantum of data before $Q$. When a queue is empty, FQ-CoDel pushes the queue at the end of the "old" list. This feature, referred to as the Starvation Prevention Mechanism, is supposed to prevent starvation, providing some transmission opportunities for the flows already placed in the "old" list.

### D. *AQM in FQ-CoDel*

Each subqueue is individually managed using the CoDel algorithm: FQ-CoDel classifies each packet, it timestamps the packet and appends it to the tail of the selected subqueue. CoDel controls the maximum size of each subqueue, using its default parameters: target delay of 5 ms and interval 100 ms. CoDel applies its control law and may discard at least one packet from the head of a scheduled subqueue if needed, before returning a packet for dequeuing (or no packet if the subqueue becomes empty). This should avoid buffer overflow and guarantee low queuing delay, if there are many flows and the scheduling introduces a non-negligible queuing delay.

## III. EVALUATION TOOL SET FOR THE CAPACITY LIMITED NETWORK USE CASE

This section justifies our focus on the capacity-limited network use case. We also present the simulation tool set and the network topology used in our simulations.

A 10 Mbps (or higher speed) bottleneck that experiences congestion, has a transmission speed that is sufficiently low to result in negligible transmission delay, even for large packets, compared to the latency required by typical thin stream applications (approx. 1.2 ms for a packet of 1500 B sent at 10 Mbps). When the bottleneck has a smaller rate (e.g., the downlink of a rural access link operating at 1-2 Mbps or an uplink, operating at 1/10 of this speed), the packet transmission delay increases further (e.g., 12-6 ms for 1500 B packets). The cumulative effect of scheduling many competing flows can result in some flows becoming "choked". Because (1) the impact of the thin streams on the flow starvation of FQ-CoDel is exacerbating and (2) the latency sensitive applications would be more affected when there is no priority scheduling, we therefore focus on this use case.

Figure 2 presents the dumb-bell topology for our simulations in *ns-2*. The capacity of the bottleneck was 2 Mbps and the one-way-delay was 47.5 ms. The non-bottleneck links were configured with 1.25 ms one-way-delay and 100 Mbps capacity. The RTT of the network path was 100 ms.

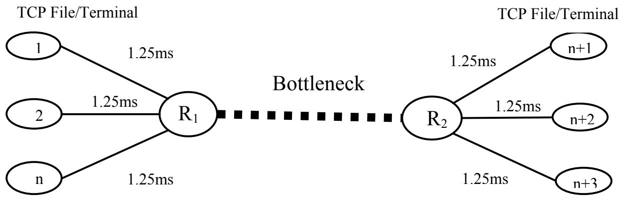

Figure. 2. Dumb-bell simulation topology

At node R1, the buffer size was set to twice the size of the Bandwidth-Delay Product (BDP) and either FQ-CoDel or CODel applied to the node. For other nodes, the queuing discipline was DropTail, with a buffer size of 300 packets.

## IV. Analyzing the Performance of Thin Streams with FQ-CoDel

FQ-CoDel may be divided into different constituent mechanisms. There is one AQM scheme per subqueue, one priority scheme and a set of flow isolation techniques. If FQ-CoDel is to be used to support thin streams, it is important to assess which mechanism within FQ-CoDel is actually responsible for realizing any benefits observed.

### *A. The Benefits of Flow Isolation for Thin Streams*

To support our evaluation, this subsection introduces a custom non-prioritization version of FQ-CoDel, FQ-CoDel Without Prioritization (FQ-CoDel WP). FQ-CoDel WP does not make the distinction between the "new" and "old" list and subqueues are created one after the other, while the scheduler still visits subqueues similar to SFQ. This is used to assess the benefits of the flow isolation for the thin streams applications.

This subsection compares the suitability of using FQ-CoDel WP, CoDel or SFQ to carry thin streams applications. The following traffic was considered: (1) 1 to 5 TCP bulk flows, using File Transfer Protocol (FTP) for the entire period of the simulation with a packet size of 1500 B. (2) 1 thin TCP with an inter-packet interval of 638 ms and a packet size of 100 B, which is representative of the traffic generated by the game Anarchy Online [12]. (1) and (2) used TCP New Reno with the SACK option and an initial window of 3 packets. Figure 3 shows the Cumulative Density Function (CDF) of the queuing delay experienced by the thin-stream flows, when the AQM is CoDel, SFQ or FQ-CoDel WP.

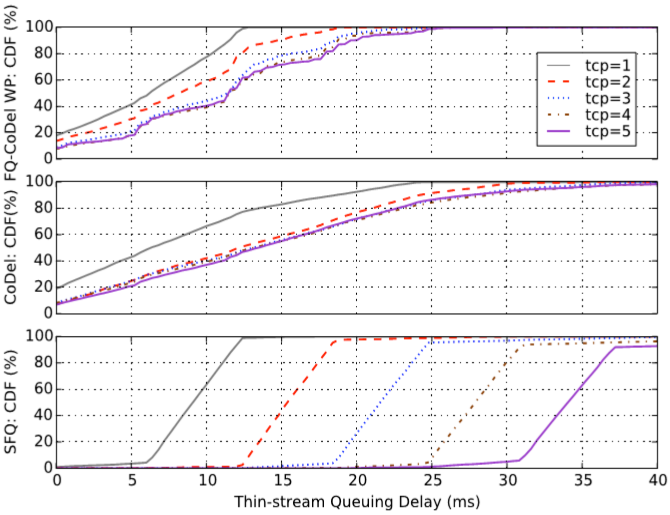

Figure 3. CDF vs. Gaming Flow Queue Latency

With SFQ, the queuing delay increases as the level of network congestion increases (increasing number of TCP flows). For each TCP packet, the Round Robin scheduler of SFQ results in a 6 ms transmission time per packet (1550 B for a 2 Mbps bottleneck). As expected, a Round Robin scheduler that services a higher number of queues leads to increased queuing delay: we observe a linear increase of the queuing delay when the number of TCP flows increases. As an example, the thin stream subqueue will experience 5*6ms >= 30ms queuing delay with 5 bulk TCP flows before it has the opportunity to again be serviced.

With CoDel, the observed queuing delay is lower than with SFQ, whatever the number of TCP bulk flows. The early drops in CoDel tend to maintain a small queue that reduces the delay experienced by the thin stream flows. This queuing delay does significantly increase with the number of flows competing for queue space, as compared to SFQ.

With FQ-CoDel WP, the queuing delay experienced by the thin streams is lower than when using CoDel and lower than with SFQ. These results show that (1) dropping packets with CoDel enables a latency reduction; (2) flow isolation alone cannot reduce the queuing delay experience; (3) the performance of flow isolation techniques are sensitive to the traffic load. We can conclude that when FQ-CoDel features flow isolation, it results in lower queuing delay than with CoDel alone, showing that the flow isolation technique, along with CoDel drops, can offer the best of the two schemes.

### *B. The Importance of Thin Stream Prioritization and CoDel*

In this subsection, we will explore the benefits of introducing prioritization in FQ-CoDel to reduce the queuing delay experienced by thin streams. We compare the performance of the default FQ-CoDel with "FQ-CoDel WP 100ms", which is a modified version of FQ-CoDel in which the target of CoDel is increased to 100 ms (instead of the default 5 ms) and prioritization is disabled. Therefore, the differences between FQ-CoDel WP 100ms and FQ-CoDel are

(1) CoDel will allow more queuing in FQ-CoDel WP 100 ms and (2) the absence of prioritization in FQ-CoDel WP 100 ms.

The traffic considered in this section is the same as in IV-A. Figure 4 presents the CDF of the queuing delay experienced by the thin streams for various numbers of TCP flows with FQ-CoDel or FQ-CoDel WP 100ms as an AQM.

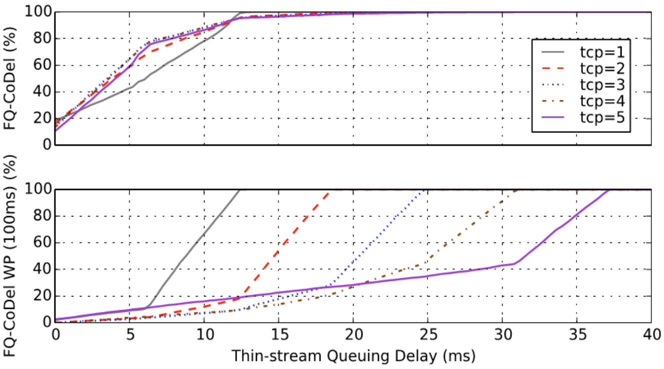

Figure 4. CDF vs. Game Flow Latency for FQ-CoDel (5 ms target delay) and non-prioritized FQ-CoDel WP (100ms target delay)

The performance with FQ-CoDel WP 100ms shows that without prioritization and with CoDel less aggressive, the performance of this scheme is close to those of SFQ. The flow isolation of FQ-CoDel may reduce the latency (as shown in the previous section), but the CoDel part of the algorithm has a non-negligible benefit in reducing the queuing delay.

Also, if we compare the performance of FQ-CoDel and those of FQ-CoDel WP, shown respectively in Figure 4 and Figure 3, we see that the prioritization contributes in reducing the queuing delay experienced by the thin streams. FQ-CoDel provides lower queuing delay, for the traffic loads considered, whereas it is sensible to the traffic load with FQ-CoDel WP.

Therefore, based on the results presented in this subsection, we can conclude that the flow prioritization of FQ-CoDel provides a useful latency reduction and makes the queuing delay of the thin streams less sensible to the traffic load. We also confirm the conclusions of section IV-A, which are that the CoDel part of FQ-CoDel is essential to provide low latency in the context of capacity-limited networks.

### C. *Thin Stream with various inter-packet arrival times*

Depending on the burst size and the frequency between bursts, when a second burst of packets reach the queue, the packets of the previous burst might still be in the “old” list, or they might have left the queue and the second burst would be prioritized. Flows with a different pattern of packet inter-arrival times but similar packet sizes can be treated differently.

We consider three cases of traffic generation: (1) 1 gaming flow, 1 TCP bulk flow and 1 VoIP flow; (2) 1 gaming flow, 1 TCP bulk flow and 5 VoIP flows; (3) 1 gaming flow, 5 TCP bulk flows and 1 VoIP flow. Both the VoIP and the gaming flows use TCP. The inter-packet arrival time for the VoIP flows is in 20-30 ms and in 600-1000 ms for the gaming flows. The packet size for both applications is on average 100 B. Results are shown in Figure 5.

With FQ-CoDel WP 100 ms and FQ-CoDel WP, the queuing delay for both voice and gaming flows increases in case (3), because of the high number of TCP flows, showing again the importance of CoDel coupled with isolation and prioritization. However, in cases (1) and (2), the load level is lower than in case (3), and we see a small gain of using the priority scheme of FQ-CoDel. With FQ-CoDel WP, we notice a small difference between gaming and voice flows this may be related to their different inter-packet arrival times, showing that the priority scheme of FQ-CoDel actually lets the AQM scheme reduce the queuing delay for applications with different inter-packet arrival times.

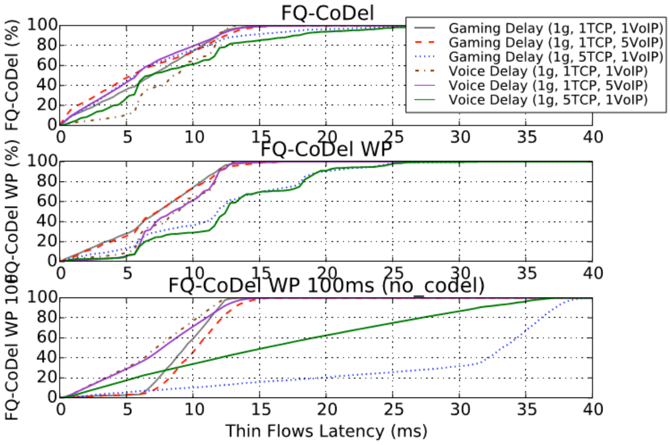

Figure 5. Thin Flows Latency for prioritized and non-prioritized FQ-CoDel

The results presented in this subsection let us conclude that FQ-CoDel can deal with thin streams flows that have different inter-packet arrival times. We also highlighted that the prioritization mechanism seems to provide benefits. Another possible source of improvement might be the starvation prevention mechanism; we expect future work to evaluate this.

## V. FLOW STARVATION AND FQ-CODEL

We discuss here the limits of the flow starvation prevention mechanism. The starvation prevention mechanism of FQ-CoDel is the following: FQ-CoDel would place an empty queue from the “new” list to the end of the “old” list. Also, if a queue is empty in the “old” list, it would be removed and considered as “new” when its packets reach the queue.

We assess the limits of the starvation prevention mechanism of FQ-CoDel. Under Denial of Service (DoS) attacks, FQ-CoDel scheduler may loop only over the “new” list, preventing the flow starvation mechanism to work. It is crucial therefore to assess the performance of the flow starvation mechanism when thin streams contribute a large proportion of the traffic compared to bottleneck capacity. A starvation prevention method that may prevent this issue from occurring. To verify the benefits of using this mechanism, we implemented FQ-CoDel Without its Starvation Prevention Mechanism (FQ-CoDel WSPM). FQ-CODel WSPM is a version of FQ-CoDel where starvation prevention is disabled, that is while looping over the “new” list, the algorithm would not move empty lists from the “new” list to the “old” list, but rather wait that the deficit for the list is negative.

The following traffic were considered: (1) 1 TCP bulk flows, using FTP for the entire period of the simulation with a packet size of 1500 B; (2) 5 to 45 thin unresponsive UDP flows with an inter-packet interval of 30 ms and a packet size of 100 B, which is representative of the traffic generated by a Skype session [12]. (1) used TCP New Reno with the SACK option enabled and an initial window of 3 packets.

Figure 6 presents the throughput achieved by the TCP flow as a function of the number of thin flows, with FQ-CoDel and with FQ-CoDel WSPM. When the number of thin streams is greater than 30, the TCP bulk flow becomes starved, both when using FQ-CoDel and with FQ-CoDel WSPM. This shows that unresponsive traffic can impact the performance, since the flow starvation prevention mechanism of FQ-CoDel is not sufficient to prevent the resulting congestion. The lack of a difference between FQ-CoDel with and without this mechanism highlights that it does not provide significant benefit when there is a high level of congestion.

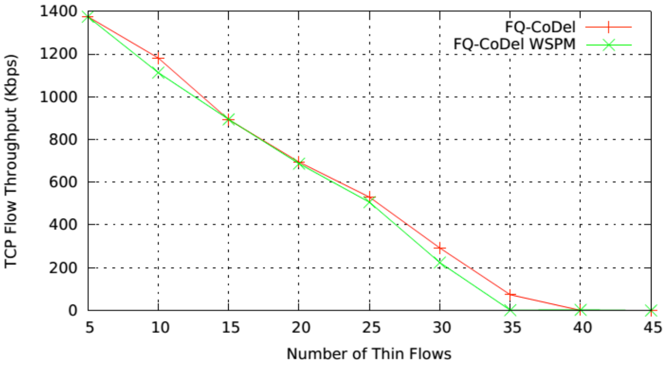

Figure 6. TCP Throughput with FQ-CoDel and with FQ-CoDel WSPM

We finally consider the response to overload. An FQ-CoDel scheduler could be vulnerable to a Denial of Service (DoS) attack where traffic intentionally tries to disrupt normal operation, e.g., a large number of thin streams would be intentionally injected into the network bottleneck. This may cause the scheduler to become locked serving only flows in "new" list subqueues, leading to unwanted interactions between TCP flows and bursts of unresponsive flows that result in delay. For such a low capacity link, flows in the "old" list would barely receive an opportunity for transmission, and would become starved. CoDel would also drop queued packets from the old list. This suggests the need to design more sophisticated overload protection [6].

We note that other modern AQM algorithms, such as PIE, can also be combined with isolation methods [6] by introducing mechanisms similar to those described in this paper. The combined methods have been reported to support latency-sensitive thin-stream applications [8]. We leave exploration of the reasons behind the performance limits of the flow starvation prevention mechanism in capacity-limited networks as a part of our future work on this topic.

## VI. CONCLUSION

This paper describes the operation of the various components mechanisms in FQ-CoDel and explores how thin-stream applications can benefit from these mechanisms to reduce their experienced latency and mitigate the impact of sharing capacity with other types of network traffic. We measured that the flow isolation of FQ-CoDel is the main factor resulting in improved latency performance for thin flows over bottlenecks with limited bandwidth. Such isolation would be impossible with a traffic aggregate that it cannot dissect (e.g., when encrypted Virtual Private network tunnels are used). We identified that CoDel in FQ-CoDel can provide improvements for this specific traffic, this means that when a classifier cannot be used, low latency may still be guaranteed.

Flow prioritization has been introduced within FQ-CoDel to that favors low-rate traffic, or latency sensitive applications such as web traffic. Simulations have shown that this scheme provides a fair improvement of performance for thin stream traffic. We believe in general that deployed AQM algorithms should be made robust against overload and especially denial of service attacks, otherwise all the efforts spent in making the deployment of AQM a reality will be eroded. In the light of the results presented in this document, we encourage further research prior to deployment of the current version of FQ-CoDel. Other modern AQM algorithms, such as PIE, can also work in conjunction with isolation methods to better support latency-sensitive application. Therefore, we believe that more efforts should be spent on evaluating the performance of such hybrid mechanisms to support their large-scale deployment.

## ACKNOWLEDGEMENTS

This research was supported by the RCUK DE programme to the dot.rural Digital Economy Hub; EP/G066051/1 and part-funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700).

## REFERENCES

[1]. **Gettys, Jim**. *Bufferbloat: Dark Buffers in the Internet***.** IEEE Internet Computing, IEEE, 1990.
[2]. **Jacobson, Van & Floyd, Sally.** *Random Early Detection Gateways for Congestion Avoidance.* IEEE/ACM Transactions on Networking, 1993.
[3]. **Nichols, Kathleen & Jacobson Van** *Controlling Queue Delay.* IETF work-in-progress, 2015.
[4] **Pan, Rong, et al..** *PIE: A lightweight control scheme to address the bufferbloat problem.* IETF work-in-progress, 2015.
[5]. **McKenney, Paul E** *Stochastic Fairness Queuing***.** Beaverton : s.n., June 1990. In proceedings of INFCOM. Vol. 2, pg. 733-740.
[6] **Baker, F & Fairhurst, G**. AQM Recommendations Regarding Active Queue Management. IETF work-in-Progress
[7]. **Hoeiland-Joergensen, T., et al.** FlowQueue-CoDel. IETF work-in-progress, 2015
[8]. **White, Greg** *Active Queue Management in DOCSIS 3.X.Cable Modems*.**.** Cable Television Laboratories. 2013.
[9] **N. Leavitt**, "Network-Usage Changes Push Internet Traffic to the Edge", IEEE Computer, Vol 43, Issue 10, October 2010.
[10]. **Claypool, M. and Claypool, K**. Latency and player actions in online games. *Communications of the ACM 49*, 11 (Nov. 2005), 40-45.
[11] **Nabeshima, M.; Yata, K.** An Effective Queue Management Scheme for Data Communication. *IEEE Proceedings-Communications*. 2005
[12]. **Petlund, Andreas et al..***TCP mechanisms for improving the user experience for time-dependent thin-stream applications.* 33rd IEEE Conference on Local Computer Networks, 2008. LCN 2008
[13] **L.D Cicco, S. Mascolo, and V. Palmisano.** *Skype Video Congestion Control: An experimental investigation*. Computer Networks, 2011
[14] **Petlund, Andreas**. *Improving latency for interactive, thin-stream applications over reliable transport.* PhD Thesis. 2010