Herding the FLOQ: Flow Optimised Queueing

Mihail Yanev
Rakuten Mobile Innovation Studio

Paul Harvey
Rakuten Mobile Innovation Studio

***Abstract*—With more people working remotely, the demands placed on network operators to deliver high quality connectivity to users is at an all time high. This is especially true for web page responsiveness, a human-centric activity. Several approaches strive to ensure high quality of service (QoS) from within the browser or in-network caching, however, they do not target a core culprit of low QoS: initial packet loss in connection establishment.**

Existing works address packet loss via the active management of packet flows within network middleboxes, however, they do not address this specific type of packet loss directly.

This work introduces a new active queue management algorithm: FLow Optimised Queueing (FLOQ). FLOQ is designed to decrease packet loss during connection establishment without affecting overall network performance. Through experimental comparison with other AQM algorithms, our preliminary results show that FLOQ solves this problem, speeding up request completion times by up to 70%, improving TCP throughput by up to 20%, and decreasing UDP packet loss by up to 23%.

I. Introduction

Perhaps more than any other period, the pandemic years of 2020 to 2022 have placed the connectivity provided by the Internet at the heart of almost all human social and economic activity. This shift highlighted the need for the highest possible levels of service from network connectivity. From the network operator's perspective, this is more keenly felt than others.

Second only to video, browsing traffic represents 13% of all global Internet application traffic [1]. As a human-centric activity, browsing web pages can tolerate responses of up to 200ms before becoming disruptive [2], making fast page load time (PLT) critical for users. Beyond user Quality of Experience (QoE), fast PLT is also a necessity for business. For example, Amazon has previously stated that a one second PLT delay results in $1.6B loss in revenue [3]

In pursuit of faster PLT previous studies have looked at the problem from the perspective of web browser optimisation [4] or Internet caches [5]. However, these approaches are unable to address a key cause of slow PLT: packet loss during connection establishment. Indeed, such packet loss has been shown to cause slower connection response and increase network latency [6].

Most packets are lost as an effect of being dropped by a network middlebox. Whether a middlebox drops a packet or not is governed by the policy that decides packet admission and management of the middlebox's internal buffers. Existing management approaches of these buffers, known as *active queue management* (AQM), treat traffic from all connection phases equally. This can lead to a mismatch between the real and the expected completion requirements.

ISBN 978-3-903176-48-5

In this paper we introduce a new AQM algorithm called Flow Optimised Queueing (FLOQ). Motivated by the need to address PLT (Section II) and the limitations of existing approaches in doing so (Section III), the goal of FLOQ (Section IV) is to reduce packet loss during connection establishment while preserving overall network performance. Our initial study on synthetic data shows our goal achieved: speeding up request completion times by up to 70%, improving TCP throughput by up to 20%, and decreasing UDP packet loss by up to 23%.

FLOQ works by categorising traffic as being either responsive or non-responsive, irrespective of protocol. Additionally, FLOQ prioritizes the initial packets of a connection, decreasing establishment latency and thus improving PLT. These results are demonstrated experimentally (Section V) and compared against other state-of-the-art AQM approaches (section VI). In this way, FLOQ is **connection aware**, **protocol agnostic**, **resource preserving**, and **network load adaptive**. To the best of our knowledge, FLOQ is the first global AQM algorithm to directly address the PLT challenge. Based on the summary of the results (Section VIII) FLOQ has demonstrated real promise as an avenue for future exploration (Section VII).

II. Motivation

The Internet is a best-effort packet switched network with *middleboxes*. To help with the overall performance of the network, these middleboxes contain buffers where packets can be temporarily stored and eventually forwarded on to the next hop in the packet's journey. The management of the buffers and the admission (or not) of packets to these buffers is know as *active queue management* (AQM).

When packets are not admitted they are dropped and lost. Senders can then react to packet loss in various ways. Some actively monitor loss and upon detection reduce their sending-rate, perceiving the loss as a signal of congestion. Others will not try to detect or react to the loss.

There are multiple approaches that endpoints may use to detect congestion. Some approaches rely on acknowledgement data sent from the receiver [7], [8], others rely on timeouts [7], [9]. Generally, the former approach is faster than the latter, however, when the network experiences a serious state of *bufferbloat*, an acknowledgement-based approach, and Internet traffic generally, can suffer significantly [10].

To tackle bufferbloat some AQM algorithms, such as PIE [11], and CoDel [12] were introduced - addressing bufferbloat based on connection latency. However, a purely latency-based approach does not consider which part or phase of the connection it effects. This is important for protocols

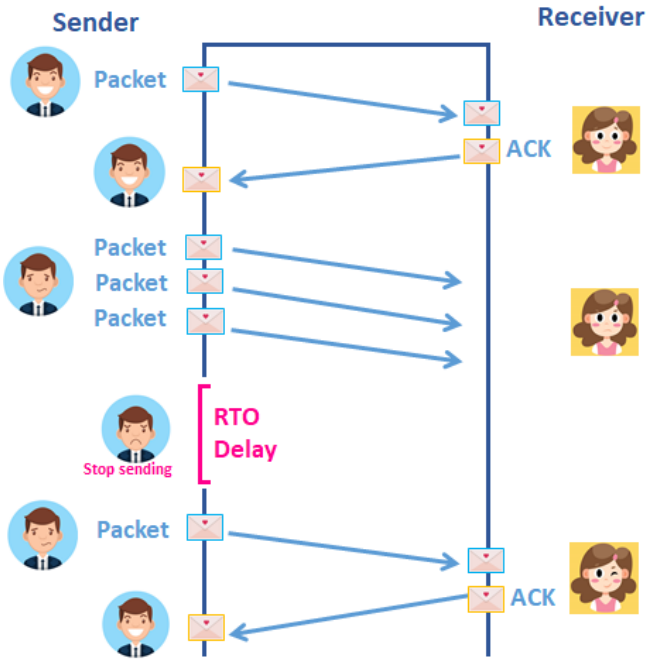


Figure 1: RTO Delay

that rely mainly on retransmission timeouts (RTO) during the connection establishment phase and shortly thereafter.

RTOs are used in protocols when there has been no acknowledgement data (ACK) for a transmitted packet. This situation may occur because the sender's packet(s) got lost, the packets were dropped at a middlebox, or because the sender has already acknowledged all of its received packets. During connection establishment, as a critical mass of packets have not been sent, the sender is often unable to rely on ACK retransmissions, and uses RTO instead, as shown in Figure 1. Any approach to improve latency and responsiveness would benefit if it minimises packet loss during establishment.

III. Related Work

The following provides an overview of related work in AQM. Middleboxes, such as Internet routers, lower level switches, device drivers, etc. contain buffers. These buffers are meant to absorb traffic bursts, however improper configuration of these buffers can lead to reduced performance in low-latency interactive applications and traffic overall. The problem of how buffers should be managed is an active research question. The Internet Engineering Task Force (IETF) had created working groups, dedicated to developing standards and recommendations for AQM algorithms[1] Historically, many AQM policies have been proposed, but a few of these made it in the real networks.

The most basic form of queue management is the first in first out (FIFO) algorithm. As the name implies, the first packet to enter the buffer will be the first packet to leave. Once the buffer is full, no more packets may enter and are dropped. All forms of AQM are based on this basic principle. FIFO and other passive approaches have proven to be unstable [13].

Random Early Detection (RED) [14] was the first form of *active* queue management. Here, the network middleboxes had an active role in the decision process of dropping packets. In RED, the current occupancy of the network middlebox is monitored. Then, after a certain threshold, each incoming packet is marked with a probability P of being dropped (either a function of the buffer occupancy or a fixed number). Furthermore, RED does not differentiate between bursty and non-bursty traffic nor establishing or established packets in a connection. One of the major criticisms of RED is the complexity around setting all required parameters.

Another AQM algorithm, Controlled Delay (CoDel) [12] was proposed to address the problem with bufferbloat [10] - it works by repeatedly measuring the RTT of the connections flowing through its buffers. It uses this information to mark a connection as having good or bad "buffering". Here, connections with bad "buffering" are connections likely to experience bufferbloat. CoDel will drop packets from connections with bad "buffering" until their "buffering" stabalizes, thus reducing budfferbloat.

Yet another AQM algorithm targeting bufferbloat is Proportional Integral Enhanced controller (PIE) [11]. Like CoDel, PIE also monitors the latency of the active connections and like RED, PIE applies a packet drop probability for each connection's packets after congestion is inferred. However, unlike RED, PIE detects a connection's congestion based on the inter-arrival time of the connections packet's at the middlebox. Thus, PIE tackles the bufferbloat problem by applying a probabilistic control theory-based approach of when to drop packets. Similar to CoDel, PIE was designed without many adjustable parameters to promote easier use and deployment.

Other works attempt to address bufferbloat by applying reinforcement learning (RL) [15] or machine learning(ML) [16] to AQM. As the Internet is a dynamic constantly changing environment, the ability to predict or model Internet traffic behaviour is an open problem. For example, since the beginning of the 2019 global pandemic, VoIP traffic has increased to previously unprecedented levels [17]. Consequently, high quality training data that accurately captures the multitude of operational conditions found in real networks is almost impossible to provide. Similarly, The data required to train RL or ML-based appraches suffer the same issue. Regarding online learning in the network itself, AQM middleboxes are often resource constrained precluding the use of resource hungry learning applications. For these reasons, learning-based approaches have mostly remained as academic proposals and have not made it to production environments.

Traffic classification is not a novel concept [18]. It has been used for network solutions since the mid 90s. Still, by early 2010s there was no standard way of doing it and researchers based classification on traffic features, protocol and port numbers, and host behaviour [19]. Since then it has been used in many solutions from advanced classification algorithms [20] to intrusion detection systems [21].

Other techniques which can compliment AQM systems

[1]https://datatracker.ietf.org/group/aqm/about/

include transport extensions, such as explicit congestion notification (ECN) [22] or new TCP implementations [23].

IV. FLOQ

In this section we introduce FLOQ, a novel AQM algorithm. Its main goal is to address packet loss during connection establishment and to reduce PLT without sacrificing overall network performance. Influenced by previous work, FLOQ builds upon monitoring connections' properties, as in CoDel and PIE, and applying scaling drop probability, as in RED.

The two main novelties of FLOQ are that a) it partitions its buffer space not by protocol but by the *responsiveness* of the traffic and b) connection establishment packets are prioritised for responsive traffic. Using these traffic categorisations, FLOQ then applies probabilistic admission to arriving packets.

A. Traffic Types

In FLOQ, a responsive traffic flow is defined as one which reduces its sending-rate after packet loss is observed. FLOQ uses inferred network knowledge to determine this. Specifically, for a given connection, FLOQ records the number of transferred packets, the number of packets it dropped, the timestamp of the last dropped packet, and the connection's sending-rate. This information is used in the admission policy, described below. An unresponsive traffic flow is defined as one in which the sending-rate of packets, as observed by FLOQ, is independent of the packet loss, as observed by FLOQ.

Initially all connections are marked as responsive, however, connections are reclassified should they not reduce their sending-rate after their packets are dropped. We do not infer the traffic's responsiveness from the observed protocol number because such information does not guarantee useful insights with respect to responsiveness. For example, consider TCP connections which do not congestion control, or responsive protocols (e.g., QUIC [24]) atop, network protocols not known for intrinsically supporting congestion control.

Additionally, FLOQ further subdivides both traffic types into *establishment* and *other* phases corresponding to the establishment or otherwise of a connection. If a number of exchanged packets on a connection is less than a configurable threshold, the connection is identified as being in an establishment phase. In this work, the threshold is set at 10.

B. Admission Policy

Using the previous definitions of traffic types, FLOQ uses a weighted probability function to admit packets (Eq 1). All its parameters are described in Table I. Additionally, Algorithm 1 shows FLOQ's admission policy in the form of pseudo-code. The admission policy function returns the percentage likelyhood that an incoming packet gets accepted to the buffer queues.

$$P_{in}(rq, uq, t, ep, ap) = \begin{cases} 1 & \text{if } t = \text{RT and rq} < \text{RQ} \\ 1 & \text{if } t = \text{UT and uq} < \text{UQ} \\ F_t(t, ep) * F_d & \text{if } ap > AT \\ 1 & \text{otherwise} \end{cases} \quad (1)$$

Algorithm 1 FLOQ's Admission Policy

```
C : Connection
P ← 0;
if C is responsive then
    t ← "Responsive" if rq < RQ then
        return 100;
    end
else
    t ← "Unresponsive" if uq < UQ then
        return 100;
    end
end
if Quota[T] is full then
    P ← P + F_t(t);
end
if ep < 10 then
    P ← P + F_r;
end
if ap > AT then
    P ← P + F_t(t, ep) * F_d
    return P;
end
return 100;
```

Parameter	Description
rq	is the number of packets in the responsive buffer
uq	is the number of packets in the unresponsive buffer
RQ	is the packet quota for responsive traffic
UQ	is the packet quota for unresponsive traffic
RT	represents responsive traffic
UT	represents responsive traffic
t	is the type of traffic
ep	is the number of exchanged packets for a connection
$F_t(t, ep)$	is the drop probability for traffic type and t with exchanged packets ep
F_d	is a scaling function based on buffer occupancy
ap	total number of packets in **all** buffers
AT	FLOQ's activation threshold

Table I: Admission Policy Parameters and Terms

$F_t(t)$ is the base or default probability of dropping a packet for a given traffic type $F_t(t, ep)$ is $F_t(t)$ with an additional scaling factor based on this connection's already exchanged packets. Different base probabilities are assigned to responsive and unresponsive traffic as FLOQ is designed to drop fewer responsive packets. This choice is made as responsive flows reduce their send-rate in response to packet loss making them more sensitive to packet loss and thus having a higher impact on network traffic overall. In this work, $F_t(t, ep)$ is set to 2% and 5% for responsive and unresponsive traffic, respectively.

To address packet loss of initial connection packets, $F_t(t, ep)$ is increased when the number of exchanged packets (ep) for responsive traffic is larger than the configurable connection setup threshold. This is current set to 10 packets, see Section IV-A.

Finally, FLOQ increases the packet drop scaling factor F_d per traffic type as the number of stored responsive (rq) or unresponsive (uq) packets increase. In this work, F_d is a factor that scales with the total buffer occupancy ap. Furthermore, F_d is additive for responsive traffic, and multiplicative for unresponsive traffic. Finally, F_d is only applied after a traffic

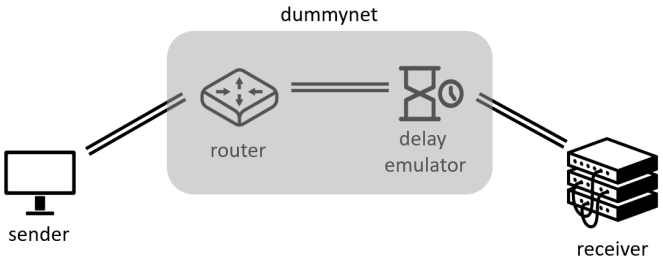


Figure 2: Experimental Setup

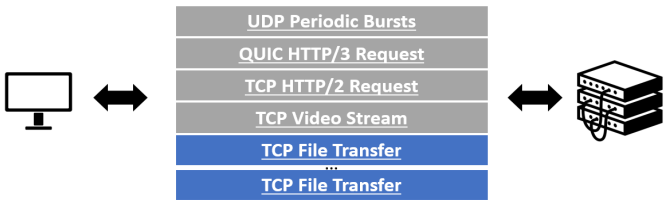


Figure 3: Per-Experiment Traffic Composition

type reaches its pre-defined packet quota (RQ or UQ) and ap is over the activation threshold AT. This is done to prevent FLOQ from dropping packets from a given type when there is available capacity in the other type's buffer. Also, FLOQ further increases the scaling factor F_d if a packet from the responsive traffic would overflow into the unresponsive buffer and vice versa. FLOQ allows such traffic type "overflows", since it can use the remaining "other" quota to store bursts of the traffic type which buffers were overflown. Additionally, the increased F_d for the overflowing traffic and the unconditional admittance policy for the overflown type aims to preserve the preset admission boundaries as much as possible.

Drawing inspiration for managed memory heaps [25], FLOQ partitions the available middlebox memory buffer into three logical domains, as opposed to dedicated memory buffers per traffic type. Here, each domain corresponds to responsive traffic, unresponsive traffic, or other traffic. In order to prevent FLOQ *overfitting* quota sizes to the problem studied, each quota has a soft limit. This means that packets can overflow into the other memory buffers. However, as explained in the previous paragraph, FLOQ ensures that such traffic overflows would not be disruptive to FLOQ's operation in general.

The size of each domain is determined by the quotas RQ and UQ. The final *other* traffic quota is the remaining space after RQ's and UQ's values are combined. In this work RQ is set to 50%, UQ is set to 30%, leaving 20% for the *other* traffic. As the Internet is has more responsive than unresponsive traffic [1], RQ has the most allocated space. Despite concerns that storing larger of amounts of responsive traffic would benefit PLT at the expense of unresponsive traffic's performance, experimental results in Section VI did not confirm this effect. Nevertheless, exploration of the quota ratio is an area of future work. When all buffers are full, FLOQ reverts to FIFO behaviour.

Dropping initial packets is sometimes unavoidable, either due to many initial packets arriving at once and filling the buffers or because the buffers were already at full capacity. In such cases, FLOQ would try to keep the occurrence of this to a minimum by preemptively dropping packets from other connections. When an initial packet needs to be dropped, FLOQ would give subsequent initial packets on that connection higher admission probability. Thus, dropping as few initial packets of the same connection, FLOQ would speed-up that connection's overall completion time and allow for more connections to complete within the experiment duration.

It is important to note that FLOQ introduces no more overhead than state-of-the-art algorithms, such as CoDel and PIE. While FLOQ keeps per-connection state, these are simple counters, comparable to PIE's connection RTT tracking.

V. Experimental Setup

As discussed in Section IV, the main goal of FLOQ is to reduce PLT without sacrificing overall network performance. To explore this, the following experimental setup was used.

A. Testing Configuration

All experiments were performed on a simulated topology where traffic flows from a sender on one side to a receiver on the other via a router and a delay emulator, as shown in Figure 2. FLOQ is implemented as a FreeBSD kernel module to emulate the router and is used in conjunction with dummynet [26] to emulate delay. The router's buffer capacity was set to the widely accepted value of the bandwidth delay product (BDP) plus an additional 20% extra capacity. The extra capacity was added to intentionally introduce some bloated buffer space as CoDel and PIE were designed to operate well in such cases. Consequently, buffer sizes were 20 and 200 packet slots for the 10 Mbps and the 100 Mbps link scenarios respectively. The link scenarios are described below. Furthermore, many provisioned network middleboxes experience some level of bloated buffers [10], this enables exploration of FLOQ's behaviour in this context.

The sender, receiver, router, and delay emulator are all separate processes in a single virtual machine. This VM had two processor cores and 4GB of virtual memory executing on a Intel Core i5-10210U 4 x 1.6 - 4.2 GHz. In future work, a more complicated set of topologies and testbed will be explored.

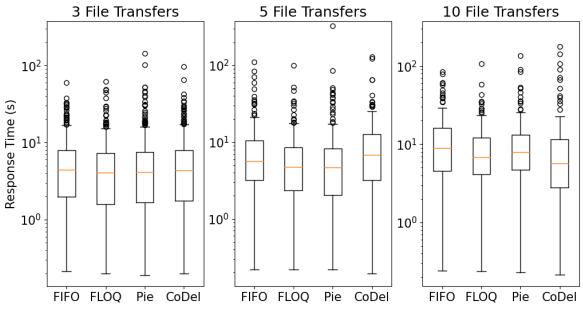
B. Experimental Traffic

For each experiment, various traffic types were used to simulate background network activity and web connections. These traffic types are shown in Figure 3 and discussed below:

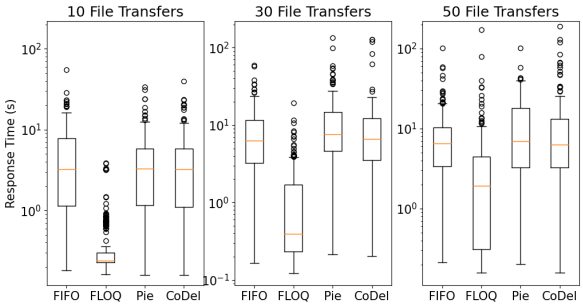
UDP Periodic Bursts: Periodic burst of constant bit-rate traffic. In each of the experimental sets described below, the UDP traffic was configured to use 15% of the total link capacity. For example, UDP traffic was 1.5Mbps with the 10 Mbps link. Such traffic can be representative of VoIP or gaming traffic.

QUIC Web: Sequential HTTP/3.0 requests from a sender to a local copy of Google's index page stored on the receiver. Here, the sender and receiver correspond to the quiche client and server respectively[2]. After a request completes (successfully or

[2]https://github.com/cloudflare/quiche

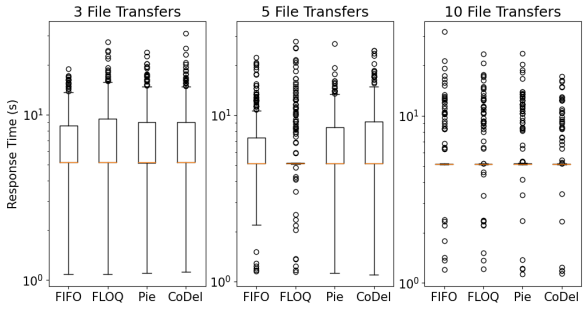


(a) 10 Mbps

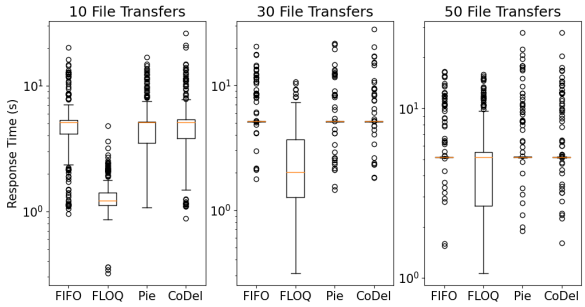


(b) 100Mbps

Figure 4: PLT Distribution for TCP Web Traffic



(a) 10 Mbps



(b) 100Mbps

Figure 5: PLT Distribution for QUIC Web Traffic

otherwise), the client will wait 6 seconds before sending another request. This was done to simulate user browsing behaviour.

TCP Web: Sequential HTTP/2.0 requests to a local copy of Google's index page. Here, the sender and receiver correspond to `wget`[3] and the nginx[4], respectively. After a request completes, the client will wait 6 seconds before sending another request. This was done to simulate user browsing behaviour.

TCP Video Stream: A single adaptive bit-rate streaming application operated for the full duration of each experiment. The video used the 6 second segmented MPEG-DASH dataset[5]. The client used was scootplayer[6] and nginx as the server.

TCP File Transfer: file transfer of an "infinitely large" file.

Based on the above traffic types, we conducted experiments using link bandwidths of 10 Mbps and 100 Mbps. These values were chosen to correspond to representative DSLv2 and FTTC links [27], respectively. Each experiment was run for 600 seconds, as the TCP Video Stream traffic had a hard deadline of just over 600 seconds. No server or network buffer warm-up was performed prior to any of the experiments. This is since, no caching took place and we wanted to monitor how FLOQ operates with the network buffers as their capacity builds up.

In each link scenario, the number of concurrent TCP file transfers was altered to explore the performance of FLOQ under different network loads. As each TCP flow tries to claim as much bandwidth as possible, the pressure on the single (bottleneck) link and the load on the AQM are increased. Section VI describes the performance of FLOQ and the traffic in this context. Our dataset is built using the results from 10 repeated runs for each link capacity/link load/AQM combination. That is, 10 experiments on the 10Mbps link with 3 background file transfers using FLOQ, another 10 using FIFO, etc. Adding up to a total of 240 runs (2 link capacities, 3 link load scenarios, 4 algorithms, 10 repetitions).

[3]https://www.gnu.org/software/wget/

[4]https://www.nginx.com/

[5]https://github.com/camilanovaes/MPEG-Dash-Simulation

[6]https://github.com/broadbent/scootplayer

VI. Results

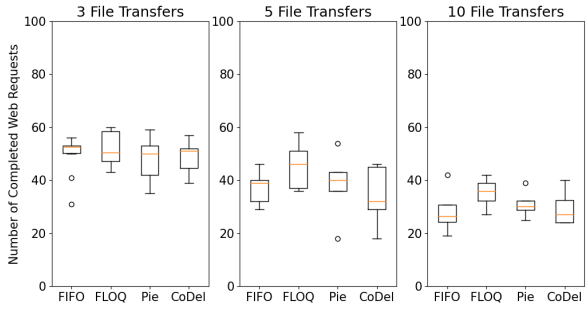
Using the setup described in Section V, we now discuss how FLOQ compared against FIFO, PIE, and CoDel in terms of number of completed TCP and QUIC connections, page load times, jain's fairness index, overall traffic throughput/performance, and UDP loss percentages. PIE and CoDel were configured with their default dummynet[7] settings on a Free BSD 11.3 kernel implementation.

We compare FLOQ with FIFO, PIE, and CoDel as these three algorithms are used to manage the majority of our currently deployed infrastructure and are accepted as standards.

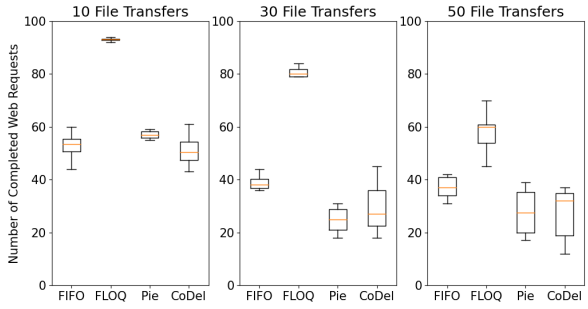
A. Web Browsing (Responsive) Performance

Figures 4 and 5 show the distribution of TCP Web (HTTP/2) and QUIC Web (HTTP/3) page load times across 10 experimental runs. Please note the log scale on the y-axes. As noted in Section V, we use all background traffic and vary the number

[7]https://github.com/luigirizzo/dummynet

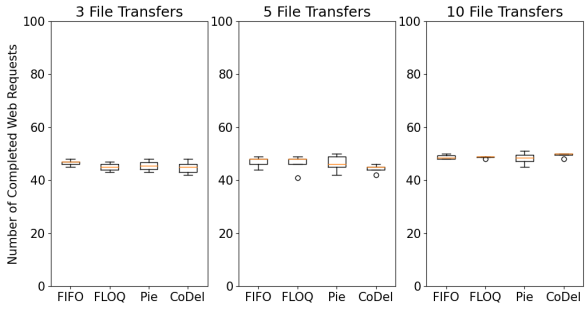


(a) 10 Mbps

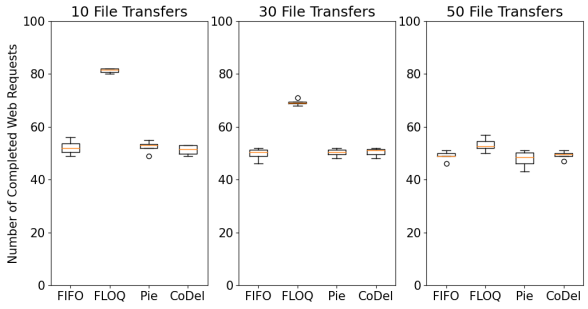


(b) 100Mbps

Figure 6: Completed TCP Web Traffic Requests



(a) 10 Mbps



(b) 100Mbps

Figure 7: Completed QUIC Web Traffic Requests

of TCP file transfers across experiments. For 10 Mbps we use 3, 5, and 10 concurrent file transfers and for 100 Mbps we use 10, 30, and 50 concurrent file transfers.

For both TCP and QUIC Web traffic at 10 Mbps (Figures 4a and 5a) we see that FLOQ performs comparably to the other AQM approaches when comparing the distributions. The only exception being FLOQ's performance on QUIC traffic in the presence of 5 TCP files transfers, Figure 5a. This behaviour is also seen by the other AQM algorithms in the 10 TCP file transfer case. FLOQ exhibits this behaviour earlier as FLOQ starts dropping packets comparatively earlier.

Also, we see that in Figure 4a FLOQ shows comparable performance to the best AQM algorithm. However, we do not see the full benefit of FLOQ as it will only apply to the last 4 packets in the buffer. This corresponds to final 20% of the buffer, as explained in Section V.

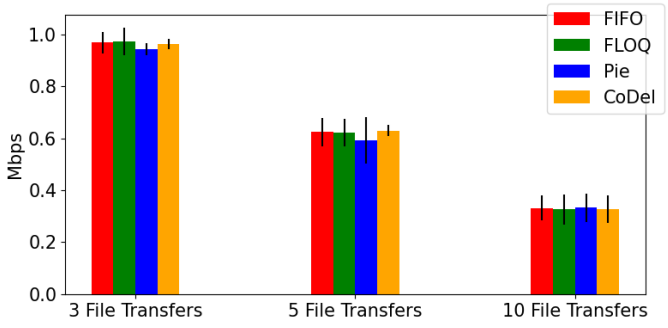
When considering TCP and QUIC Web traffic at 100 Mbps (Figures 4b and 5b) the advantage of FLOQ becomes more clear. Here we see that FLOQ's median response time is lower in five of the six experiments, and comparable for 50 TCP file transfers in Figure 5b. We ascribe the latter case to the heavy load of the link meaning that FLOQ is unable to always preserve the initial packets. However, it can be observed that FLOQ has a wider interquartile range in this case. Inspecting further, we can see that the lower quartile is near the RTO value for a connection and the upper quartile is twice the RTO. This means that when a connection will loose initial packets FLOQ causes this loss to gravitate towards only one or two packets, corresponding to the one or two RTOs.

Given a larger BDP, and hence a larger buffer, FLOQ's initial setup preservation policy works as intended and the hypothesised lower response times are observed. Here we see that FLOQ is able to prioritise a connection's initial packets enabling either a connections' setup and congestion window (CWND) build-up or fast re-transmit to work effectively.

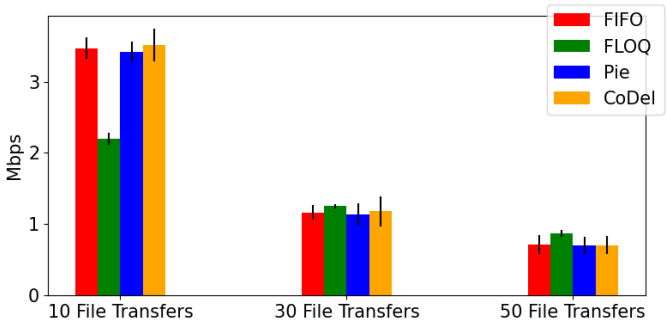
A similar pattern of results can be seen when considering the number of completed TCP and QUIC web requests, across 10 experimental runs in Figures 6 and 7. Again we see that FLOQ performs comparably for the 10 Mbps experiments, but when provided with enough buffer space, prioritising a connection's initial packets leads to an increase in the number of completed web connections in the 100 Mbps experiments. Additionally, Figures 6b and 7b show that for both TCP and QUIC traffic the number of completed connections increases as the response time decreases. Here FLOQ enables more connections to successfully establish without loss, leading to more connections which complete faster.

B. Overall TCP Performance & Throughput

Figure 8 shows the average throughput for all TCP file transfers across the different experiments for 10 runs. The error bars show the standard deviation. When considering the 10 Mbps experiments (Figure 8a) it can be seen that FLOQ achieves comparable throughput to that of connections controlled by other AQMs. However, when comparing the fairness indexes in Figure 9a, FLOQ performs slightly worse.

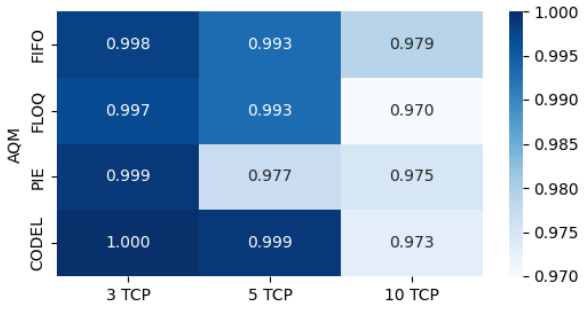


(a) 10 Mbps

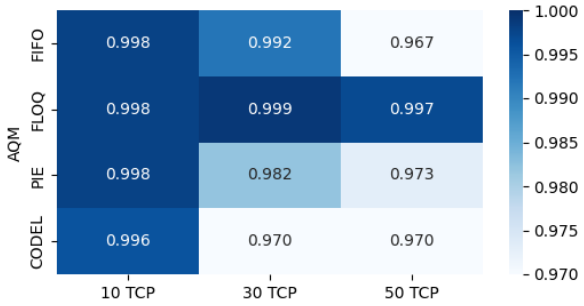


(b) 100Mbps

Figure 8: Average TCP File Transfer Throughput



(a) 10 Mbps



(b) 100Mbps

Figure 9: Average Jain's fairness index

While FLOQ aims to not drop multiple packets from the same connection, the buffer space in the 10 Mbps case was so low that FLOQ would fall back to tail drops. Hence, some connections would experience more loss compared to others leading to decreased speed.

The 100 Mbps experiments yield more interesting results. When considering the fairness index, Figure 9b shows that FLOQ consistently performs equal or better to all other approaches across all experiments. This is because other AQM approaches have no mechanism to balance the lost packets across connections. Conversely, FLOQ's preservation policy is able to manage and sustain high fairness across all connections. Additionally, in this case FLOQ has sufficient buffer space combined with enough packets from different flows to trigger FLOQ's activation thresholds.

When considering the 100 Mbps throughput experiments in Figure 8b, FLOQ is less consistent. When the traffic load is low (10 TCP file transfers), FLOQ controlled queues achieve lower overall throughput compared to others. This is because FLOQ preemptively starts dropping packets. Specifically, as the middlebox's buffer is sufficiently large, FLOQ's activation threshold will make it drop packets earlier compared to the other AQMs. Put simply: when the link is not loaded by enough flows, combined with FLOQ's early drop policy, flows may observe worse performance. However, despite the decrease in the TCP file transfer throughput, Figure 4b shows improved PLT even though both traffic types are TCP and responsive.

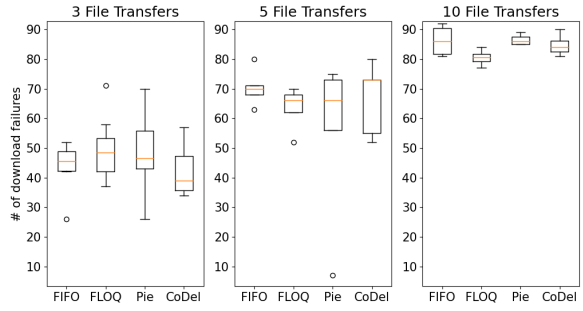
This TCP performance decrease can be addressed by exploring alternate FLOQ configurations on how aggressively it drops packets and possibly by adding the network load as another input to Eq 1. However, this must be balanced against the impact on PLT. We leave this as discussion for future work.

C. Streaming Performance

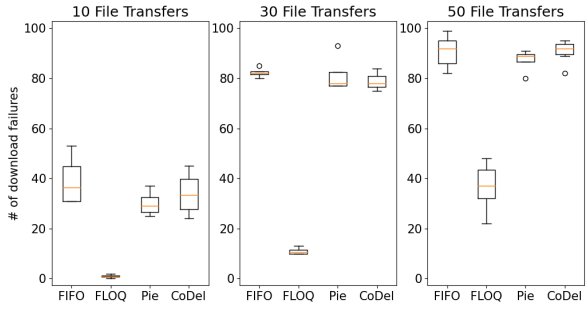
Over 70% of TCP Internet traffic is video [1]. Although the primary goal for FLOQ is to improve PLT, it is necessary to ensure that FLOQ does not disrupt video traffic more than other AQMs. Volume aside, such traffic is important to consider because it has very different operational characteristics compared to file transfer or web traffic.

As the current standard for on-demand video delivery is to pre-encode the video in different representations and then segment each one into equally sized time chunks, video connections typically reuse the same single connection for the duration of video download. Also, such connections experience periods where no data exchange happens due to client buffering. Additionally, as each requested chunk may be of different quality, such traffic is usually *bursty* traffic at irregular intervals.

Figure 10 shows the distribution of how often the video client failed to download a video chunk before it had to be played-out, thus contributing to the playback rebuffering or "freezing". As with other results, the 10 Mbps experiments show FLOQ as comparable to other AQM's. However, for the 100 Mbps experiments, FLOQ's stricter admission policy allowed for more video chunks to meet their respective deadlines, thus reducing the number of chunk download errors.

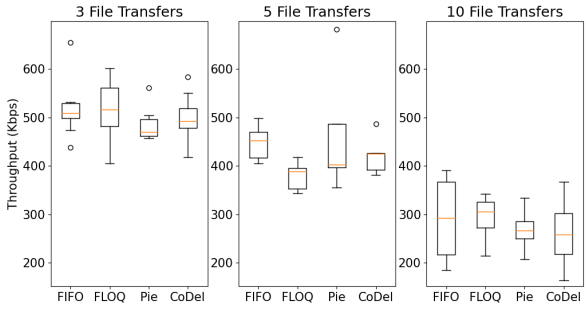


(a) 10 Mbps

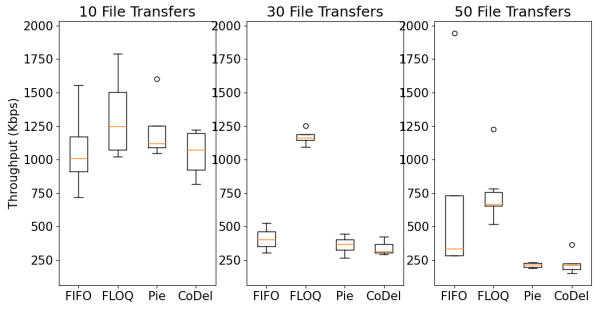


(b) 100Mbps

Figure 10: Failed Video Chunk Downloads



(a) 10 Mbps



(b) 100Mbps

Figure 11: Video Throughput (Render Quality)

Figure 11 shows the distribution of average throughput (rendered quality) for the video stream across 10 runs. Again, FLOQ's performance is comparable for the 10 Mbps experiments. For the 100 Mbps experiments FLOQ is able to achieve higher average video throughput. This is confirms the behaviour observed for FLOQ in Section VI-B.

In summary, FLOQ not only enables fewer download errors but also increases the overall quality of video playback.

D. UDP (Unresponsive) Performance

Figure 12 shows the average percentage packet loss for the UDP periodic bursty traffic using the different AQM approaches across 10 runs. Again, FLOQ behaves consistently with previous results. For the 10 Mbps experiments FLOQ displays comparable loss to the other AQMs (Figure 12a). For the 100 Mbps experiments, Figure 12b shows that for 10 concurrent file transfers, UDP loss increases and for 30 and 50 concurrent file transfers it decreases. As before, we explain the *increase* in UDP packet loss as a result of FLOQ pre-emptively dropping packets to preserve room for a connection's initial packets. We explain the *reduction* in UDP packet loss as a side effect of FLOQ's effective management of the TCP file transfer traffic, as see in Figure 8b. Again, by preemtively dropping TCP file transfer packets, we are able to decrease UDP packet loss as more free capacity is available in the middlebox's buffer.

E. Summary

We have observed that for links with lower capacity, such as DSLv2 lines, FLOQ shows comparable performance to existing AQM approaches. This is since FLOQ would require a certain capacity of the buffer to be filled, in order to start operating. For lower capacity cases, this allows FLOQ to manage too few packets, so that its benefits can be fully demonstrated.

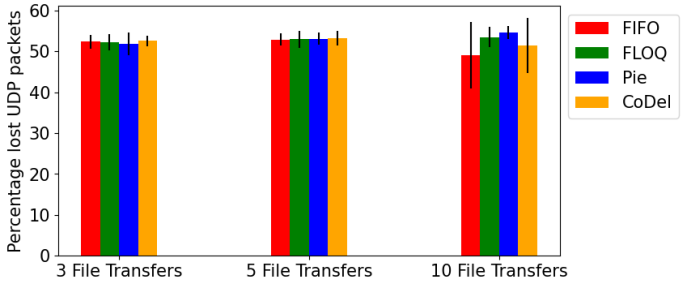
For links with higher capacity, such as FTTC lines, we see that FLOQ's preemptive packet drops, combined with the initial packet prioritisation can - for the majority of cases - both reduce page load times and increase completion rates for TCP and QUIC web traffic, maintain or improve TCP throughput and fairness, improve video streaming performance, and reduce UDP packet loss. These benefits are seen when multiple throughput demanding connections compete on the same link or more generally when the links are busy. For cases where the link is lightly loaded, while FLOQ achieves its primary goal of reducing page load time, this comes at the cost of decreased performance for other traffic, for example, TCP throughput and increased UDP packet loss.

VII. Future Work

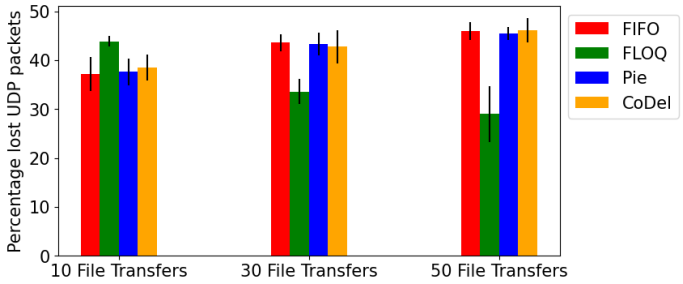
While influenced by anecdotal evidence from Rakuten Mobile's national network, this work has currently been based on synthetic data and topologies. Next steps involve increasing the complexity of the testing environment and studying FLOQ with real network traffic.

Additionally, we are also exploring the autonomous and distributed operation of FLOQ (and more generally AQM) following the approach being proposed in the ITU-T FGAN [28].

We leave testing and deploying FLOQ as scope for future work. However, we note that FLOQ's deployment should be



(a) 10 Mbps



(b) 100 Mbps

Figure 12: Average UDP Packet Loss

relatively straightforward for network operators. Currently, we have implemented FLOQ as a FreeBSD module, and it can be switched enabled or disabled dynamically at the network hardware. Additionally, we note that similar to RED, FLOQ exposes a multitude of configurable parameters, however, our current solution would pick default values based on our internal testing and heuristics. Nevertheless, in future we would also like to explore adjusting FLOQ's parameters during run-time, in the context of autonomous distributed systems.

VIII. Conclusions

This work presents a new AQM algorithm called FLOQ. Unlike other AQM approaches, FLOQ categorises traffic by behaviour, not protocol, and prioritises initial connection packets based on these traffic categorisations. Through experimental comparison against state-of-the-art AQM algorithms, FLOQ's performance across various scenarios ranges from comparable in lightly loaded links, to significantly improved across a range of traffic types for the majority of cases.

IX. Acknowledgements

This work is partially supported by the National Institute of Information and Communications Technology (NICT), JAPAN.

References

[1] Sandvine, "The Global Internet Phenomena Report," 2019.

[2] R. Netravali, A. Goyal, H. Balakrishnan, J. Mickens, and M. Csail, "Polaris: Faster Page Loads Using Fine-grained Dependency Tracking," p. 123, 2016. [Online]. Available: http://x.com/first.js''/

[3] R. Rehrmann, M. Keppner, W. Lehner, C. Binnig, and A. Schwarz, "Workload merging potential in sap hybris," in *Workshop on Testing Database Systems*. Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3395032.3395326

[4] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, *Demystifying Page Load Performance with WProf*, 2013.

[5] M. Al-Fares, K. Elmeleegy, B. Reed, and I. Gashinsky, "Overclocking the yahoo! cdn for faster web page loads," in *ACM SIGCOMM Conference on Internet Measurement Conference*. Association for Computing Machinery, 2011, p. 569–584. [Online]. Available: https://doi.org/10.1145/2068816.2068869

[6] J. Hall, I. Pratt, I. Leslie, and A. Moore, "The effect of early packet loss on web page download times," in *Passive and Active Measurement Workshop, La Jolla, California USA*, 2003.

[7] V. Jacobson, "Congestion avoidance and control," *ACM SIGCOMM Computer Communication Review*, vol. 18, no. 4, pp. 314–329, aug 1988. [Online]. Available: https://dl.acm.org/doi/10.1145/52325.52356

[8] S. Floyd, J. Mahdavi, M. Mathis, and D. A. Romanow, "TCP Selective Acknowledgment Options," RFC 2018, Oct. 1996. [Online]. Available: https://www.rfc-editor.org/info/rfc2018

[9] M. Sargent, J. Chu, D. V. Paxson, and M. Allman, "Computing TCP's Retransmission Timer," RFC 6298, Jun. 2011. [Online]. Available: https://www.rfc-editor.org/info/rfc6298

[10] J. Gettys, "Bufferbloat: Dark buffers in the Internet," *IEEE Internet Computing*, vol. 15, no. 3, may 2011.

[11] R. Pan, P. Natarajan, F. Baker, and G. White, "Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem," RFC 8033, Feb. 2017. [Online]. Available: https://www.rfc-editor.org/info/rfc8033

[12] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar, "Controlled Delay Active Queue Management," RFC 8289, Jan. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8289

[13] M. Bramson, "Instability of fifo queueing networks with quick service times," *The Annals of Applied Probability*, pp. 693–718, 1994.

[14] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, 1993.

[15] H. Fawaz, D. Zeghlache, T. A. Q. Pham, J. Leguay, and P. Medagliani, "Deep reinforcement learning for smart queue management," *Electronic Communications of the EASST*, vol. 80, 2021.

[16] X. Lin and D. Zhang, "Kemy: An aqm generator based on machine learning," in *International Conference on Communications and Networking in China*, 2015, pp. 556–561.

[17] G. Forecast *et al.*, "Cisco visual networking index: global mobile data traffic forecast update, 2017–2022," *Update*, vol. 2017, p. 2022, 2019.

[18] K. C. Claffy, H.-W. Braun, and G. C. Polyzos, "A parameterizable methodology for internet traffic flow profiling," *IEEE Journal on selected areas in communications*, vol. 13, no. 8, pp. 1481–1494, 1995.

[19] H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee, "Internet traffic classification demystified: Myths, caveats, and the best practices," in *ACM CoNEXT Conference*. Association for Computing Machinery, 2008. [Online]. Available: https://doi.org/10.1145/1544012.1544023

[20] U. Prabu and V. Geetha, "Self-organizing deep learning model for network traffic classification," in *Inventive Communication and Computational Technologies*. Springer, 2022, pp. 419–425.

[21] M. Di Mauro, G. Galatro, G. Fortino, and A. Liotta, "Supervised feature selection techniques in network intrusion detection: A critical review," *Engineering Applications of Artificial Intelligence*, vol. 101, 2021.

[22] S. Floyd, D. K. K. Ramakrishnan, and D. L. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," RFC 3168, Sep. 2001. [Online]. Available: https://www.rfc-editor.org/info/rfc3168

[23] B. Briscoe and A. S. Ahmed, "TCP Prague Fall-back on Detection of a Classic ECN AQM," nov 2019. [Online]. Available: http://arxiv.org/abs/1911.00710

[24] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," RFC 9000, May 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc9000

[25] A. W. Appel, "Simple generational garbage collection and fast allocation," *Software: Practice and experience*, vol. 19, no. 2, pp. 171–183, 1989.

[26] L. Rizzo, "Dummynet: A simple approach to the evaluation of network protocols," *SIGCOMM Comput. Commun. Rev.*, vol. 27, no. 1, p. 31–41, jan 1997. [Online]. Available: https://doi.org/10.1145/251007.251012

[27] Ofcom, "UK Home Broadband Performance," Tech. Rep., 2021.

[28] P. Harvey, L. Wong, X. Cao, and X. Song, "FGAN-I-167: High level architecture framework for Autonomous Networks," UN ITU-T Focus Group on Autonomous Networks, Tech. Rep., 2021. [Online]. Available: https://extranet.itu.int/sites/itu-t/focusgroups/an/input/FGAN-I-167.docx