Improving Cloud Gaming traffic QoS: a comparison between class-based queuing policy and L4S

Philippe Graff, Xavier Marchal, Thibault Cholez, Bertrand Mathieu, Stéphane Tuffin, Olivier Festor

Improving Cloud Gaming traffic QoS: a comparison between class-based queuing policy and L4S

Philippe Graff*, Xavier Marchal*, Thibault Cholez*, Bertrand Mathieu†, Stéphane Tuffin†, Olivier Festor*
*Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France, {first.last}@loria.fr
†Orange Innovation, Lannion, France, bertrand2.mathieu@orange.com, stephane.tuffin@orange.com

***Abstract*—Some kinds of application traffic, such as Cloud Gaming (CG), are particularly demanding for a network to transport because they require at the same time a low-latency and a high-bitrate. Quality of Experience (QoE) can quickly deteriorate when the network Quality of Service (QoS) is not met regarding bandwidth and delay requirements. In particular, the competition with some capacity seeking flows may induce a high queuing latency on the bottleneck (buffer-bloat phenomenon).**

In this paper, we evaluate two network level solutions that allow CG traffic to be processed in specific queues, but exhibit different operational constraints. The first solution uses a class-based queuing policy (*Hierarchical Token Buckets*, HTB), which requires prior traffic classification and some traffic engineering. The second solution leverages the new *Low Latency, Low Loss & Scalable Throughput* (L4S) architecture and the *DualPI2* Active Queue Management (AQM), but it needs the application support.

We perform extensive measurements on an experimental CG platform that integrates the L4S-compliant SCReAM CCA and that we made to evaluate both approaches regarding their QoS enforcement capability and fairness against different competing flows that are driven by TCP CUBIC or BBRv2. We show that both solutions succeed to preserve the QoS of CG traffic.

***Keywords*—Low Latency, Quality of Service, Active Queue Management, Bufferbloat, L4S, Cloud Gaming, SCReAM**

I. Introduction

Networks are designed to provide maximum bandwidth while handling packet bursts. Therefore, large buffers are by default included in network devices to keep the pipe full. That is particularly true for cellular networks such as 4G/5G where buffers help to cope with their varying capacity nature. However, filling these buffers can cause high queuing delays, known as the bufferbloat phenomenon [1], that affect latency-sensitive applications and significantly reduce their Quality of Experience (QoE) when the network capacity is challenged. For instance, the increasingly popular Cloud Gaming (CG) service requires at the same time a high bandwidth to transport real-time encoded high-definition multimedia flows at high frame rates, and a low latency to have the game react quickly to user inputs, making it particularly difficult for the network to transport. For instance, this implies a very short buffer size to reduce playout delays but which prevent to retransmit lost packets. Bufferbloat becomes the main component of the latency chain of CG when network resources are limited, accounting for several dozens of milliseconds. To find a trade-off between bitrate and delays, applications must rely on the configuration of their own congestion control algorithms (CCAs). In a previous paper [2], we assessed how four prominent CG platforms adjust their traffic in response to network constraints. One of them does not react to latency increase but only to packet drops, which leads to prohibitive delays in some conditions, while others may react too late to a capacity variation, or sometimes overreact, which also deteriorates the QoE. This shows that pure application-level mechanisms are not sufficient alone to maintain traffic Quality of Service (QoS) in some cases.

Traditional AQM systems apply a single traffic management policy irrespective of the differences in delay-throughput objectives of applications. Without a way to decouple different types of traffic at the network level, the competition with non-collaborative flows (loss based) leads inevitably to bufferbloat. The main motivation of our work is thus to evaluate systems allowing the management of distinct delay-throughput tradeoffs for CG traffic and capacity seeking traffic. This paper focuses on the evaluation of two network solutions that incorporate distinct queues to avoid applications where congestion control is unaware of queuing delays to fill the same buffer used by latency-sensitive applications, and thus enhance the QoS of traffic like CG. The first solution is based on the queuing discipline *HTB*, which considers traffic classes, determining the scheduling of packets at the bottleneck [3]. However, prior traffic classification and accurate traffic engineering is required. Our previous works have successfully addressed this concern by accurately identifying CG traffic at line rate [4], [5]. The second solution leverages the Low Latency, Low Loss, Scalable Throughput (L4S) architecture and the *DualPI2* AQM, which only considers two traffic classes and does not require prior traffic classification [6]. By setting the 2 bits of the IP Explicit Congestion Notification (ECN) header, an application can state whether its traffic is *classic* or *scalable*. *scalable* traffic is forwarded to the *L4S* queue, guaranteeing low delays and minimal loss. However, this approach also imposes an additional constraint, requiring application support through the implementation of a *scalable* CCA compliant with L4S requirements. L4S being a recent technology, it is not leveraged by commercial CG platforms to date. So we designed our own experimental CG platform to fully master the network aspects. Its bitrate is managed by SCReAM, an open-source and *scalable* CCA specifically designed for real-time services [7]. To the best of our knowledge, no previous study has addressed the QoS of real low-latency traffic transported over L4S and especially of RTP-based applications. Previous

research either relied on simulation [8] or TCP Prague.

We conduct several experiments to evaluate the impact of the aforementioned network solutions on traffic QoS and fairness. It involves generating real CG traffic and synthetic concurrent traffic (with iperf) and to observe their interactions at the network bottleneck. The two main CCAs, `Cubic` and `BBRv2`, alternately handle the competing traffic. For both TCP and CG traffic, we monitor the 3 following metrics: bitrate, queuing delay, and loss rate. We also conduct some experiments with a simple *droptail* queue to define the baseline.

To sum up, our contributions are the following:

- we conduct the first study comparing a class-based queuing policy and L4S to improve the QoS of high-bitrate and low-latency traffic by preventing bufferbloat;
- we conduct the first study evaluating L4S against a real low-latency application traffic, in our case Cloud-Gaming;
- we consider the SCReAM CCA and show its ability to properly drive CG traffic;
- we provide an open-source and modular Cloud-Gaming experimental platform that can be used to further investigate the network aspects of CG.

The remainder of this paper is organized as follow. Section II presents the related work on queuing disciplines and *scalable* CCAs. We then present our experimental CG platform in Section III and we assess its proper behavior on emulated cellular network conditions. Section IV presents our evaluation of network solutions to preserve the QoS of CG traffic. In Section V, we further discuss the advantages and disadvantages of each approach. Finally, Section VI concludes this study and introduces future work.

II. Related work

It is widely known that network congestion can cause additional delays, which can undermine QoS. The primary reason of network congestion is the filling of large network buffers, commonly referred to as the "*bufferbloat*" phenomenon [1]. Empirical studies have established that downstream transfer of Cloud Gaming traffic exerts a statistically higher impact on QoE than upstream transfer. Jarschel & al. [9] contend that downstream packet loss and delays have the most profound influence on QoE. Consequently, it is imperative to maintain these factors at an appropriate level to deliver high-quality services to end users.

One approach to address the *bufferbloat* phenomenon is by employing Active Queue Management (AQM) mechanisms. AQMs, such as RED, Codel, or PIE play a vital role in notifying endpoints when the congestion level exceeds a significant threshold. These systems are designed to send congestion signals in proportion to queue occupancy [10], [11], [12]. They thereby enable the endpoints to take preventive measures to avoid network congestion. While AQMs typically discard packets to indicate network congestion, it is also possible to send an explicit congestion signal (`ECN`) by setting a codepoint in the IP header [13]. However, the endpoints must comply with this kind of signal.

In [14], Alizadeh & al. introduce *DCTCP*, a CCA that leverages ECN to detect congestion. Unlike most CCAs, DCTCP adjusts its throughput based on the fraction of congestion signals, which allows for a balance between low queue length and high throughput. DCTCP is typically limited to data center environments due to its aggressive nature compared to traditional CCAs. Therefore, Bondarenko & al. recommend implementing two separate queues at the bottleneck to handle different types of traffic [15]. This approach allows for a smooth coexistence of *scalable* and *classic* flows. The *DualPI2* AQM is based on this idea and ensures that neither queue experiences starvation [6]. The congestion signals sent by each queue are tailored to the type of traffic it handles. *L4S* ("*Low Loss Low Latency Scalable throughput*") designates the queue associated with the *scalable* traffic.

The "*Prague L4S requirements*" offer guidance to both *L4S* transports and network elements [16]. These requirements emphasize the need for a *scalable* CCA that reduces its dependence on RTT and controls its bursts to be "*L4S-capable*". This list is not comprehensive, and readers can refer to RFC 9331 [16] for further details. TCP Prague is an adaptation of DCTCP to *L4S* [17]. The project contributors maintain a list[1] of CCAs that are compliant with the L4S standards, including TCP Prague and Self-Clocked Rate Adaptation for Multimedia (SCReAM) [7], another IETF experimental standard. Even if some other CCAs have been designed with applications such as Cloud Gaming in mind, like SQP [18] or C3G [19], we selected SCReAM because of 1) its support of L4S, 2) its support of the Real Time Protocol (RTP) that is the main transport protocol used by CG platforms (most of the time along webRTC) [2], 3) its ability to accommodate cellular network conditions and 4) its documentation and open-source nature. Even if SCReAM was originally developed for conversational video over normal LTE networks, it appears to be versatile enough and well-suited for CG as well, as we will show. SCReAM relies on the Real-time Transport Control Protocol (RTCP), the companion protocol of RTP, to transport the feedback of the network to the sender through the receiver. This feedback is used to compute the bytes in flight and to measure the One Way Delay (OWD).

The two closest works to our study are [20] and [8]. In [8] Brunello et al. evaluate the benefit of using L4S over 5G to transport a high-rate, latency-critical application. They also use SCReAM but their approach is completely different because their results are based on a proprietary network simulator, while we generate and process real application traffic. Also, contrary to us, they do not consider alternative queuing discipline approach like HTB, nor the competition with other flows driven by other CCAs. This last point is precisely the purpose of [20]. Xu et al. measure how three commercial CG platforms respond to competing TCP Cubic and TCP BBR flows on a congested network link. The fundamental difference is that they only consider a drop-tail queue while we want precisely to investigate more advanced network solutions.

[1] https://l4steam.github.io/

It is worth to note that they witnessed different behaviors between platforms regarding the competing flow. In some cases, one platform may allow the competing flow to have twice the fair bandwidth amount, while in other cases, either another platform or the same platform may do the opposite to another competing flow, resulting in unbalanced traffic sharing configurations. Another noteworthy finding is the correlation between large bottlenecks queues and adverse effects on gaming systems: 1) increased delays, undermining QoE, 2) prolonged response/recovery times to competing flows. This problem is even more pronounced when competing with TCP Cubic, a loss-based CCA, whereas TCP BBR contains the extent to which the bottleneck queue grows. This motivates our approach for distinct queues.

III. Experimental Cloud Gaming Platform

A. Software architecture

In this section we present an experimental Cloud Gaming Platform (eCGP)[2] we have developed in order to master every aspect of the CG use case: from the server to the client, including the network part and the choice of the congestion control algorithm (CCA) in place. It is based around the FFmpeg API that contains a lot of tools to process audio and video streams and exposes an unified API for a large variety of encoders. It shares similarities with GamingAnywhere [21], the sole open source CG platform available at the time we started our study. However, its current utility is constrained by the reliance on deprecated APIs and the use of RTSP over TCP, which does not accurately reflect the practices of modern CG platforms that predominantly use RTP on top of UDP as the transport protocol [2]. The necessity to make numerous adjustments, coupled with the relatively simplistic macro logic, prompted us to develop our own platform tailored to our requirements around CCA, but drawing inspiration from this existing framework.

We first present the general architecture of the client and server before describing how we included the *SCReAM*(v2) CCA in the platform as a proxy service for a better modularity.

1) CG server: The platform is developed for a GNU/Linux operating system and works in a similar way to a remote desktop protocol (RDP). A client connects to the server and performs actions on it remotely. It receives audio and video streams in return. These flows result from captures made on the server.

The `X11` windowing system is used to make the video stream by enabling to record the rendered screen at a given sampling rate. Raw video frames (`AVFrame`) are generated. It is illustrated by the *VideoGrabber* and *Encoder* components in Figure 1. An intermediate processing is made before the encoding to adapt the frames to the format expected by the encoder in use. In particular, two popular encoders are available: `H.264` and `H.265`, but any other could work as long as they have a known RTP payload format (packetization).

[2]https://github.com/mosaico-anr/eCGP

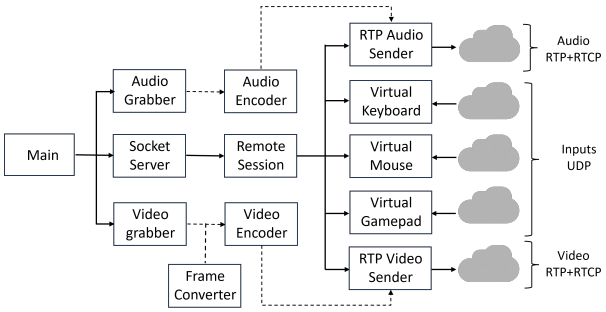

Fig. 1: Functional architecture of our CG server

The encoder can also use *NVENC* to take advantage of GPU hardware encoding on Nvidia discrete graphic cards. It offers a fast encoding and great reactivity to on-the-fly bitrate changes.

Audio follows a similar pipeline but is captured by the `ALSA` software, which is part of the Linux kernel. It provides an API to interface with the sound chipset driver. The *Audio-Grabber* component generates audio frames that are directly passed to the *Encoder*. The `Opus` format is used to encode audio frames. It is very efficient and widely used in real-time communications.

The last component manages inputs and is illustrated in the center of Figure 1. First, clients need to connect to a server socket via `TCP` to get a session number that will allow to identify operations related to each client. There are two types of operations: platform commands and game inputs.

- *Game inputs* represent a user's actions on the controller(s) and are conveyed in a *JSON* format transferred over `UDP`. Three different components manage game inputs depending on the peripheral in use and are illustrated on the right part of Figure 1:

- *VirtualKeyboard*: manages user's actions of the keyboard;
- *VirtualMouse*: manages user's actions of the mouse;
- *VirtualGamepad*: manages user's actions of the gamepad.

Those three components write the received commands in `/dev/uinput`, a kernel module managing virtual devices.

- *Platform commands* manage the signaling between a client and the server. They are transmitted over `TCP` using the same socket that was used to initiate the connection. More precisely, *SDP(Session Description Protocol)* messages are sent to describe multimedia sessions. They tell the client to which port are sent audio and video flows. The other main function is to implement a *RTP Sender* that will transmit encoded audio or video flows to the client thanks to the RTP+RTCP protocols that are specifically designed to transport multimedia flows with low latency requirements. RTP is used in most commercial Cloug Gaming platforms (Google Stadia, Microsoft Xbox Cloud Gaming, Nvidia GeForceNow, Amazon Luna) through webRTC sessions.

2) CG client: The CG client architecture is depicted in Figure 2. Connections are managed by the *Command Socket* component. The platform commands manage the configuration of the two *RTP Receivers* that are linked to their decoders

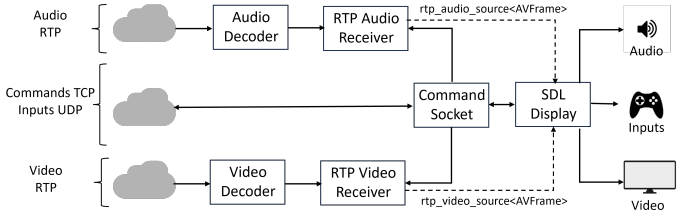

Fig. 2: Functional architecture of our CG client

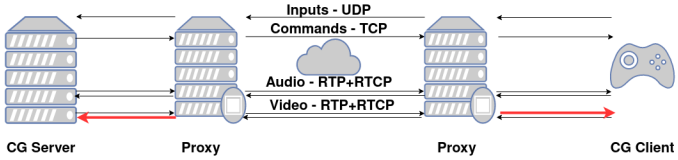

Fig. 3: CG platform's flows

(Audio or Video). Decoders create the frames from the raw data present in the buffer. The resulting frames are given to the *SDL Display (Simple DirectMedia Layer)* component, a library that offers an abstraction of the underlying hardware and allows to play audio and video tracks.

Game inputs are also managed by SDL. There is a polling of system events that are filtered to select only the relevant entries. A *JSON* file is created and gather all events recorded over the sampling period. It is then sent to the server by the *Command Socket* component that directly uses UDP.

3) Congestion Control with SCReAM proxy*:* Our experimental Cloud Gaming Platform needs a way to adapt the traffic throughput to the actual network conditions. We chose to rely on the SCReAM CCA developed by Ericsson[3] to drive packet sending. But in our will to have a modular platform to experiment different network components, we did not integrate directly SCReAM source code but made a proxy to use it. Two local SCReAM proxies are thus deployed on the same computer than the client and the server to manage their respective sending rate. The point of the proxy design is to be able to easily use an alternative CCA by defining interfaces without needing to integrate the CCA code directly into the code of the platform.

a) Server-side SCReAM proxy: The server-side SCReAM proxy is by far the most critical because it handles high-bitrate multimedia flows. The CG server interacts with the local SCReAM proxy through the loopback interface (`lo`). The proxy handles the six flows composing a CG session through dedicated sockets that can be of four types.

- *UDP Socket*: to send or receive UDP segments;
- *TCP Client*: to send or receive TCP segments;
- *TCP Server*: like *TCP Client* but does not initiate a TCP connection;
- *SCReAM*: to send or receive RTP packets with congestion control.

Pre-defined port numbers are used for each flow. One must note that only the RTP video flow is actually processed by the SCReAM algorithm. This is because it is by far the most significant flow and the only one that can truly benefit from congestion control to adapt its bitrate by adjusting the video codec parameters (mostly through the quantization factor that defines the level of video compression). The other flows going through the SCReAM proxy are just transferred untouched to the corresponding socket of the client.

The reception sockets extract the payload of packets and pass it to the corresponding sending sockets through a queue. The two TCP sockets (*TCP Client* and *TCP Server*) used by the proxy are full duplex. They handle the signaling traffic between the client and server.

The SCReAM component is connected to the client to send the video flow and receive its feedback. The packets sent by the CG server are placed in a *FIFO* queue which reading depends on the network conditions. The sending of packets is driven by SCReAM's CCA, according to Ericsson's specifications [7]. It takes into account the length of the RTP queue and the periodic RTCP feedback from the client, and of course ECN network feedback when available. Thanks to this feedback SCReAM can monitor the network delay, packet loss and the state of the bottleneck. The CCA finally defines a target bitrate to meet the current conditions and that will be given to the video encoder. In particular, SCReAM can issue two commands:

- *Bitrate Request*: a new target bitrate defined for the encoder;
- *I_Frame Request*: a new I_Frame is recommended to prevent the propagation of visual artefacts when a packet loss is detected.

Theses commands are translated to a format compatible with the actual encoder by the *Bitrate Converter* component and sent to the server through the loopback. Finally, these instructions are processed on the server by a *remote_session* class that drives the video encoder.

b) Client-side SCReAM proxy: Like the server-side proxy, most flows are just transferred to the client by the client-side proxy. Only the video stream is processed specifically. When a packet is received, the SCReAM component forwards the UDP payload to the *RTP Converter*. It extracts some information and computes statistics on the received video flow and periodically creates a RTCP report that is sent back to the server. In particular, the value of the `ECN-CE` bits in the received packets that denotes an occurring congestion on the path is extracted and is considered by the CCA.

A global view of the CG platform's components and network flows is given in Figure 3.

B. Assessment of our experimental CG platform

Although designed to be a CCA suitable for real-time interactive media over RTP, notably video, SCReAM has never been tested with real CG traffic. In this section, we evaluate

[3]https://github.com/EricssonResearch/scream

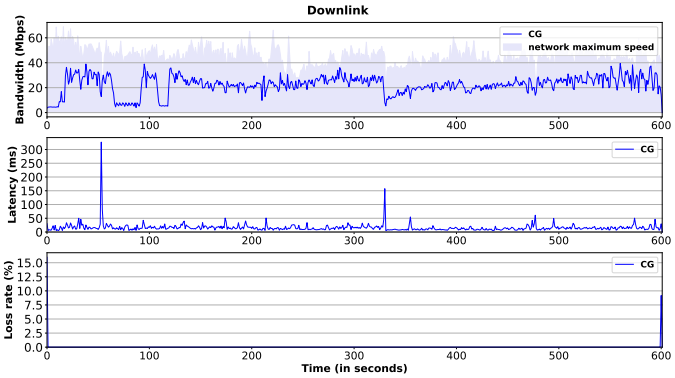

Fig. 4: SCReAM performance on an emulated cellular network

our CG platform to assess if the SCReAM CCA behaves as expected. For this, we consider a demanding network condition, i.e. a 4G cellular network in which the end user capacity can quickly vary over time.

Our testbed is composed of one CG client, one CG server and one router in-between. For all the experiments in the paper, the CG server is configured to use the H265 hardware accelerated codec provided by Nvidia (NVENC) and the video stream is set to 1080p at 60 fps. The most notable additional parameters are the P4 preset with ultra low latency tune and zero latency mode and a CBR rate control. The router is responsible for emulating mobile network conditions. For this, we use the *Mahimahi* linkshell tool[4] which replays the transmission opportunities (txops) as we captured them from a real cellular network from a national operator[5]. Traffic is captured by *wireshark* twice, as packets enter and leave the bottleneck router. This allows us to compute the queuing time for each packet and the loss rate.

Fig. 4 plots the evolution of throughput, delays, and loss rate. We notice a sudden drop in link capacity, more precisely a short loss of connectivity that often happen in cellular networks, around t=50s, leading to an increase in latency (>300ms) as packets can not be transmitted during that time. This increase in latency triggers a strong response from SCReAM, causing an immediate bitrate decrease. Despite the resurgence of transmission opportunities (`txops`), the CG bitrate fluctuates around 7 Mbps for about 20 seconds. Nevertheless, the bitrate recovering seems quicker after the second ($\approx$ t=105s) and third ($\approx$ t=330s) bandwidth decrease events. This can be explained because in those two cases latency is less impacted. The length of the two plateaus around 7 Mbps might be due to the encoder rather than the CCA. Indeed, it is worth noting that the bitrate ultimately depends on the transmitted video. The encoder can take some time to adapt to the bitrate request given by SCReAM. With the exception of two punctual latency spikes due to the aforementioned short disconnections, observed delays and packet loss are fully compatible with a good gaming experience.

[4]http://mahimahi.mit.edu/

[5]https://cloud-gaming-traces.lhs.loria.fr/cellular.html

TABLE I: Queuing delays measured over a cellular network trace for different CCAs

	No CCA	SCReAM CCA	GFN CCA
Average Latency	47.9ms	15.98ms	-
% of packets > 10ms	99.8%	73.9%	52.4%
% of packets > 20ms	85.2%	18.4%	30.4%
% of packets > 50ms	21.1%	1.5%	10.0%
% of packets > 100ms	6.8%	0.3%	2.6%

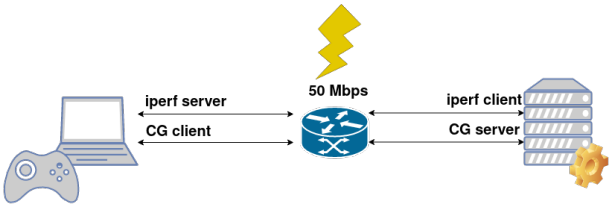

Fig. 5: Testbed SCReAM CG vs concurrent traffic

Table I summarizes the latency statistics measured on the CG platform with and without SCReAM CCA. We have also included the numbers of the GeForce Now (GFN) CG platform from Nvidia. Unfortunately GFN being closed-source, the CCA it uses is unknown but our evaluation showed [2] that it is the most capable to adapt its bitrate to network conditions, offering a good tradeoff between a quick adaptation to network events without overreacting. In the absence of CCA, our platform maintains its bitrate at 30 Mbps, regardless of network conditions. Enabling SCReAM reduces the average latency from 47.9ms to 15.9ms. Actually, some outliers hurt the average latency (the two spikes resulting from a loss of connectivity). In the case of SCReAM, only 0.3% of packets exceed the 100ms threshold above which the experience is considered non-acceptable for gamers, when it reaches 6.8% in the absence of CCA. In terms of delays, SCReAM's performance is even better than GFN's for the last 3 thresholds. Whatever the latency threshold considered, our platform always has fewer packets exceeding it than GFN, proving that SCReAM is capable to properly drive a CG application. This shows that SCReAM(v2) is already very capable to drive CG flows and could be a good candidate to build an open-source CCA for CG with some fine tuning of its internal parameters.

IV. Evaluation of Network Queuing Solutions

A. Testbed and experimental protocol

The main goal of our study is to evaluate to what extent network queuing solutions can help to improve the QoS of CG traffic when it is in competition with other flows at the bottleneck. In particular, we consider a class-based queuing policy, HTB, and the L4S architecture when processing CG traffic alongside flows driven by two of the most widely used CCAs today, TCP Cubic and TCP BBRv2.

The testbed is shown in Fig. 5[6], where all flows (CG and concurrent ones) pass through an intermediate router which

[6]It should be mentioned that the iperf client is the entity sending the data and the iperf server the one receiving it.

applies the desired link characteristics via tc-netem rules. The client runs on a laptop and the server on a workstation with a GeForce RTX 3060 to use Nvenc. They are connected to the router through a 1Gbps ethernet interface. We add a fixed RTT delay of 30 ms (15 ms in each direction) between the CG client and CG server, so that our tests are more representative of a real platform operating on the Internet. We limit the link capacity to 50 Mbps, a value close to the average bandwidth of a 4G network [22].

TCP iperf is used to generate concurrent TCP traffic driven by either Cubic and BBRv2 CCAs. BBRv2 was preferred because BBRv1 is known to have fairness issues by not reacting adequately to loss-based congestion signals sent by L4S in the best-effort queue. For a fair comparison with Cubic, we disable ECN support of BBRv2 to have its traffic handled by the BE queue. The TCP iperf target throughput is increased every minute, so that the flow rates have time to converge. It takes the following values: 10Mbps, 20Mbps, 30Mbps, 40Mbps, 45Mbps. This is materialized in Figures 6-11 by red vertical lines.The concurrent flows are started one minute after the CG traffic. This first "warm up" minute is not considered in the statistics. Each experiment is reported after a single experimental run. Indeed, our point in the paper is to evaluate the impact of queuing policies with real CG traffic. The drawback is that, for each experiment, someone must actually play a game on the CG platform. This part cannot be easily automated.

At the end of the experiments, we analyse 3 QoS metrics that have been proven [23] to be highly correlated with CG's QoE: the *throughput*, the *queuing latency* and the *loss rate*. We think that it is a good approximation of the QoE which requires a lot of work to be performed properly by applying a methodology limiting human bias, which is beyond the scope of this paper. Also, low level metrics are easier to interpret to understand the effect of underlying network mechanisms.

In the following figures, we draw the results for CG in blue solid lines and for Iperf in orange lines with circle markers. The following subsections evaluate in turn the 3 queuing policies: drop-tail, HTB and L4S. We only consider the high bitrate server-to-client traffic that can actually be driven by a CCA. The client-to-server traffic instead has a negligible bitrate being mostly made of user's inputs, plus it's bitrate cannot really be driven by a CCA. In case of loss, it is admitted that the player must simply redo the inputs. For each type of traffic and queuing policy considered, Table II gives additional information about the distribution of the latency and bitrate metrics, the average loss rate and the link utilization. Please note that the statistics are calculated only when the bottleneck is saturated and the flows are actually competing for bandwidth (from the third minute and onward).

B. Baseline: the drop-tail scenario

The first policy is a simple drop-tail queue, the basic strategy of most network devices, with a buffer size of 250 packets. When the queue is full, packets are simply dropped.

1) SCReAM vs Cubic: Fig. 6 shows the behaviour of SCReAM when competing with TCP Cubic traffic. We see that the link begins to be congested from second 120. The TCP flow then has a target throughput of 20 Mbps, while the CG traffic has been slightly lowered of a few Mb/s to stay below 30 Mb/s. Subsequently, TCP steals gradually the bandwidth, each increase of target throughput is granted to the detriment of CG traffic. The queuing delays affecting the the two traffics are identical (average 44ms, cf Table II) because they share a unique queue. The third graph reveals the total absence of losses. It is for this reason that iperf scrupulously respects its target bitrate instructions. In the absence of a congestion signal, Cubic can transmit without restriction. Conversely, the increase in queuing times causes SCReAM to react, by decreasing its bitrate down to the minimum and thus by degrading the video, which causes unacceptable QoE. The penalty is even double because even with such a minimal bitrate, latency can reach high values.

2) SCReAM vs BBRv2: Fig. 7 shows that the cohabitation between SCReAM and BBRv2 is better than with Cubic, with a more equitable bandwidth sharing. Indeed, even if BBRv2 takes over SCReAM, it leaves more room to SCReAM than Cubic. According to Table II, the first quartile of throughput is 9.69 Mbps. This is two times more than what was measured with Cubic (4.29 Mbps). The flow bitrates also seem to stabilize from the 4th level. The value of 40 Mbps (second 240) generates a final increase in TCP throughput. This increase nevertheless generates network delays. In the absence of losses, the increase in delays prevents BBRv2 from further increasing its throughput. The packet queuing time oscillates around 35ms, which, added to the 30ms RTT and the time taken to encode and decode video frames, is very close to the limit of 100ms above which the service is not considered responsive. Yet we can clearly see the benefit of BBRv2 considering delays to evaluate congestion.

In real life, the nature of competing flows can not be controlled. Bandwidth sharing depends on the characteristics of the bottleneck and the CCAs used. This section showed that the distribution of resources can be very unfair and is detrimental to CG flows in both cases. Despite the bitrate of CG traffic decreasing to an unacceptable extent, the bufferbloat issue we want to address still clearly arises in this drop-tail scenario with 85.54% of CG traffic suffering from an additional queuing latency above 20 ms when facing Cubic flows, and 68.88% for BBRv2 flows, with some notable spikes above 80ms in the former case. For this reason, Turkovic & al. advocate the creation of traffic classes [24]. According to them, applications with low latency constraints must be isolated from the other flows. Thus, in the next section we evaluate the HTB queuing discipline.

C. Hierarchical Token Bucket (HTB) queuing discipline

Our Hierarchical Token Buckets (HTB) configuration is based on the work [25]. We create two traffic classes: CG and Best Effort (BE), with CG having the higher priority. We guarantee a minimum throughput of 10 Mbps for CG traffic. This

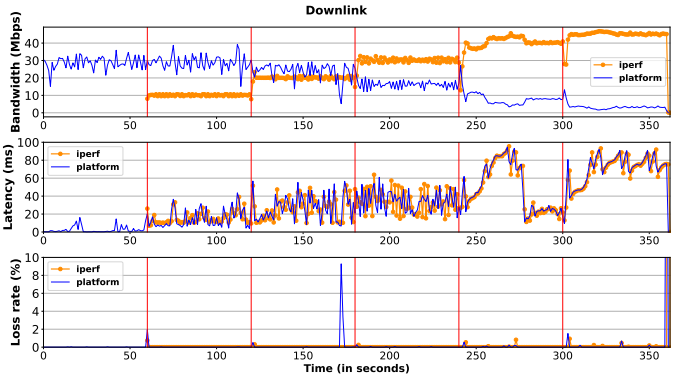

Fig. 6: Drop-tail : SCReAM vs Cubic

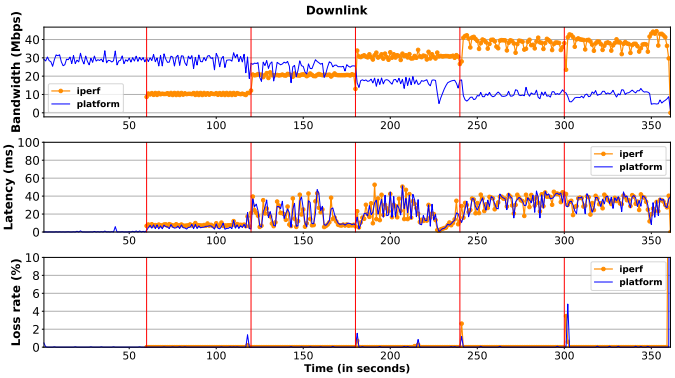

Fig. 7: Drop-tail : SCReAM vs BBRv2

TABLE II: Cloud Gaming (SCReAM) vs TCP Iperf (Cubic or BBR) with the 3 queuing disciplines

		Droptail		HTB		L4S	
		Cubic	BBRv2	Cubic	BBRv2	Cubic	BBRv2
CG Traffic	Avg Latency	44.9 ms	26.5 ms	1.16 ms	0.61 ms	0.53 ms	0.31 ms
	Variance	24.34	11.93	0.92	0.8	0.99	0.51
	% > 10 ms	96.28%	84.23%	0%	0%	0%	0%
	% > 20 ms	85.54%	68.88%	0%	0%	0%	0%
	% > 50 ms	34.71%	0.41%	0%	0%	0%	0%
	% > 100 ms	0%	0%	0%	0%	0%	0%
	Avg Bitrate	12.83 Mbps	15.18 Mbps	28.64 Mbps	28.66 Mbps	24.25 Mbps	21.66 Mbps
	1st quartile	4.29 Mbps	9.69 Mbps	27.22 Mbps	27.63 Mbps	21.83 Mbps	19.78 Mbps
	2nd quartile	12.17 Mbps	12.6 Mbps	29.14 Mbps	28.9 Mbps	25.06 Mbps	22.32 Mbps
	3rd quartile	18.81 Mbps	18.98 Mbps	30.78 Mbps	30.23 Mbps	27.52 Mbps	24.34 Mbps
	Loss Rate	0.25%	0.33%	0.24%	0.05%	0.08%	0.1%
TCP Traffic	Avg Latency	44.45 ms	25.96 ms	99.78 ms	67.38 ms	4.68 ms	7.57 ms
	Variance	24.57	12.31	26.84	20.91	4.03	5.67
	% > 10 ms	99.18%	84.71%	99.18%	99.17%	6.75%	25.42%
	% > 20 ms	82.3%	66.53%	99.18%	98.76%	0.84%	4.66%
	% > 50 ms	35.8%	0.83%	96.3%	81.41%	0%	0%
	% > 100 ms	0%	0%	48.15%	2.48%	0%	0%
	Avg Bitrate	33.34 Mbps	31.66 Mbps	19.56 Mbps	19.28 Mbps	16.43 Mbps	21.68 Mbps
	1st quartile	21.14 Mbps	21.73 Mbps	18.12 Mbps	17.55 Mbps	13.52 Mbps	19 Mbps
	2nd quartile	31.69 Mbps	32.74 Mbps	19.54 Mbps	19.31 Mbps	16.56 Mbps	22.25 Mbps
	3rd quartile	43.91 Mbps	38.31 Mbps	21.32 Mbps	20.73 Mbps	19.58 Mbps	24.08 Mbps
	Loss Rate	0.42%	0.57%	0.56%	0.16%	0.25%	0.33%
Link utilization	CG Traffic	25.65%	30.35%	57.27%	57.32%	48.5%	43.33%
	TCP Traffic	66.68%	63.31%	39.12%	38.57%	32.86%	43.37%
	Total	92.33%	93.66%	96.39%	95.89%	81.36%	86.7%

value is below of the reference flow of commercial platforms, however, it does not prevent them from operating. Regarding queues, we use two classless mechanisms: Stochastic Fairness Queuing (SFQ) for the CG class and Packet First-In, First-Out (pFIFO) for the BE class. As recommended by [25], we take care to limit their respective sizes: the pFIFO to 250 packets, the SFQ to 63 packets.

1) SCReAM vs Cubic: Fig. 8 shows the results of the experiment. We can see that the CG traffic maintains its throughput at 30 Mbps while maintaining negligible delays. According to Table II, we reach an average of 1.16 ms. The delays suffered by TCP flows are much greater (100ms on average), which is however not damaging since the service is not sensitive to latency. The TCP flow bitrate then stabilizes around 20 Mbps. So the fairness is far better. We notice some latency peaks, generally not exceeding 150 ms. This value is consistent with the average throughput observed and the size of the queue. Remember that the BE is limited to 250 packets, or approximately 3 Mb. Given a throughput of 20 Mbps, it only takes 150 ms to fill the queue. Delays observed allow us to conclude that the pFIFO is sometimes full, leading to a few packet losses. We do not see them on the third graph because Cubic reacts quickly to adapt its bitrate. Its average loss rate is low, approximately 0.56%.

2) SCReAM vs BBRv2: We can observe in Fig. 9 that the CG flow maintains its initial flow rate for the entire duration of the experiment. As pointed out before, its queuing times remain very low and losses are almost non-existent. This results in excellent QoE despite the competing traffic. Whether it competes with Cubic or BBRv2, HTB effectively preserves the QoS of the CG traffic. The statistics of the CG traffic are very similar between the two experiments. BBRv2, however, has a better control over delays (95ms on average for Cubic vs 67ms for BBRv2), and presents fewer latency peaks exceeding

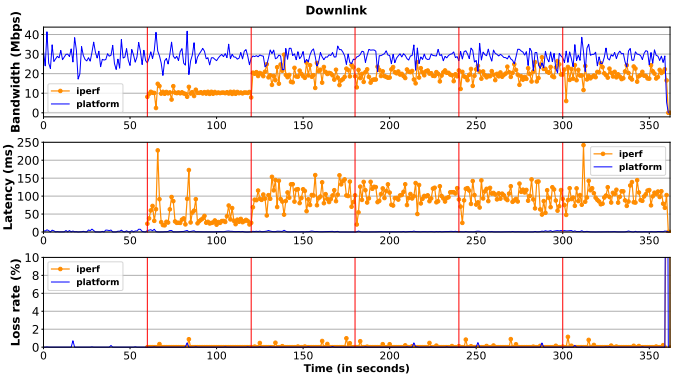

Fig. 8: HTB : SCReAM vs Cubic

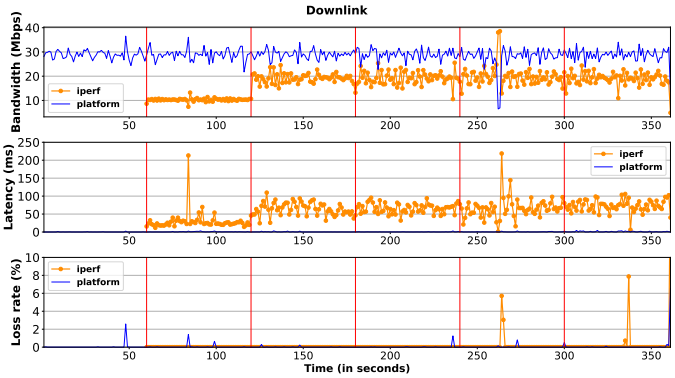

Fig. 9: HTB : SCReAM vs BBRv2

100ms.

When looking attentively to bandwidth curves, the reader may wonder why the throughput of CG can suddenly drop (second 260 in Figure 9). Actually, this is not caused by the network load but by the video encoder. Indeed, the CG traffic being from a real application, some in-game events (typically a fixed loading screen) can reduce the video complexity and thus the CG traffic throughput. We also see that iperf does not take long to exploit it where the curves are crossing each other, but the CG traffic is also able to quickly restore its throughput. This shows a good interaction between the CCAs and HTB.

D. Low Latency, Low Loss, and Scalable Throughput (L4S)

The IETF L4S architecture is proposed for low-latency services. We use the DualPI2 AQM in the network router (being the bottleneck), with one queue for low-latency traffic and one for best effort traffic. We set the maximum size of the L4S queues to 1000 packets (approximately 12 MB). The rest of the configuration is let by default, as suggested by the IETF.

1) SCReAM vs Cubic: Fig. 10 shows that L4S is also able to preserve the QoS of CG traffic against Cubic. The three metrics: average bitrate (24.25 Mbps), average latency (0.53 ms) and loss rate are fully compatible with a good experience. Regarding the TCP traffic, the queuing latency is much reduced compared to HTB (4.68 ms instead of 99.78 ms) but it is at the expense of a slightly lower average bitrate. We can also note that HTB achieved a higher throughput for CG (28.64 Mbps), that can lead to a better video quality. Another point is that link utilization was better with HTB (96.39%) than L4S (81.36%). This is another important criteria for a network provider.

2) SCReAM vs BBRv2: In this experiment, bandwidth sharing between iperf and CG traffic is perfectly balanced. As depicted in Figure 11, both bitrates tend to stabilize at approximately 22 Mbps. Regardless of whether CG traffic is contending with Cubic or BBRv2, its delays typically remain below 1 ms. Conversely, best-effort traffic (*e.g iperf*) may encounter prolonged queuing times but which should not be detrimental to their applications. The DualPI2 AQM incorporates a coupling mechanism between its two queues, proactively transmitting congestion signals to manage bandwidth utilization. The type and the frequency of these signals varies based on the traffic type, distinguishing between L4S and best-effort traffic. A comparison between Figures 10 and 11 illustrates the contrast between Cubic and BBRv2. Cubic exhibits high sensitivity to packet loss. As observed in [26], even a minimal 0.1% loss can diminish its throughput by a factor of 10. Consequently, TCP experiences fewer queuing delays under Cubic CCA but achieves a significantly lower bitrate while BBRv2 successfully maximizes its bitrate while containing its queuing delay budget.

V. Discussion

Our evaluation shows that the use of traffic classes can effectively preserve the QoS of CG traffic. Thanks to HTB, SCReAM does not appear to suffer from concurrent traffic as it does with the simple drop-pail strategy, where both traffic types share a single queue. A single queue can lead to an increase in queuing delays when concurrent traffic does not reduce its bitrate enough, or only to loss events. However, the HTB approach presents some inherent difficulties to implement from an operational point of view. Indeed, the proper configuration of the parameters for each traffic class requires a good knowledge of the flows composing the network traffic at the bottleneck, what can be challenging and difficult to maintain in time. We can also mention the difficulty to identify CG traffic. The machine learning models proposed in [5], [4] must be regularly updated and the misclassification of a CG service can lead to the violation of net neutrality.

The use of L4S solves these issues. It requires less configuration and no traffic analysis, what saves computing resources. Only two traffic categories are considered (classic "best effort" traffic and scalable "low-latency" traffic). The distinction is made in a deterministic manner at the protocol level (according to the setting of the 2 IP-ECN bits), which is much simpler for an operator that has only a few parameters to configure in L4S, and those do not depend on the precise evolution of traffic shares. L4S, however, requires an effort from the application provider to implement a scalable CCA (which is the case of our CG platform, in which we integrated SCReAM), whereas HTB can be used by any current application with

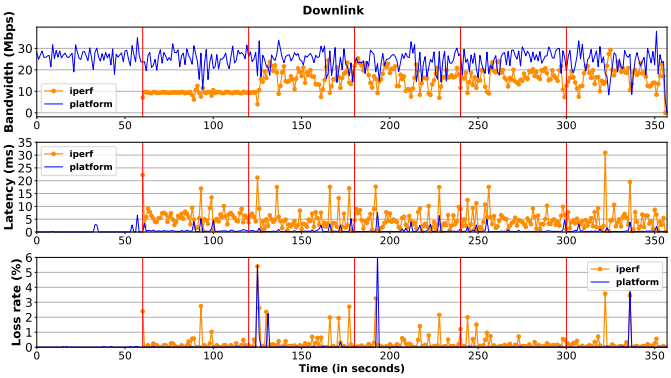

Fig. 10: L4S : SCReAM vs Cubic

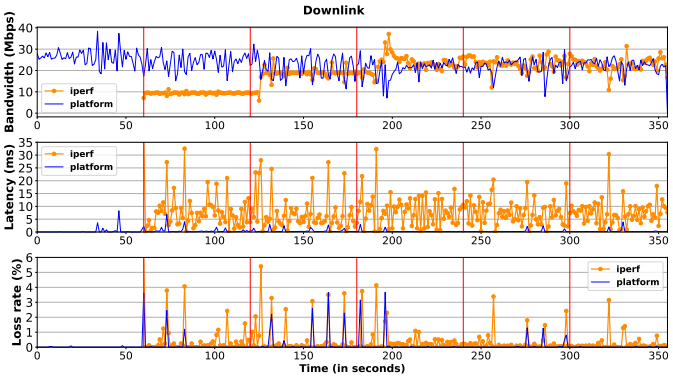

Fig. 11: L4S : SCReAM vs BBRv2

no modification. Because both approaches are able of good performance, the adoption of one or another will probably be driven by the operational constraints and the actors that will be the driving force pushing innovation in networks between networked application providers and network providers.

HTB and L4S not only differ in their implementation requirements, but also in their strategy to manage their queues which results in different degrees of isolation between low-latency traffic and classic traffic. With L4S, there is no specific outgoing bandwidth configuration for each queue since the two share the same outgoing bandwidth. As such, if one traffic has a low bitrate, the other one can take more bandwidth. More precisely, the fairness in L4S is implemented by the coupled AQM algorithm that achieves window fairness between L4S and best effort traffic by balancing the congestion marking probabilities. This balance is a parameter that can be changed, among others, when deploying DualPi2. We considered the default value for all parameters but they were defined after a DCTCP and TCP Cubic scenario and may be suboptimal in our case. This could explain the lower global link utilization reported with L4S and the fact that the level of balance depends on the CCA's reaction to congestion signals (32% of link utilization with Cubic vs 43% with BBRv2). Since DualPI2 doesn't guarantee a minimum bitrate for CG, an unresponsive TCP CCA can also potentially compromise the QoS of CG traffic. The lack of responsiveness to loss-based or ECN congestion signals can result in the starvation of L4S traffic, as evidenced by the behavior of BBRv1. This fairness concern led to the development of BBRv2. In a class-based queuing approach like HTB, each packet is forwarded to the dedicated queue, which has been previously configured with specific parameters: Ceil (maximum outgoing bandwidth capacity), Rate (guaranteed bandwidth), Prio (class priority). This allows to achieve a perfect pre-defined balance between classes, whatever the CCA in use, but some improper configurations, for instance too protective towards a class, can lead to bandwidth waste.

VI. Conclusion

The main objective of this work was to enhance the QoS of CG traffic by avoiding the buffer-bloat phenomenon and the starvation against competing flows. To achieve this, it is crucial to consider both the queuing discipline enforced at the bottleneck and the CCAs used. We considered two queuing disciplines: *HTB* and *DualPI2*. The latter consists of 2 coupled queues that enforce fairness between classic and low latency flows. However, traffic going through the *low latency* queue (*L4S*) must satisfy a few requirements. Since current CG platforms do not comply with L4S requirements, we developed our own experimental CG platform and used SCReAM v2 for congestion control due to its compatibility with L4S. As far as we know, this CCA has never been evaluated with real CG traffic.

Our evaluations show that SCReAM is well suited for CG traffic. Our CG platform even outperformed GeForce Now (GFN) in terms of queuing delays under cellular network conditions. Resource sharing between competing CCAs negatively impacted CG traffic on a simple *drop-tail* queue. On the other hand, both HTB and DualPI2 perfectly preserved the CG traffic QoS from bufferbloat by using dedicated queues.

Finally, we discussed the fact that HTB configuration can be tricky and requires prior traffic classification. Conversely, setting up the L4S queuing discipline is much simpler for a network operator but requires the application support, what is made easier by open-source scalable CCAs like SCReAM.

Our future work will pursue this research topic in several directions. We will evaluate the capacity of HTB and L4S to preserve CG traffic QoS in cellular network conditions. We will replace the competing `TCP iperf` traffic by more realistic competing traffic types such as *DASH* video streaming and consider *scalable* competing traffic within the L4S queue to further challenge the L4S architecture. We will also work on designing an open-source CCA for CG by optimizing the configuration of SCREAM and evaluating it against other platforms.

Acknowledgment

This work is partially funded by the French ANR MOSAICO project, No ANR-19-CE25-0012.

REFERENCES

[1] J. Gettys and K. Nichols, "Bufferbloat: Dark buffers in the internet: Networks without effective aqm may again be vulnerable to congestion collapse." *Queue*, vol. 9, no. 11, p. 40–54, 11 2011. [Online]. Available: https://doi.org/10.1145/2063166.2071893

[2] X. Marchal, P. Graff, J. R. Ky, T. Cholez, S. Tuffin, B. Mathieu, and O. Festor, "An analysis of cloud gaming platforms behaviour under synthetic network constraints and real cellular networks conditions," *J. Netw. Syst. Manag.*, vol. 31, no. 2, p. 39, 2023. [Online]. Available: https://doi.org/10.1007/s10922-023-09720-9

[3] M. Devera, "Hierachical token bucket theory." http://luxik.cdi.cz/~devik/qos/htb/manual/theory.htm, 1999, viewed on Septembre 12, 2023.

[4] P. Graff, X. Marchal, T. Cholez, B. Mathieu, and O. Festor, "Efficient identification of cloud gaming traffic at the edge," in *NOMS 2023, IEEE/IFIP Network Operations and Management Symposium, Miami, FL, USA, May 8-12, 2023*. IEEE, 2023, pp. 1–10. [Online]. Available: https://doi.org/10.1109/NOMS56928.2023.10154417

[5] J. R. Ky, P. Graff, B. Mathieu, and T. Cholez, "A hybrid P4/NFV architecture for cloud gaming traffic detection with unsupervised ML," in *IEEE Symposium on Computers and Communications, ISCC 2023, Gammarth, Tunisia, July 9-12, 2023*. IEEE, 2023, pp. 733–738. [Online]. Available: https://doi.org/10.1109/ISCC58397.2023.10217863

[6] O. Albisser, K. De Schepper, B. Briscoe, O. Tilmans, and H. Steen, "DUALPI2 - Low Latency, Low Loss and Scalable (L4S) AQM," in *Proc. Netdev 0x13*, Mar. 2019. [Online]. Available: https://www.files.netdevconf.org/f/febbe8c6a05b4ceab641/?dl=1

[7] I. Johansson, "Self-Clocked Rate Adaptation for Conversational Video in LTE," in *Proceedings of the 2014 ACM SIGCOMM Workshop on Capacity Sharing Workshop*, ser. CSWS '14. New York, NY, USA: Association for Computing Machinery, 2014, pp. 51–56, event-place: Chicago, Illinois, USA. [Online]. Available: https://doi.org/10.1145/2630088.2631976

[8] D. Brunello, I. Johansson S, M. Ozger, and C. Cavdar, "Low latency low loss scalable throughput in 5g networks," in *2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)*, 2021, pp. 1–7.

[9] M. Jarschel, D. Schlosser, S. Scheuring, and T. Hoßfeld, "An evaluation of QoE in cloud gaming based on subjective tests," in *2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pp. 330–335.

[10] 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.

[11] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg, "PIE: A lightweight control scheme to address the bufferbloat problem," in *2013 IEEE 14th International Conference on High Performance Switching and Routing (HPSR)*, 2013, pp. 148–155.

[12] K. Nichols and V. Jacobson, "Controlling Queue Delay: A Modern AQM is Just One Piece of the Solution to Bufferbloat." *Queue*, vol. 10, no. 5, pp. 20–34, May 2012, place: New York, NY, USA Publisher: Association for Computing Machinery. [Online]. Available: https://doi.org/10.1145/2208917.2209336

[13] 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

[14] M. Alizadeh, A. G. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *Proceedings of the ACM SIGCOMM 2010 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, New Delhi, India, August 30 -September 3, 2010*, S. Kalyanaraman, V. N. Padmanabhan, K. K. Ramakrishnan, R. Shorey, and G. M. Voelker, Eds. ACM, 2010, pp. 63–74. [Online]. Available: https://doi.org/10.1145/1851182.1851192

[15] O. Bondarenko, K. De Schepper, I.-J. Tsang, B. Briscoe, A. Petlund, and C. Griwodz, "Ultra-low delay for all: Live experience, live analysis," in *Proceedings of the 7th International Conference on Multimedia Systems*, ser. MMSys '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2910017.2910633

[16] K. D. Schepper and B. Briscoe, "The Explicit Congestion Notification (ECN) Protocol for Low Latency, Low Loss, and Scalable Throughput (L4S)," RFC 9331, Jan. 2023. [Online]. Available: https://www.rfc-editor.org/info/rfc9331

[17] B. Briscoe, K. D. Schepper, O. Albisser, O. Tilmans, N. Kuhn, G. Fairhurst, R. Scheffenegger, M. Abrahamsson, I. Johansson, P. Balasubramanian, D. Pullen, G. Bracha, J. Morton, and D. Täht, "Implementing the 'prague requirements' for low latency low loss scalable throughput (l4s)," 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:180259180

[18] C. Smith, D. Chu, D. Ray, S. Seshan, and T. Wei, "SQP: Congestion control for low-latency interactive video streaming," in *arXiv*, 2022.

[19] A. Alós, F. Morán, P. Carballeira, D. Berjón, and N. García, "Congestion control for cloud gaming over udp based on round-trip video latency," *IEEE Access*, vol. 7, pp. 78 882–78 897, 2019.

[20] X. Xu and M. Claypool, "Measurement of cloud-based game streaming system response to competing tcp cubic or tcp bbr flows," in *Proceedings of the 22nd ACM Internet Measurement Conference*, ser. IMC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 305–316. [Online]. Available: https://doi.org/10.1145/3517745.3561464

[21] C.-Y. Huang, K.-T. Chen, D.-Y. Chen, H.-J. Hsu, and C.-H. Hsu, "Gaminganywhere: The first open source cloud gaming system," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 10, no. 1s, 1 2014. [Online]. Available: https://doi.org/10.1145/2537855

[22] "L'etat d'internet en france," 7 2023. [Online]. Available: {https://www.arcep.fr/uploads/tx_gspublication/RA-2023-TOME3_etat-internet-france_juillet2023.pdf}

[23] M. Jarschel, D. Schlosser, S. Scheuring, and T. Hoßfeld, "Gaming in the clouds: Qoe and the users' perspective," *Mathematical and Computer Modelling*, vol. 57, no. 11, pp. 2883–2894, 2013, information System Security and Performance Modeling and Simulation for Future Mobile Networks. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0895717711007771

[24] B. Turkovic, F. A. Kuipers, and S. Uhlig, "Interactions between congestion control algorithms," in *2019 Network Traffic Measurement and Analysis Conference (TMA)*, 2019, pp. 161–168.

[25] E. C. Janczukowicz, "Qos management for webrtc : loose coupling strategies," Ph.D. dissertation, 2017, thèse de doctorat dirigée par Bonnin, Jean-Marie Informatique Ecole nationale supérieure Mines-Télécom Atlantique Bretagne Pays de la Loire 2017. [Online]. Available: http://www.theses.fr/2017IMTA0010

[26] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: congestion-based congestion control," *Commun. ACM*, vol. 60, no. 2, p. 58–66, 1 2017. [Online]. Available: https://doi.org/10.1145/3009824