IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 9, 2019, accepted June 4, 2019, date of publication June 17, 2019, date of current version June 28, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2923294

Congestion Control for Cloud Gaming Over UDP Based on Round-Trip Video Latency

ALBERTO ALÓS[1], FRANCISCO MORÁN[1], PABLO CARBALLEIRA[2], DANIEL BERJÓN[1], AND NARCISO GARCÍA[1]

[1]Information Processing and Telecommunications Center, Grupo de Tratamiento de Imágenes (GTI), ETSI Telecomunicación, Universidad Politécnica de Madrid, 28040 Madrid, Spain

[2]Video Processing and Understanding Lab, Escuela Politécnica Superior, Universidad Autónoma de Madrid, 28049 Madrid, Spain

Corresponding author: Alberto Alós (aas@gti.ssr.upm.es)

This work was supported in part by the Ministerio de Ciencia, Innovación y Universidades (AEI/FEDER) of the Spanish Government through the Project "Open Graphics Gaming Cloud" under Grant RTC-2016-5676-7 and the Project "Immersive Visual Media Environments" under Grant TEC2016-75981.

ABSTRACT We describe a network congestion control mechanism for cloud gaming (CG) platforms based on the user datagram protocol (UDP). To minimize the contribution of the downstream transmission delay to the total end-to-end latency in the interaction–perception loop, we first define the round-trip video latency (RTVL) and develop a congestion model. Based on them, we design and implement an adaptation strategy that detects the early stages of congestion to prevent high values of RTVL and network bufferbloat, thus avoiding packet losses. Using data measured from the network, our strategy modifies the target output bitrate of the video encoder to throttle down or upto the data flow sent by the server to the client. In the presence of sudden downstream channel capacity drops of over 40%, our algorithm reactively manages to satisfy the key CG requirements for interactive games by entirely avoiding the packet losses and keeping the RTVL below 100 ms. In reasonably stable network conditions, our algorithm proactively keeps exploring for higher bitrates and building a "network state dictionary," due to which it achieves an effective downstream channel capacity use of ∼95%.

INDEX TERMS Cloud gaming, congestion control, adaptive video coding, QoS, BBR.

I. INTRODUCTION

The improvement of network infrastructures in recent years, mainly due to the proliferation of FTTX (Fiber To The X) deployments which provide high-speed and symmetric bandwidth connections, has gone hand in hand with the growth of cloud services. Among the variety of services flowing through Internet, some of them are more demanding than others in terms of the QoS (Quality of Service) parameters that can be measured in the network. Even more important than the objective QoS is the QoE (Quality of Experience) perceived by the user, especially for interactive audio-visual services, whose QoE is very dependent on the total latency, and on the quality of the video displayed to the user.

CG (Cloud Gaming), a.k.a. GaaS (Games as a Service), is one of such services, which has been possible since powerful GPUs (Graphics Processing Units) and FTTX enabled the (almost) real-time execution of the "game play and rendering, plus video coding, transmission, decoding and display" chain. But it is among the interactive services with the strictest requirements on transmission delays and packet losses. Jarschel *et al.* [1], [2] carried out a study of the factors that degrade the QoE for CG, which sets some boundaries in objective terms of QoS, based on subjective tests. They concluded that downstream packet loss is the most important parameter, followed by downstream delay and jitter. This makes it very challenging to design a user-friendly CG platform.

Indeed, guaranteeing no losses and a low latency throughout a game session is not always possible: there are relatively static limitations, such as the network technologies/capabilities by region [3], but also intrinsically dynamic ones, such as network congestion. Some Internet services are delivered to the client over managed network infrastructures, but CG must be understood as an OTT (Over-The-Top) service competing with other OTT services over unmanaged networks. In the OTT framework, there exist some solutions called "DiffServ vs. InServ" (Differentiated vs.

The associate editor coordinating the review of this manuscript and approving it for publication was Nizar Zorba.

Integrated Services), but their deployment depends on the underlying architecture of the Internet service provider, so they are not a short-term solution for us.

Our research has focused on designing an algorithm for C^{3}G (Congestion Control for Cloud Gaming) to meet the strict CG requirements of having minimal losses and keeping latency within playability limits. To do so, we first worked on understanding the network congestion process to find in latency good predictors of network ''bufferbloat'' [4] (which ultimately leads to losses), and this prompted us to analyze the measure of both losses and latency.

Video packet losses yield different decoding errors depending on the type of packets which are lost, e.g., the video compression picture type (I, P or B) they belong to. In a non-interactive video streaming scenario, these different errors have a varying impact on the perceived quality of the decoded video, but in a CG scenario they may also disturb in completely different ways the user interaction, i.e., the game play itself. However, defining and measuring losses is trivial.

On the other hand, defining interactive latency is more complex, because many delays contribute to it, so several studies related to our work have started from different definitions of the end-to-end delay. We highlight Wen and Hsiao's RTRD (Round-Trip Reaction Delay) [5], which includes all delays in the interaction-perception loop. Other researchers related (parts of) RTRD to playability [6]–[10] and concluded that latency thresholds depend on the game type, but one can generally assume that keeping the RTRD below 100 ms guarantees a good playability, because it does even for the ''fastest'' games, such as racing ones or first person shooters.

In the general context of video streaming, and regarding packet loss reduction, a long-term buffer could be included at the client side to allow for more retransmissions, and thus ensure the arrival of complete frames before they must be decoded. But the price is adding an arbitrary delay at the decoder, thus increasing latency. Several widely used video streaming techniques help reduce latency and mitigate the image quality degradation due to packet losses. For instance, in scenarios like ours where low latency is critical, it is good encoding practice to avoid using B frames altogether, or set the VBV (Video Buffering Verifier) size to the target frame size. These common techniques to reduce latency logically increase the downstream bitrate, or limit the adaptation capability of the platform to the channel capacity.

As for the more specific context of CG [11], congestion control is still a relatively unexplored field. In Section II, we summarize our findings on channel-adaptive algorithms developed to meet the strict QoS requirements mentioned above, both in terms of latency and losses, and at the same time try and maximize the effective use of the downstream channel capacity.

Our contributions start by modeling the behavior of the network at different stages of the congestion process. As we explain in Section III, we immediately found it essential to determine the contribution of the downstream delay to the RTRD. We therefore defined the RTVL (Round-Trip Video Latency) before designing our congestion model, which is based on that of BBR (Bottleneck Bandwidth and Round-trip propagation time) [12], a congestion control algorithm recently developed by Google. Our model allows us to establish a relationship between the maximum channel capacity and the buffer size of the potential network bottlenecks, the amount of data sent to the network (by setting a video encoder target bitrate), and of course the RTVL itself.

As a result of our congestion model, we also designed a generic adaptation strategy explained in Section IV, and implemented an algorithm for C^{3}G based on UDP at the application layer. This implementation was adapted according to the limitations of the commercial CG platform we used for testing, as described in Section V.

Section VI reports on the tests we carried out with our algorithm, in particular to compare it with BBR. The experimental results show how our algorithm reacts quickly to channel capacity drops by reducing the target video encoding bitrate, and manages to completely avoid losses for sudden drops of up to ∼42%. The results show as well that our algorithm keeps the RTVL below 100 ms for steady bandwidth limitations, while continuously exploring for higher target bitrates. This proactive exploration strategy results in an effective use of 93–96% of the available channel capacity.

II. STATE OF THE ART

In this Section, we review existing congestion control algorithms designed to work at either the transport or the application layer, and based on TCP (Transmission Control Protocol), UDP (User Datagram Protocol), or none of them.

A. TRANSPORT LAYER POTENTIAL SOLUTIONS

Theoretically, flow and congestion control is a service that belongs in the transport layer; indeed, the ubiquitous TCP provides applications with transparent flow control and ordered reliable delivery, and has been a cornerstone of the Internet over the past few decades because it excels at delivering non-latency-sensitive loads (up to now the bulk of the Internet traffic).

On the other side of the spectrum, UDP is the trivial transport protocol, which provides no service beyond multiplexing, thereby leaving all flow control, error checking, and message ordering to the applications themselves. In between these two extremes, there are other protocols that provide subsets of the functionality provided by TCP, such as SCTP (Stream Control Transmission Protocol) [13] and DCCP (Datagram Congestion Control Protocol) [14]. However, they see very little use because most commercial routers or firewalls do not support them, leaving application developers with TCP and UDP as the only practical choices.

It is worth mentioning that the design of TCP is generic enough to enable interoperation between implementations using different congestion algorithms, but no matter the specific algorithm the latency is potentially unbounded on account of TCP's reliable nature; therefore, TCP is generally not appropriate for real-time, low-latency traffic. Still, some

of the congestion control algorithms that have been proposed for TCP provided us with a good starting point. Most of them use packet losses as congestion signal (Tahoe, [New] Reno, [CU]BIC, etc.) and are therefore not acceptable for CG; some others, such as FAST TCP [15] and the already mentioned, more recent BBR [12], [16], use delay measurements to detect congestion before it actually happens. We will cover in more depth different aspects of the latter, which is the most similar algorithm to ours, in Subsections III-A and IV-B.

B. APPLICATION LAYER POTENTIAL SOLUTIONS

Given the problems described above to implement real-time streaming at the transport layer, multiple application-layer solutions have been designed.

VoD (Video-on-Demand) is a service superficially similar to CG in the sense that it needs to react to network conditions in real time, and adapt the video quality accordingly. Popular techniques like ABR (Adaptive BitRate) [17] consist in that the (typically HTTP-based) server offers the same content in a variety of coding presets that the client autonomously switches between, depending on the evolution of its own reception buffer. However, these techniques are only possible because the content is not user-dependent and users can easily tolerate delays in the order of seconds, allowing time for the server to pre-generate all preset qualities.

Video conference, on the other hand, is a service much closer to CG: its content is session-specific and produced in real time, and it has stricter delay requirements [18]. Still, video conference can tolerate significantly higher delays than CG, and also drastic video bitrate reductions because the essential source of information is (low-bandwidth) audio, while the video information is essential to CG. Hence, most proposals in the literature relevant to CG have been purpose-designed.

Jarvinen *et al.* [19] used TCP's RTT jitter to detect network congestion, and then decide upon the triggering of the bitrate adaptation. In their solution, a video adaptation module constantly monitors the network status, and dynamically adjusts the encoder target bitrate using an AIMD (Additive-Increase/Multiplicative-Decrease) scheme, just like TCP. They consider RTT jitter to be a binary congestion signal, which causes too much oscillation in the target bitrate when there is a persistent bandwidth limitation. Furthermore, their proposal is TCP-based and does not offer a mechanism to cope with losses and subsequent retransmissions.

Wang and Dey [20] proposed a procedure that requests one of several preset- and game-specific bitrates depending on downlink delay thresholding, as well as a method, based on the same metric, for reducing the play-out buffer delay. Another proposal of theirs [21], also based on delay thresholding, dynamically modifies game rendering parameters to modulate video complexity. Both proposals use a previous network probing mechanism [22] to measure delays and losses, and share the same shortcomings: they require a deep analysis of the characteristics of each game, and their use of discrete bitrate presets does not provide sufficient adaptation granularity.

More recently, Hong *et al.* [23] proposed a MOS (Mean Opinion Score) based model for dynamic frame rate and bitrate adaptation on the open-source CG platform GamingAnywhere [24]. The model implicitly considers the available bandwidth, which estimation is inspired in WBest [25]; it keeps track of the dispersion time of the video packets but does not consider a threshold over the latency.

Finally, other proposals for CG [24] simply leverage RT[C]P (Real-time Transport [Control] Protocol) over UDP, a protocol designed for audio and video streaming that provides QoS feedback, temporal reconstruction and loss detection. However, they do not implement bandwidth management, guarantee a given QoS, or provide any means to address congestion control.

III. CONGESTION MODEL

From the different approaches reviewed in Section II, we based our work on BBR, whose network congestion model and adaptation strategy are explained in more detail in Subsection III-A. Subsection III-B introduces the congestion model of our C^{3}G algorithm. Before explaining in Section IV our control strategy to adapt the server output bitrate to a varying channel capacity, we describe our C^{3}G discrete-time congestion detection algorithm in Subsection III-C.

A. BBR's CONGESTION MODEL AND ADAPTATION STRATEGY

BBR's network congestion model is based on the one defined by Kleinrock [26], which considers that an arbitrary complex path formed by many links behaves as a single one, whose bandwidth is the minimum of those of all individual links. Kleinrock defined the OOP (Optimal Operating Point) of a global path or individual link as the transmission bitrate allowing to use that path/link at its maximum channel capacity while keeping a minimum transmission delay. If the sender transmits a bitrate lower than OOP, the channel is underused; but if it tries to transmit a higher one, the delay increases while the effective delivered bitrate does not.

All these bitrates, delays, and therefore OOPs may of course vary along time in a real network, and BBR relies on TCP to define and estimate two time-dependent variables: $BtlBW$, the bottleneck bandwidth, and $RTprop$, the round-trip propagation time. The latter is the minimum of all RTTs reported by TCP over some time window (typically tens of seconds to minutes), and the former is the maximum of the delivery rates (ratios of delivered data to elapsed time) calculated over some other time window (typically 6-10 RTT). Based on these two variables, BBR defines as well the BDP (Bandwidth Delay Product), which is simply $BDP = BtlBW \cdot RTprop$.

The key premise of BBR's congestion model is that the OOP is found when $BDP = inflight$, a native TCP parameter representing the number of unacknowledged sent packets.

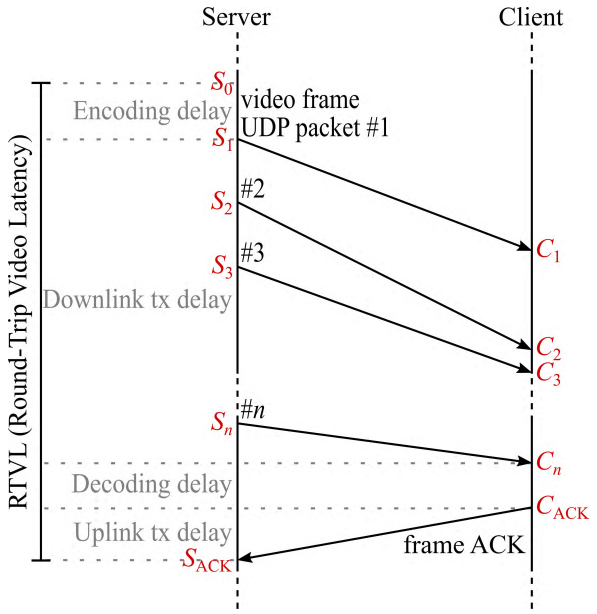

FIGURE 1. We measure *RTVL*(*t*) at the server, for each video frame, as the time elapsed since the raw frame output by the game engine was sent to the video encoder (S_0) until its decoding ACK is received from the client (S_{ACK}).

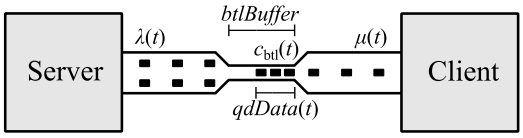

FIGURE 2. Bottleneck of maximum capacity $c_{btl}(t)$ and buffer size *btlBuffer*. Soon after $\lambda(t) > c_{btl}(t)$, *qdData*$(t) > 0$.

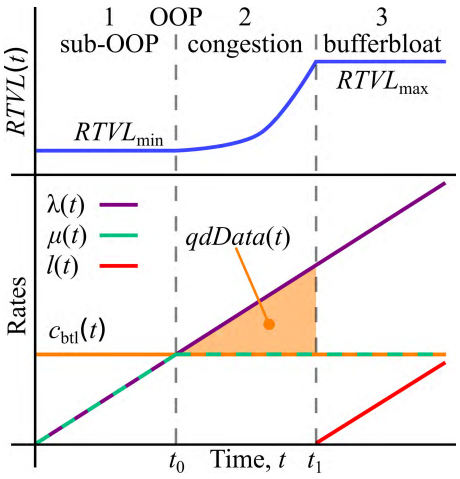

FIGURE 3. Three phases in our congestion model: 1) no congestion and minimal RTVL, but channel underused; 2) congestion beyond OOP (t_0), but reaction still possible until buffer saturation (t_1); 3) bufferbloat, to be avoided.

If *inflight* $>$ *BDP*, a packet queue starts to grow at some bottleneck link, and *RTT* increases linearly with *inflight*.

BBR's adaptation strategy consists in continuously estimating *BtlBW* and *RTprop*, hence *BDP*, and pacing the packet transmission to have *inflight* match or remain just below *BDP*. It also aims higher periodically by tentatively increasing the transmission bitrate, and then immediately decreasing it, according to what is known as a MIMD (Multiplicative Increase / Multiplicative Decrease) scheme based on a cycle of fixed gains: see Subsection IV-B.

B. C^{3}G's CONGESTION MODEL

BBR's congestion model cannot be implemented over UDP since it requires the native TCP parameters mentioned in the previous Subsection. Besides, in our application layer context of video transmission and CG in particular, it makes more sense to have the client send ACKs per video frame, instead of per UDP packet (which anyway would completely defeat the purpose of using UDP vs. TCP).

This led us to define the RTVL (Round-Trip Video Latency) illustrated by Figure 1. C^{3}G's congestion model assumes that the client sends an explicit ACK after having received all the UDP packets of a video frame and decoded it. Upon reception of that ACK at instant S_{ACK}, the server computes *RTVL*(t) for that particular frame as the time elapsed since its raw version output by the game engine was sent to the video encoder at instant S_0.

Our application-oriented, per-frame *RTVL*(t) is meant to be analogous to BBR's (in fact, TCP's) *RTT*. To continue with this analogy, and be able to seek an OOP defined similarly to BBR's, we need a way to estimate an equivalent to TCP's *inflight*, which is inherently impossible in UDP. This is why we measure the server-sent and client-received bitrates, $\lambda(t)$ and $\mu(t)$, which are the accumulated sizes of the sent and received UDP packets during a certain time window. Note that *RTVL*(t) includes, as explicitly shown in Figure 1, the video frame encoding and decoding delays, which we will assume constant for the moment, although they do depend (weakly) on $\lambda(t)$ and $\mu(t)$.

C^{3}G's ideal congestion model also assumes that the time windows of both server and client are properly aligned, so when there is no congestion $\mu(t) = \lambda(t)$. However, as shown in Figure 2, a link in the network may turn into a bottleneck if its maximum capacity $c_{btl}(t)$, which is equivalent to BBR's *BtlBW*, is lower than $\lambda(t)$: in such a case, its buffer of size *btlBuffer* will start holding an amount of queued data *qdData*$(t) > 0$ waiting to be delivered, and $\mu(t) < \lambda(t)$.

Figure 3 illustrates how C^{3}G's congestion model works by showing how $\mu(t)$, *qdData*(t), *RTVL*(t), and packet losses $l(t)$ would evolve over time in a rather academic example scenario where $\lambda(t)$ would increase linearly with time, while $c_{btl}(t)$ would remain constant. Three phases can be identified:

1. $t < t_0$: $\lambda(t) < c_{btl}(t) \Rightarrow \mu(t) = \lambda(t)$, so there is no congestion. Besides, $RTVL(t) = RTVL_{min}$, so the latency is minimal, but this phase is nevertheless sub-optimal because the channel is underused.
2. $t_0 \leq t \leq t_1$: The OOP is reached at t_0, when and because $\lambda(t_0) = c_{btl}(t)$, but then, as $\lambda(t)$ keeps growing, UDP packets start to accumulate in the buffer because $\mu(t)$ remains equal to $c_{btl}(t)$. Both *qdData*(t) and *RTVL*(t)

increase:

$$qdData(t) = \int_{t_0}^{t} [\lambda(t) - \mu(t)]\, dt,$$

$$RTVL(t) = RTVL_{\min} + \frac{qdData(t)}{c_{btl}(t)}.$$

During this congestion phase, it is still possible to react before incurring losses, because the buffer is not full until $t = t_1$, when RTVL reaches its maximum value $RTVL_{\max}$, which is imposed by the buffer size.

3. $t > t_1$: Once the buffer is full, if $\lambda(t)$ keeps increasing $RTVL(t)$ remains equal to $RTVL_{\max}$ but packets are dropped and $l(t) > 0$. This phase must be avoided.

C. C³G's DISCRETE-TIME CONGESTION DETECTION ALGORITHM

Figure 3 is obviously an over-simplified version of what can be observed and done in a real system. Real congestion detection and control algorithms, like ours, operate in discrete time and typically in an iterative way, because they are invoked periodically. In such real systems, it might be hard to align the server and client time windows used to measure $\lambda(t)$ and $\mu(t)$, hence to accurately estimate $qdData(t)$. Besides, these time windows, even if properly aligned, might be of a different size from the one used to measure $RTVL(t)$ at the server.

This is why our C³G algorithm, indeed invoked with period Δt, works with average rates between two successive executions at instants t_{prev} and $t_{now} = t_{prev} + \Delta t$: $\bar{\lambda}(t_{now})$, $\bar{\mu}(t_{now})$ and $\overline{RTVL}(t_{now})$ are the respective averages of all $\lambda(t)$, $\mu(t)$ and $RTVL(t)$ samples recorded for $t \in [t_{prev}, t_{now}]$. As for $qdData(t_{now})$, it is calculated by adding all differences $\lambda(t) - \mu(t)$ since the execution instant t_{cong} at which congestion was detected for the first time.

We will elaborate on t_{cong} in Subsection IV-A, and in Section V on other implementation details such as the misalignment of the $\lambda(t)$ and $\mu(t)$ time windows, but we precisely state already here how our C³G algorithm should ideally calculate its Boolean variable $congestion(t_{now})$, to be able to better explain its adaptation strategy:

$$congestion(t_{now}) = qdData(t_{now}) > 0 \;\&\overline{RTVL}(t_{now}) > \overline{RTVL}(t_{prev}). \quad (1)$$

Note that both conditions must be met for $congestion(t_{now})$ to become true: the mere fact that $qdData(t_{now}) > 0$ is not too worrying in itself, because *btlBuffer* might suffice to mitigate the variations of $\lambda(t)$ and $c_{btl}(t)$ — this is precisely why the buffer is there! In fact, when we detect that $qdData(t_{now}) > 0$, we take advantage of it to estimate $c_{btl}(t_{now}) = \bar{\mu}(t_{now})$. But if we see that, on top of having queued data, $\overline{RTVL}(t)$ has increased, then we do declare $congestion(t_{now})$ true.

IV. ADAPTATION STRATEGY

As hinted above, our C³G algorithm estimates the channel capacity and tries to reach the OOP by adapting $\lambda(t)$ accordingly. Slivar *et al.* [27] carried out tests to derive video encoding adaptation strategies for CG, and concluded that the game type must be taken into account when evaluating QoE, but for both games analyzed in their paper, fluidity (framerate) had a more significant impact on the QoE than video quality (bitrate). The framerate being a very sensitive parameter, we decided not to act on it to modify $\lambda(t)$, but only on the target bitrate for the video encoder, $v(t)$. In Subsection V-C we elaborate on the difficulties involved in commanding $\lambda(t)$ through $v(t)$ but, for the purposes of this Section, we will assume that our control algorithm is indeed able to set $\lambda(t)$ directly.

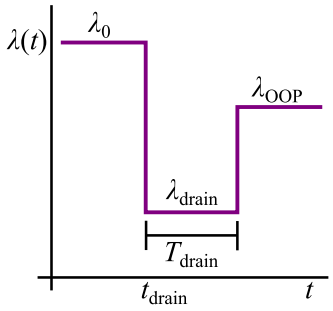

FIGURE 4. Reactive adaptation sequence for $\lambda(t)$.

There are two distinct situations for $\lambda(t)$ adaptation, respectively analyzed in the two Subsections below:

A. When our algorithm detects a channel capacity drop, it reactively decreases the target $\lambda(t)$, which is challenging: if it is too low, $qdData(t) = 0$ will indeed be reached soon, thus exiting the congestion phase 2 of Figure 3, but video quality, hence QoE, will be poor while the buffer is drained; on the other hand, if $\lambda(t)$ is set too high, i.e., too close to its pre-congestion value, video quality might remain acceptable, but at the risk of reaching the bufferbloat phase 3.

B. At all times during a period of apparently stable channel capacity, our algorithm proactively explores higher acceptable values of $\lambda(t)$ to escape the sub-OOP phase 1. Again, this must be done carefully to avoid stepping too deep into the congestion phase 2 and increasing too much the latency or, even worse, ending up in the bufferbloat phase 3.

A. REACTIVE BITRATE DECREASE

Once we detect congestion (see Equation 1), we know we might have to launch a $\lambda(t)$ adaptation sequence like the one shown in Figure 4, which starts by lowering it to λ_{drain} to drain the buffer and reach $qdData(t) = 0$ again as soon as possible.

However, to avoid over-reacting, we do not necessarily lower $\lambda(t)$ whenever we detect $congestion(t_{now})$. Instead, we tentatively set $t_{cong} = t_{now}$ (to mark the starting point for the $qdData(t)$ sum, as explained in Subsection III-C), and then try and predict in two ways whether $RTVL(t)$ will exceed the maximum acceptable threshold $RTVL_{th}$ (e.g., 100 ms) during the next control loop cycle, i.e., for some $t \in [t_{now},\ t_{next} = t_{now} + \Delta t]$. We calculate the following two predicted values:

1. $\widetilde{RTVL}_1(t_{\text{next}})$ is linearly extrapolated from all $RTVL(t)$ samples recorded during the last control cycle, $[t_{\text{prev}}, t_{\text{now}}]$;
2. $\widetilde{RTVL}_2(t_{\text{next}})$ assumes that $qdData(t_{\text{now}})$ is still far below *btlBuffer* (i.e., that t_{now} is still close to t_0 in Figure 3), and that it will increase the last known value $RTVL(t_{\text{last}})$ at a rate given by $\bar{\mu}(t_{\text{now}})$, which we found to be a reasonable estimate of the bottleneck capacity $c_{\text{btl}}(t_{\text{now}})$, as explained at the very end of Subsection III-C:

$$\widetilde{RTVL}_2(t_{\text{next}}) = RTVL(t_{\text{last}}) + \frac{qdData(t_{\text{now}})}{\bar{\mu}(t_{\text{now}})}.$$

If any of these two predicted values is larger than $RTVL_{\text{th}}$, we declare $drainNeeded(t_{\text{now}})$ true, set $t_{\text{drain}} = t_{\text{now}}$, and trigger the adaptation sequence, as explained below.

But if none is, we keep calm and carry on... This means that $congestion(t)$ might be true at several successive runs of the C^{3}G algorithm, and then turn false without $drainNeeded(t)$ ever becoming true. Another couple of good things that might happen "naturally", after one or more successive runs in which $congestion(t)$ is still true, is that $\bar{\lambda}(t_{\text{now}}) < \bar{\mu}(t_{\text{now}})$ ($qdData(t_{\text{now}})$ is still positive, which is the first condition for $congestion(t_{\text{now}})$ to be true, but smaller than $qdData(t_{\text{prev}})$, so the buffer is draining) and that the $RTVL(t)$ samples recorded during the last control cycle show a decreasing tendency (although $\overline{RTVL}(t_{\text{now}}) > \overline{RTVL}(t_{\text{prev}})$, which is the second condition for $congestion(t_{\text{now}})$ to be true). If those two things do happen, we reset $t_{\text{cong}} = t_{\text{now}}$, again tentatively.

Once $drainNeeded(t_{\text{now}})$ is true, so a server-sent rate adaptation sequence like the one of Figure 4 must be launched, we start by calculating λ_{drain} to set it as the new target for $\lambda(t)$. This value must of course be lower than the estimated bottleneck capacity to help drain the buffer and reach $qdData(t) = 0$ again as soon as possible. In principle, it could be desirable to completely drain the buffer in a single control loop cycle, which would be achieved with:

$$\lambda_{\text{drain}} = \bar{\mu}(t_{\text{drain}}) - \frac{qdData(t_{\text{drain}})}{\Delta t}. \tag{2}$$

But this could imply setting $v(t)$ below the minimum acceptable target video encoding bitrate, v_{min}. This is why we impose a drain period T_{drain}, possibly much larger than Δt, during which the target server-sent rate is kept equal to λ_{drain}. We calculate it as follows:

$$T_{\text{drain}} = \frac{qdData(t_{\text{drain}})}{\bar{\mu}(t_{\text{drain}}) - v_{\text{min}}}.$$

Finally, as suggested as well by Figure 4, once T_{drain} is over and the buffer is completely empty again, our C^{3}G algorithm increases the target server-sent rate to achieve the OOP, and sets it to $\lambda_{\text{OOP}} = \bar{\mu}(t_{\text{drain}})$, which is the channel capacity estimated just before entering the drain period.

B. PROACTIVE BITRATE INCREASE (EXPLORATION)

In times of apparently stable channel capacity, any OOP-seeking $\lambda(t)$ adaptation algorithm must explore higher acceptable rates, but must try to be very cautious in doing so, to avoid exceeding $c_{\text{btl}}(t)$ so much that $RTVL(t) > RTVL_{\text{th}}$, or that $l(t) > 0$ due to bufferbloat. This is especially true in the CG context, where a radical increase of $\lambda(t)$, when it is already close to $c_{\text{btl}}(t)$, may cause a fast and vast data accumulation in the bottleneck buffer, hence any of the two undesired consequences above, which lead both to an unacceptable QoE.

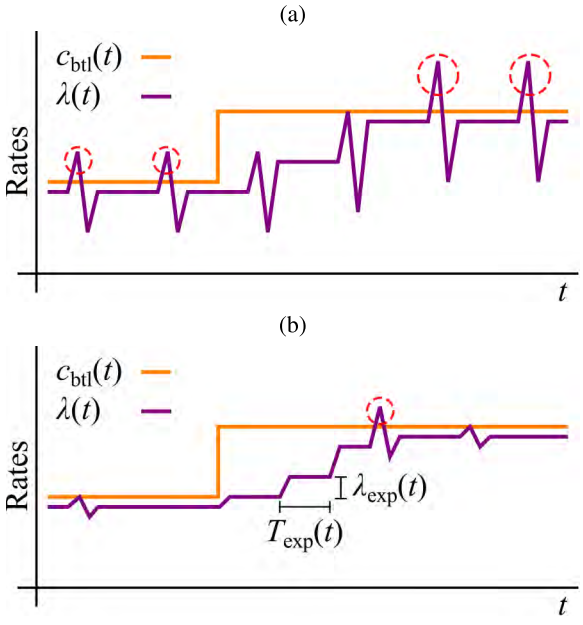

FIGURE 5. Behavior of BBR vs. C^{3}G during their typical target server-sent bitrate exploration sequences. (a) BBR. (b) C^{3}G.

As summarized in Subsection III-A, in terms of exploration, BBR uses a MIMD scheme based in a periodic cycle of gains applied to the server-sent rate to try and reach the OOP ($BDP = inflight$) of a channel of potentially increasing capacity. Figure 5a shows (from left to right) how:

1. The first two attempts to apply 1.25 factors (i.e., 25% gains) to the server-sent rate are not successful, because $c_{\text{btl}}(t)$ is exceeded (which is detected because $inflight > BDP$), so the target rate remains the same for the following cycles.
2. On the contrary, the third and fourth attempts are successful, because the channel capacity $c_{\text{btl}}(t)$ has indeed increased, and the corresponding 1.25 factors for the target rate are both consolidated.
3. The last two attempts to increase the target rate are unsuccessful, like the first two.

BBR's exploration strategy is inadequate for our CG over UDP context for two main reasons, the most obvious being that the x1.25 (i.e., +25%) gain is too greedy, which often impacts badly the QoE. The straightforward solution to this problem would be lowering the gain factor (to, say, x1.2 or x1.15). However, this would not only cause a delay in reaching the new (higher) channel capacity, but also an undesirable oscillation in $\lambda(t)$. Indeed, the second reason why BBR's exploration strategy is unsuitable in our context is

that its fixed pattern of gains ignores the history of the already explored target rates, along with the QoS parameters (*BDP* and *inflight*). Unlike BBR's, our strategy does take exploration history into account to dynamically modify its gain factor and, by doing so, yields: *i)* a smoother $\lambda(t)$, thus avoiding too many oscillations in the video encoding bitrate, and ultimately in the QoE; and *ii)* a more efficient channel use. We manage to be conservative when a recent tentative increase of $\lambda(t)$ over the last known $c_{\text{btl}}(t)$ has caused *drainNeeded*(t), but more aggressive if this has not happened so recently, which usually means that $c_{\text{btl}}(t)$, hence λ_{OOP}, have themselves increased.

Like BBR's, our exploration strategy is based on cycles, or exploration periods, $T_{\text{exp}}(t)$, but their duration is not fixed. Our steps for $\lambda(t)$, called $\lambda_{\text{exp}}(t)$, are not fixed either, and do not depend exclusively on $\lambda(t)$. In fact, our exploration period and step depend on our exploration gain $g(t)$, which is an integer between 1 and $g_{\max}$:

$$T_{\text{exp}}(t) = (g_{\max} - g(t) + 1)\ \Delta t, \quad \lambda_{\text{exp}}(t) = g(t)\ \lambda_{\text{exp,min}}.$$

[NB: in our tests, we set $g_{\max} = 50$ and $\lambda_{\text{exp,min}} = 10$ kb/s.]

At the end of each exploration period, we increase, maintain or decrease $g(t)$, and then set $T_{\text{exp}}(t)$ and $\lambda_{\text{exp}}(t)$ for the new period. Before explaining our criteria for this decision, we want to stress that, as shown in Figure 5b, our steps of variable width and height allow us to be cautious in the vicinity of a constant channel capacity, whatever the current target rate. On the other hand, when there is indeed a newer, higher channel capacity to be discovered and matched, the simultaneous decrease of $T_{\text{exp}}(t)$ and increase of $\lambda_{\text{exp}}(t)$, both due to an increase of $g(t)$, yield an "exponentialish" increase of $\lambda(t)$, which matches quickly the new $c_{\text{btl}}(t)$.

We decide to increase, maintain or decrease $g(t)$ by comparing the network QoS behavior during the just-finished period with its previous, saved/logged behaviors since congestion was last detected (and $g(t)$ was initialized to $g_{\min}$). We define "network QoS behavior" by means of a channel rating function $R(\bar{\lambda})$ which combines four QoS parameters measured or estimated during an exploration period, namely the average, standard deviation, and maximum value of $RTVL(t)$, and the amount of $qdData(t)$:

$$R(\bar{\lambda}) = [a\ \overline{RTVL}(t) + b\ \sigma(RTVL(t)) + c\ \max(RTVL(t)) + d\ qdData(t)]\ /\ \bar{\lambda}. \quad (3)$$

The four coefficients a, b, c and d are meant to balance the contribution of these different QoS parameters of the rating function, i.e., to give more relative importance to any of them. In any case, they must all be positive since large $R(\bar{\lambda})$ values represent undesirable tendencies, because all its four QoS parameters do: large or highly variable/unpredictable latency, or a lot of undelivered data. [NB: in our tests, we used $(a, b, c, d) = (1, 10, 1, 2)$.]

Based on our channel rating function, what we save/log is what we call "states", defined as couples $s(\bar{\lambda}) = (\bar{\lambda}, R(\bar{\lambda}))$, where $\bar{\lambda}$ is the average value of $\lambda(t)$ during a just-finished exploration period. As time goes by, our C^{3}G algorithm gradually builds a "network state dictionary", $S = \{s(\bar{\lambda})\}$, by storing each new couple $s(\bar{\lambda})$ if $\bar{\lambda}$ had not been explored yet, or by possibly updating its $R(\bar{\lambda})$ if it had, as explained below. Note that the discretization effected by $\lambda_{\text{exp,min}}$ helps accelerate searches in this state dictionary, which are necessary to choose between the following three candidates for the new target $\lambda(t)$:

$$\lambda_+ = \bar{\lambda} + \lambda_{\text{exp}}(t), \quad \lambda_= = \bar{\lambda}, \quad \lambda_- = \bar{\lambda} - \lambda_{\text{exp}}(t).$$

After calculating these three candidates, our algorithm traverses the following decision tree, which we designed to favor increases in $\lambda(t)$: note that the second sub-case of B.b.2 is the only situation where λ_- is chosen, and bear in mind that $R(\lambda_1) < R(\lambda_2)$ means that the network behaves better for λ_1 than for λ_2.

A. If $\lambda_=$ is not found in S:
   a. If λ_+ is not found in S: λ_+ is chosen because the main mission of our exploration strategy is precisely to aim higher.
   b. If λ_+ is found in S: if $R(\lambda_+) > R(\bar{\lambda})$, λ_+ is chosen because it seems that the network behaves better now for the just-explored rate than in the past for a higher one; otherwise, $\lambda_=$ is chosen, hoping that things will improve during the next exploration period.

B. If $\lambda_=$ is found in S:
   a. If λ_+ is not found in S: if $R(\lambda_=) > R(\bar{\lambda})$, λ_+ is chosen because it seems that the network behaves better now for the just-explored rate than in the past; otherwise, $\lambda_=$ is chosen, hoping as in case A.b that...
   b. If λ_+ is found in S:
      1. If $R(\lambda_+) > R(\bar{\lambda})$, λ_+ is chosen because it seems that the network behaves better now for the just-explored rate than in the past for a higher one (see case A.b).
      2. If $R(\lambda_+) < R(\bar{\lambda})$: if $R(\lambda_=) > R(\bar{\lambda})$, $\lambda_=$ is chosen because it seems that the network behaves better now for the just-explored rate than in the past, but not as much better as in case B.a; otherwise, λ_- is chosen because it is the only sensible option.

   Once λ_+, $\lambda_=$ or λ_- is chosen, $R(\bar{\lambda})$ is updated if needed.

V. IMPLEMENTATION DETAILS

In this Section we address the implementation of our C^{3}G algorithm explained in the previous two Sections, and how we had to tailor it to PlayGiga's CG platform [28]. In particular, we give some details on how we monitor the network parameters, deal with packet losses and retransmissions, and set $\lambda(t)$ through $v(t)$, to achieve effective congestion control in a real environment.

But before doing so, we want to highlight the importance of running at the server (vs. at the client) our C^{3}G algorithm to detect congestion and adapt $\lambda(t)$. We believe this has at least the following three advantages:

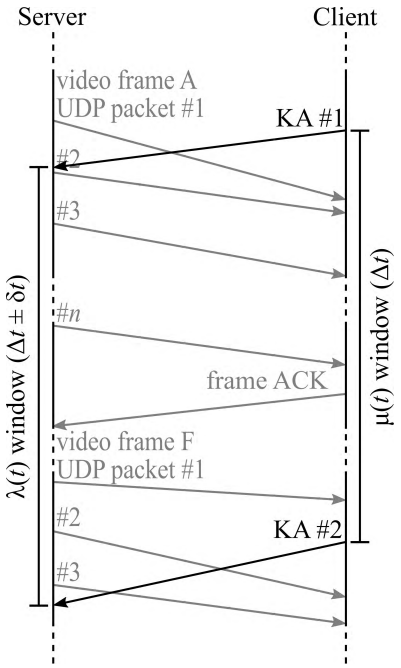

FIGURE 6. Misalignment of the time windows used to measure $\lambda(t)$ at the server and $\mu(t)$ at the client.

1. it implies less action-reaction time, since the decisions are taken where the video encoder operates, thus saving the transmission time needed to report any decision from the client to the server;
2. from the deployment viewpoint, it is easier to update one server than multiple (types of) clients;
3. a centralized algorithm helps manage multi-player services which can share the same streaming channels.

A. MISALIGNMENT OF THE SERVER AND CLIENT TIME WINDOWS

Our C^{3}G algorithm takes its decisions based on $\mu(t)$, which must be periodically measured by the client and reported to the server. It is therefore the server's responsibility to calculate $\lambda(t)$ for the same time window used by the client for $\mu(t)$, as we assumed in Subsections III-B and III-C.

In PlayGiga's CG platform, the client does measure $\mu(t)$ and send it to the server aboard TCP KA (KeepAlive) packets at regular time intervals. [NB: TCP's KA packets are typically meant to carry no meaningful data, and just used by one peer to check as needed that the other peer and the link between the two are still "alive", which is confirmed (or not) by the reception (id.) of replies to the KA probes.] Although this is not required by our C3G algorithm, the period used at the client for these KA reports is the same Δt used at the server for the congestion control loops: see Subection III-C. This is shown by the right side of Figure 6: when KA packet #2 is due, Δt after #1, the client adds the amount of data carried by all UDP packets received during that $\mu(t)$ time window (from #1 of video frame A until #1 of video frame F), divides it by Δt, and reports the resulting $\mu(t)$ via KA packet #2.

The main goal of Figure 6, however, is to illustrate the misalignment of the $\mu(t)$ time window described above with the corresponding $\lambda(t)$ window on the left, which is not only shifted "down" in time, but also of a different size, due to both the uplink and downlink propagation delays: the first affects the KA packets, which leads to the time span difference $\pm\delta t$; more importantly, the downlink delay of the UDP packets, and their potential losses, affect the number of them taking part in the computation of $\mu(t)$. Both delays yield a noisy function of $\lambda(t) - \mu(t)$ differences, thus a noisy estimation of $qdData(t)$ for our congestion detection and adaptation strategy.

To filter this noisy function, we consider only its positive values above a certain threshold ϵ, which is periodically updated to hold the maximum value of $\lambda(t) - \mu(t)$ reported without signs of an increasing RTVL. In our implementation, the first condition of Equation 1, $qdData(t_{\text{now}}) > 0$, was replaced with $\bar{\lambda}(t_{\text{now}}) - \bar{\mu}(t_{\text{now}}) > \epsilon$.

Note that all this does not affect the computation of $RTVL(t)$ at the server, for which we used exactly the procedure illustrated by Figure 1:

0. the server takes a timestamp S_0 before sending each raw frame output by its game engine to its video encoder, and, once it is encoded and fragmented in n UDP packets,

1... the server sends these UDP packets to the client, which might not receive some of them;

n. when all n UDP packets have been received, or after the presentation timeout described below, the client sends the re-assembled frame to its video decoder, and, just before displaying it, sends a frame ACK to the server;

A. upon reception of that ACK, the server takes a second timestamp S_{ACK} and computes $RTVL(t)$ for that frame.

B. PRESENTATION AND RETRANSMISSION TIMEOUTS

Contrary to what Figure 1 implies, a real CG client must consider UDP packet losses and might therefore have to stop waiting to receive all packets of a video frame before decoding and displaying it. A "presentation timeout" is typically imposed on the difference $C_n - C_1$, and if it is reached before all packets of a video frame have been received, the frame is nevertheless decoded and displayed, obviously with decoding errors. In PlayGiga's CG client, the presentation timeout was set to 100 ms.

Note that, regardless of the value chosen for the maximum acceptable latency, $RTVL_{\text{th}}$ (see Subsection IV-A), its actual saturation value, $RTVL_{\text{max}}$ (see Figure 3), should be equal, in the general case of a frame with packet losses, to the transmission delay of its first packet plus its presentation timeout. But in the particular case of a completely lost frame for which no UDP packet is ever received, $RTVL_{\text{max}}$ would be unbounded. PlayGiga's CG client handles this particular case by sending to the server an ACK for the completely lost frame after a later frame has been received, and with a low

priority (i.e., at the end of the control algorithm execution), so $RTVL_{\max}$ may in fact reach values as high as 350 ms.

Besides the presentation timeout, real CG clients also have a "retransmission timeout" associated to an application buffer which allows for requesting retransmissions of not-yet-received (and thus potentially lost) UDP packets to the server. In PlayGiga's CG client, the retransmission timeout was set to 30 ms.

Dealing with packet losses and retransmissions requires two modifications in our adaptation strategy:

1. In the event of network congestion, the client requests the retransmission of UDP packets which are lost or delayed for too long, so the server-sent rate does not only depend on the video encoding parameters, but also on the retransmission percentage. To allow for the bottleneck buffer to drain, this must be taken into account when computing λ_{drain}: $r(t)$ being the fraction of retransmissions with respect to $\lambda(t)$ for the considered analysis time window, Equation 2 becomes
$$\lambda_{\text{drain}} = (1 - r(t))\ \bar{\mu}(t_{\text{drain}}) - \frac{qdData(t_{\text{drain}})}{\Delta t}.$$
2. Since retransmissions and losses also have an impact on the network QoS, the rating function of Equation 3 must also be modified to include them:
$$R(\bar{\lambda}) = [a\ \overline{RTVL}(t) + \ldots + d\ qdData(t) + e\ r(t) + f\ \bar{l}(t)]\ /\ \bar{\lambda}.$$
[NB: packet losses $l(t)$ are calculated by the client, for each frame, as the fraction of its non-received UDP packets at the time it is sent to the decoder.]

C. SERVER-SENT VS. VIDEO ENCODER BITRATES

Setting a new target server-sent bitrate, $\lambda(t)$, by setting a new target video encoder bitrate, $v(t)$, is not straightforward, since different factors are involved, such as the CG platform used (there are several proprietary solutions, notably from NVIDIA, Intel and AMD), the video encoder provided by it and its API, and the nature and complexity of the video sequences to be encoded, i.e., the output of the game engine.

Besides, specifying a particular $v(t)$ value does not necessarily mean that the real output bitrate of the video encoder will match it exactly and even less instantly, so our C^{3}G algorithm always modifies $v(t)$ by taking into account the $\lambda(t)/v(t)$ ratio of previous iterations.

VI. EXPERIMENTAL RESULTS

In the experimental tests we carried out to compare C^{3}G's performance with BBR's in a real-world CG platform, we focused in particular on its capabilities to: *i)* rapidly adapt to network congestion while minimizing its negative effects on the user's QoE; and *ii)* use the maximum channel capacity in stable channel conditions. We tested both algorithms using PlayGiga's CG platform [28], which provides realistic conditions in an end-to-end system, including transmission over a real WAN (Wide Area Network) with realistic QoS degradation, e.g., propagation delay, spurious losses, jitter, etc. Additionally, we implemented limitations on the channel capacity in the client side to test the response of both BBR and C^{3}G under different bandwidth conditions.

Subsection VI-B describes the design of our test benchmark, and Subsections VI-C and VI-D report and discuss on the results obtained by C^{3}G vs. BBR in the two scenarios described in Section IV. But first, Subsection VI-A explains how we had to adapt BBR, originally designed to operate at the transport layer, and using TCP. Since recent works have used it at the application layer [29] in the Gaming Anywhere [24] CG platform, by following the same approach, we could compare both methods in a common UDP-based CG platform.

A. BBR's ADAPTATION TO PLAYGIGA's CG PLATFORM

Given that video transmission is based on UDP in PlayGiga's CG platform, we had to adapt BBR to operate in the absence of TCP parameters. Therefore, we had to derive BBR's parameters *RTprop*, *BtlBW* and *inflight*, described in Subsection III-A, from C^{3}G's QoS parameters $RTVL(t)$, $\mu(t)$ and $\lambda(t)$, described in Subsection III-B.

This called for changes in the implementation of BBR's rate control strategies, which were aimed at keeping intact its two cornerstones: *i)* a gain-based adaptation scheme to react to channel capacity drops; and *ii)* a probe cycle algorithm to explore higher channel bandwidth limits. We based our re-implementation of BBR on its implementations at the transport layer for *ns-3* [16], [30], and on its adaptations for the application layer [29].

1) DERIVATION OF BBR's PARAMETERS FROM C^{3}G's

1. We replaced TCP's *RTT* by $RTVL(t)$, which is meant to be analogous to *RTT*, only "frame-wise" (i.e., at the application level), instead of "packet-wise". We thus defined BBR's *RTprop* as the minimum value of all $RTVL(t)$ samples in a time window w (we used $w = 2\ \Delta t$).
2. We obtained BBR's *BtlBW* from $\mu(t)$, which represents a valid estimation of $c_{\text{btl}}(t)$ in the case of congestion, as explained at the very end of Subsection III-C.
3. We estimated the amount of unacknowledged sent bits as $inflight = (\lambda(t) - \mu(t))\ w'$ (we used $w' = 1$ s).

2) IMPLEMENTATION OF BBR NETWORK ADAPTATION STRATEGY

BBR follows a gain-based strategy for rate control with two phases, Startup and ProbeBW, which apply different strategies to derive increasing/decreasing data rate gains. Figure 7 shows the scheme of these two phases as implemented in our test platform, following the guidelines of [29]. In BBR's original implementation, these gains modify the channel bandwidth estimation *BtlBw*, which ultimately modifies the transmission rate. With the same spirit, in our implementation, these gains are applied directly to the target encoding bitrate $\lambda(t)$.

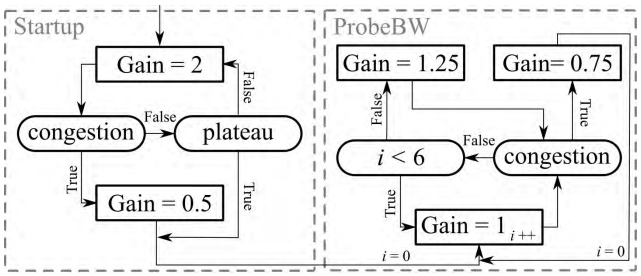

FIGURE 7. Scheme of the BBR algorithm for the adaptation to the network channel capacity implemented in PlayGiga's CG platform. [NB: *i* represents an iteration of the algorithm.]

1. Startup: In this initial phase, the bitrate is iteratively doubled (x2 gain). In each iteration, congestion is checked and, if detected, the bitrate is halved (x0.5) to allow the bottleneck queue to drain. After bitrate reduction, or when the bitrate reaches a predefined upper limit called "plateau", BBR moves to the ProbeBW phase.
2. ProbeBW: In this phase, BBR uses an approach called "gain cycling" to reach a higher throughput: the bitrate is moderately (x1.25) and, whenever congestion is detected, reduced (x0.75) and then stabilized (x1) for six iterations.

Note that BBR also includes an additional phase, the ProbeRTT cycle, which is invoked if *RTprop* has not decreased in the last ten seconds. In that phase, BBR reduces *CWND* to a minimum value (four packets) to estimate *RTprop*. This strategy is not valid in a CG application, since reducing the bitrate to a minimum necessarily results in either a degradation of video quality or an increase of *RTVL* beyond acceptable QoE limits. Therefore, in our BBR re-implementation we omitted the ProbeRTT phase.

B. DESCRIPTION OF THE TEST BENCHMARK

PlayGiga's end-to-end CG platform, that we used for our experimental tests, is depicted in Figure 8 and described below.

1. Server: its video encoder ran on an AMD RadeonTM RX 480 GPU and generated a 720p@30fps video bitstream compressed according to the AVC/H.264 standard. In all tests, we used a peak-constrained bitrate control, with a moderate range of QP (Quantization Parameter) values, namely [22, 40], to generate a stable bitrate output $\lambda(t)$ by acting on $v(t)$, as described in Subsection V-C.
2. Network: the client was nine WAN hops (and ∼30 km) away from the server. In its local network, a separate PC implemented the TBF (Token Bucket Filter) within the traffic control queueing disciplines [31], acting as a limiter for the channel capacity. The TBF simulated a configurable bandwidth limitation $c_{\text{btl}}(t)$ and the typical queuing delay of 100 ms [12], [29]. [NB: this implies that packets with a delay over 100 ms are discarded.] To avoid interference from other connections in the results, only the incoming traffic from the game server was shaped by the TBF.
3. Client: it was implemented on a PC equipped with an Intel Core i7-6500U@2.5-3.1GHz CPU with 16 GB of RAM, and an integrated Intel HD Graphics 520 GPU.
4. Video content: all tests were performed using the game "Sonic & All-Starts Racing Transformed", and the same "race" and game stage, for the sake of fair comparison. This game is very demanding for the video encoder given its high-frequency textures and fast motion (see Figure 9). The encoder was therefore able to produce a high range of bitrate values as commanded by C^{3}G.

C. C^{3}G's REACTIVE PERFORMANCE

We compare the performance of C^{3}G and BBR in the case of network congestion. The adaptation procedures to congestion of both methods are described in Subsections IV-A and VI-A. In our tests, the channel suffers a "step-shaped bandwidth drop" because its capacity is instantly reduced from an initial value, $c_{\text{btl,init}}$, to a limited one, $c_{\text{btl,lim}}$. This same method has been used in previous works to test the resilience of channel-adaptive techniques for interactive real-time applications [32]. To characterize the resilience of C^{3}G and BBR to such channel capacity drops, we measured three QoS parameters:

1. AP (Adaptation Period): the lapse of time, after the channel capacity drop, during which $RTVL(t)$ exceeds the 100 ms playabilty threshold.
2. $RTVL_{\text{peak}}$: the maximum $RTVL(t)$ value during AP.
3. L_{AP}: the number of frames suffering losses during AP.

In our tests, we used a fixed set of values for $c_{\text{btl,lim}}$, namely {5, 7, 9} Mb/s, and an initial capacity $c_{\text{btl,init}}$ proportional to $c_{\text{btl,lim}}$: $c_{\text{btl,init}} = \alpha_c\, c_{\text{btl,lim}}$. To cover a wide enough range of capacity drops, for each value of $c_{\text{btl,lim}}$, we tested $\alpha_c \in \{1.25, 1.5, 1.75, 2, 2.5\}$, and tried four times each $(c_{\text{btl,lim}}, \alpha_c)$ combination, for a total of sixty network congestion tests.

Figure 10 shows the results of this test set. All results are given with respect to the ratio of channel capacity reduction α_c, and for different values of $c_{\text{btl,lim}}$. [NB: in practice, the initial channel capacity was defined by an initial target encoder bitrate $v_{\text{init}} = \alpha_c\, c_{\text{btl,lim}}$; as the output bitrate λ_{init} does not exactly match v_{init}, α_c values in Figure 10 slightly deviate from the set {1.25, 1.5, 1.75, 2, 2.5}, but this does not affect our conclusions on the results.]

In addition, Figure 11 shows an example of the evolution in time of the bitrates and QoS parameters for one specific test case, that illustrates the behavior of each method.

The results for the three QoS parameters in Figure 10 show that C^{3}G outperforms BBR in terms of resiliency to network congestion, as we explain in the rest of this Subsection, by analyzing its three sub-Figures one by one. Note that there seems to be no dependence on $c_{\text{btl,lim}}$ of either of these three parameters, for either BBR or C^{3}G.

Figure 10a shows that BBR's AP values are consistently higher than C^{3}G's, for all values of α_c and $c_{\text{btl,lim}}$. C^{3}G's APs are between 0.4 and 0.8 seconds, while most of BBR's exceed 1 s, even for moderate capacity drops ($\alpha_c < 2$).

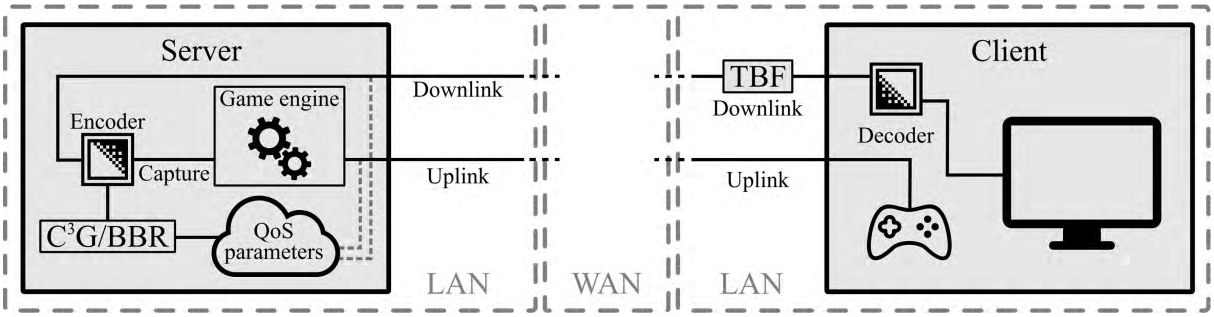

FIGURE 8. Block scheme of our test CG platform, whose server has a game engine, a video encoder and a rate-control module, and whose client includes a video decoder and a TBF (Token Bucket Filter) to shape the incoming data rate.

FIGURE 9. Screen captures of the "Sonic & All-Starts Racing Transformed" game that was used for the experimental tests.

This difference in AP lengths is illustrated by the $RTVL(t)$ graphs in Figures 11b and 11d. Furthermore, BBR's AP values are more variable and frequently over 2 s throughout all the range of tested values for α_c. Long APs have a highly negative impact on the QoE, as they are (by definition) long periods during which the end-to-end latency exceeds the playability threshold, and, besides, they increase the probability of incurring packet losses (see Figure 3).

In particular for C^{3}G, and less so for BBR, there is a moderate correlation between AP and α_c. This stems from the fact that $qdData(t)$, i.e., the amount of data to be drained, which is measured instantly after the channel capacity drop, is proportional to $c_{btl,init} - c_{btl,lim}$, which increases with α_c. C^{3}G achieves lower AP values thanks to its quicker adaptation to congestion by means of the draining mechanism described in Subsection IV-A, which derives λ_{drain} and T_{drain} from an estimate of $qdData(t)$. Instead, BBR's fixed bitrate reduction ratios (x0.75 or x0.5) result in longer APs when there is a lot of queued data, as several iterations are needed.

Figure 10b shows $RTVL_{peak}$ values for both algorithms, which are comparable and highly correlated with α_c. The lack of apparent correlation between $RTVL_{peak}$ and $c_{btl,lim}$ could be explained by how the congestion control mechanisms react to a sudden channel capacity drop: $RTVL_{peak}$ is reached very shortly after the drop, when the draining procedure has not yet started; as a consequence, $RTVL_{peak}$ solely depends on the channel capacity reduction ratio, and this for both rate control algorithms. The $RTVL(t)$ graphs in Figures 11b and 11d (for BBR and C^{3}G respectively) show that $RTVL_{peak}$ reaches $\sim$350 ms, which corresponds to the $RTVL_{max}$ value of the system, as described in Subsection V-B.

Figure 10c illustrates how C^{3}G outperforms BBR as well in terms of L_{AP}, since the number of lossy frames is consistently higher for BBR for all values of $\alpha_c > 1.75$ (for lower channel capacity drops, i.e., below $\sim$42%, there are simply no losses). C^{3}G manages to keep L_{AP} below ten frames always, but BBR exceeds this value for moderate α_c values, and reaches $L_{AP} > 20$ for quite a few experiments. While for both algorithms L_{AP} increases with α_c, this tendency is much slower for C^{3}G, again due to its faster adaptation to network congestion (BBR's longer draining periods are more likely to cause bufferbloat, and therefore losses): see for example the $RTVL(t)$ and $l(t)$ graphs in Figures 11b and 11d.

Finally, by considering Figures 10b and 10c together, it can be seen that losses kick in when $RTVL = RTVL_{max}$: see Subsection III-B. This condition is met for $\alpha_c > 1.75$, but this is true both algorithms.

What does differentiate C^{3}G from BBR is the draining strategy, which in our case is adaptive, and based on an estimate of $qdData(t)$, as already mentioned in the analysis of AP and L_{AP}. This allows C^{3}G to have shorter APs and fewer lossy frames, thus limiting much better than BBR the negative impact of channel capacity drops on QoE.

D. C^{3}G's PROACTIVE PERFORMANCE

We designed the next set of tests to analyze the performance of C^{3}G vs. BBR in re-adapting to the channel capacity c_{btl},

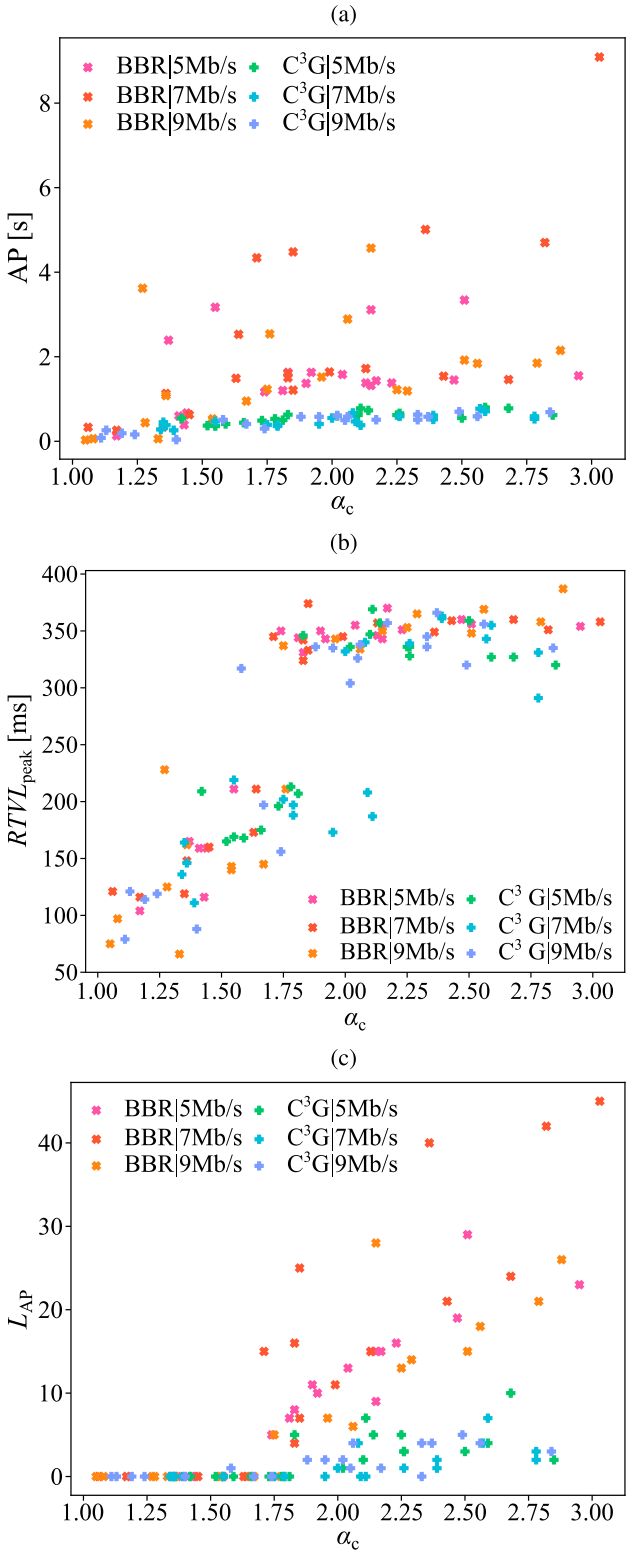

FIGURE 10. Performance results of BBR vs. C³G in the reactive bitrate decrease tests, for different α_c and $c_{btl,lim}$ values. (a) AP (Adaptation Period). (b) $RTVL_{peak}$: Maximum $RTVL(t)$ during AP. (c) L_{AP}: Number of lossy frames during AP.

either after a bitrate decrease due to congestion, or because c_{btl} has increased. The adaptation strategies of both methods are the ones described in Sections IV-B and VI-A.

We carried out two different tests, with the same fifteen ($c_{btl,lim}$, α_c) combinations described in the previous Subsection ($c_{btl,lim} \in \{5, 7, 9\}$ Mb/s; $\alpha_c \in \{1.25, 1.5, 1.75, 2, 2.5\}$), in two different channel conditions:

1. Channel capacity stability period: we evaluated BBR and C³G in terms of effective channel use and QoS metrics during 60 s at $c_{btl,lim}$.
2. Channel capacity increase: we measured the time it took each algorithm to adapt to the new channel capacity, which had increased from $c_{btl,lim}$ to $c_{btl,init}$.

Figure 11 shows an example of these tests for both algorithms. Note that the graphs of Figure 11 also include, in their left-most part, an initial channel capacity drop phase already analyzed in the previous Subsection.

The effective channel use and QoS results of the stability test are captured in Table 1, where both algorithms have been compared in terms of:

1. Effective channel use (%): average and standard deviation values of $\lambda(t)/c_{btl}(t)$.
2. $RTVL_{\{avg,peak\}}$ (ms): $RTVL(t)$ average and maximum values.
3. L_{rate} (fps): number of lossy frames per second.
4. R_{rate} (kb/s): amount of retransmitted data per second.

These measures have been computed over a time window which starts at the end of the AP and ends when the channel capacity stability period does, as shown in Figure 11. Again, C³G outperforms BBR for all measures.

Regarding effective channel use, C³G's is 2.7–6.3% higher than BBR's, thanks to its smoother bitrate increase strategy. An example of the evolution of $\lambda(t)$ for both algorithms is depicted in Figures 11a and 11c. BBR's gain cycling and memoryless bitrate increase strategy produces too much oscillation in $\lambda(t)$, which repeatedly exceeds $c_{btl,lim}$, which causes congestion detections (as indicated in the graph), which in turn calls for new $\lambda(t)$ decrease and increase cycles. Instead, C³G keeps a history of states from previously explored bitrates (see Subsection IV-B), which helps minimize the probability of exceeding $c_{btl,lim}$, while still making an efficient use of the channel capacity. Furthermore, in the limited cases where $\lambda(t)$ does exceed $c_{btl,lim}$, C³G's reactive bitrate decrease strategy (see Subsection IV-A) manages to drain the congestion without incurring heavy losses.

Besides using the channel capacity more efficiently, C³G provides even more important advantages for the QoE in CG applications, since it considerably reduces the $RTVL_{\{avg,peak\}}$ and $\{L, R\}_{rate}$ values, and yields zero lossy frames in all tests. Instead, BBR's recurrent excess of $\lambda(t)$ increases the latency, as well as the retransmission requests and lossy frames.

Table 2 presents the results of the channel capacity increase test. Both algorithms are compared in terms of the EP (Exploration Period), which is the time used to recover $\lambda(t) = c_{btl,init}$ from $\lambda(t) = c_{btl,lim}$. The results show that C³G's EP values are higher than BBR's, but this is because C³G's design prioritizes good performance on key QoS metrics for CG applications (minimal losses followed by low latency)

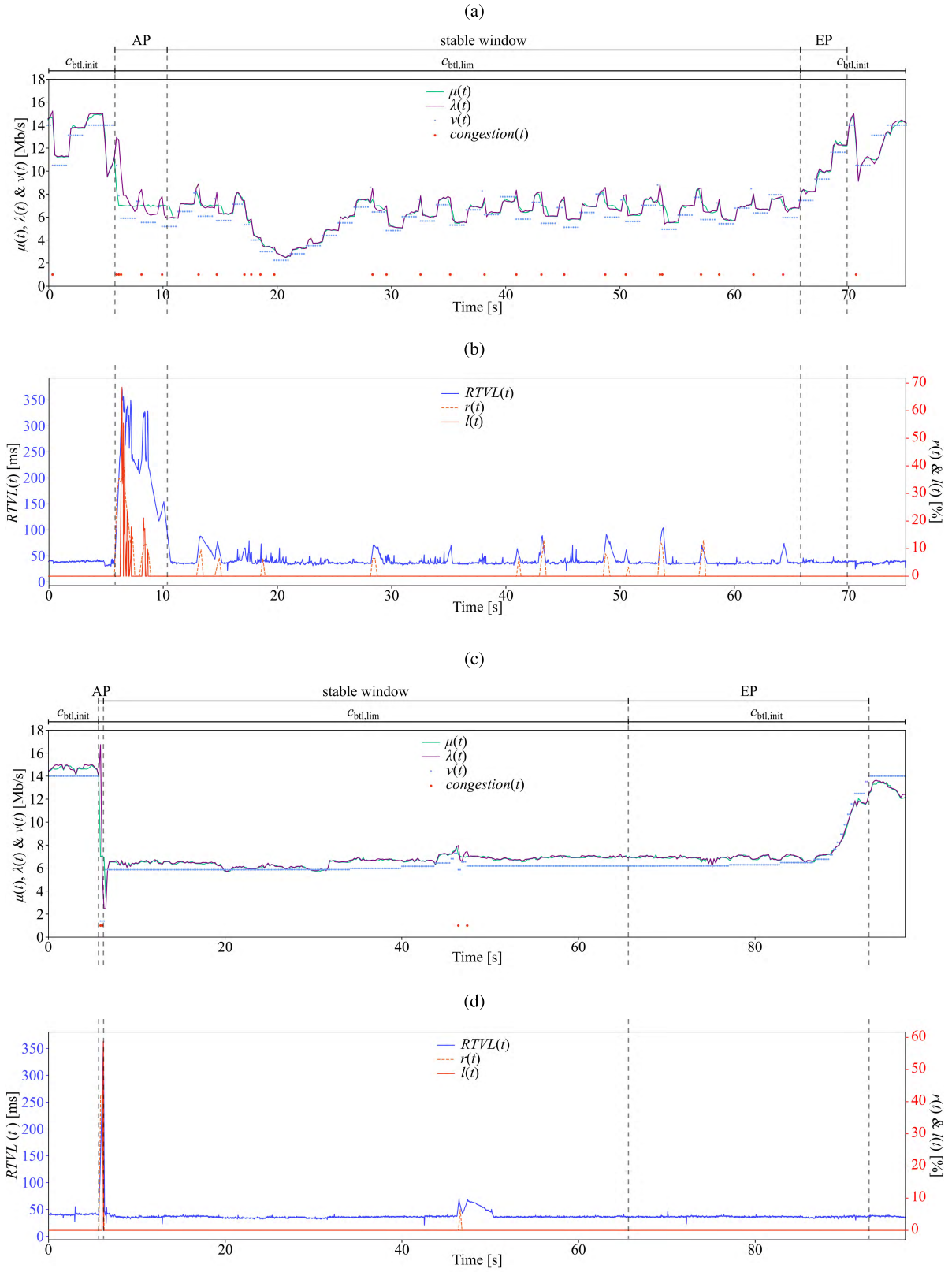

FIGURE 11. Example of the behavior of BBR vs. C^{3}G in the three test cases. From left to right: Sudden channel capacity drop from $c_{btl,init}$ = 14 Mb/s to $c_{btl,lim}$ = 7 Mb/s, stable period of 60 s, and channel capacity recovery from $c_{btl,lim}$ to $c_{btl,init}$. (a) BBR's bitrate metrics: $\lambda(t)$, $\mu(t)$, $v(t)$, *congestion*(t). (b) BBR's QoS metrics: *RTVL*(t), $r(t)$, $l(t)$. (c) C^{3}G's bitrate metrics: $\lambda(t)$, $\mu(t)$, $v(t)$, *congestion*(t). (d) C^{3}G's QoS metrics: *RTVL*(t), $r(t)$, $l(t)$.

TABLE 1. Effective channel use and QoS results for the channel capacity stability test. The results for each value of $c_{btl,lim}$ have been averaged over all tests (different values of α_C).

	$c_{btl,lim}$ (Mb/s)	Avg. ch. use (%)	σ ch. use (%)	$RTVL_{avg}$ (ms)	$RTVL_{peak}$ (ms)	L_{rate} (fps)	R_{rate} (kb/s)
BBR	5	86.9	7.8	42.5	147.3	0.0026	38.3
	7	92.6	7.5	45.3	140	0	46.1
	9	93.4	6.8	45.2	163	0.0026	53.3
C^{3}G	5	93.2	3	36.8	97.2	0	3.7
	7	95.8	2.3	38.5	86.4	0	2.8
	9	96.1	2.1	39.6	88.3	0	3.7
Δ (C^{3}G-BBR)	5	**6.3**	—	**-5.7**	**-50.1**	**-0.0026**	**-34.6**
	7	**3.2**	—	**-6.8**	**-53.6**	**0**	**-43.3**
	9	**2.7**	—	**-5.6**	**-74.7**	**-0.0026**	**-49.6**

TABLE 2. Average EP by algorithm.

	EP $\pm$ σ [s]
BBR	8.3 $\pm$ 8.6
C^{3}G	36.2 $\pm$ 16.4

over fast bitrate increase when channel conditions improve. C^{3}G's bitrate increase strategy described in Section IV-B generates a "cold" start that accelerates if no congestion is detected (see Figure 11c). Instead, BBR's "greedy" bitrate increase strategy is faster (see Figure 11a). This is the only arguable advantage of BBR over C^{3}G, but in a CG application it hardly compensates for all its flaws discussed above.

VII. CONCLUSIONS

We have presented our C^{3}G algorithm, designed to help a UDP-based CG platform suffer minimal packet losses and keep latency within playability limits, even in the presence of severe downstream channel capacity drops. The strategy of our algorithm is twofold, since it does not only react when the channel capacity decreases, but also seeks proactively to use it as efficiently as possible when it increases.

Our network congestion model is inspired by that of Google's BBR, which uses TCP's RTT parameter to detect congestion. We propose a novel round-trip latency measure, RTVL, defined at the application layer and better suited than RTT, which is defined at the transport layer, to drive rate control decisions in real-time video streaming applications such as CG. RTVL proves to be a much better congestion predictor than losses, which typically occur when it is already too late to react. When C^{3}G detects congestion, unlike BBR, it decreases the target bitrate of the video encoder in an amount and during a drain period that both depend on the new estimated channel capacity. On the contrary, in reasonably stable network conditions, it proactively explores for higher acceptable downstream bitrates, and gradually builds a network state dictionary to characterize the channel capacity behavior. Both these reactive decreases and proactive increases of the server-sent bitrate may happen within a given game session, and without breaking the playability limits.

Indeed, the experimental results show how C^{3}G is clearly better suited than BBR to the CG context, since it manages to completely avoid losses for sudden downstream channel capacity drops of up to ~42%, and to keep RTVL below 100 ms, while continuously exploring for higher target bitrates, thus achieving an effective use of 93–96% of the available channel capacity.

Nevertheless, we already foresee some desirable improvements to our C^{3}G algorithm. For instance, it would be desirable to better align the $\lambda(t)$ and $\mu(t)$ time windows to achieve a better characterization of the network state. This could help us take more accurate decisions in the early stages of the congestion process, and reduce the exploration times, thus allowing us to be more greedy when increasing the target bitrate. Another avenue for improvement may be the use of other encoding parameters along with the final target video encoding bitrate. This could help C^{3}G produce smoother target bitrate transitions in its reactive phase, and have a deeper control over bitrate bursts during its proactive phase.

ACKNOWLEDGMENT

The authors would like to thank PlayGiga for their support during the design and development of our C^{3}G algorithm, and for collaborating with us during its tests on their CG platform [28]. P. Carballeira was with the Grupo de Tratamiento de Imágenes, Universidad Politécnica de Madrid, 28040 Madrid, Spain.

REFERENCES

[1] M. Jarschel, D. Schlosser, S. Scheuring, and T. Hoßfeld, "An evaluation of QoE in cloud gaming based on subjective tests," in *Proc. 5th IEEE Int. Conf. Innov. Mobile Internet Services Ubiquitous Comput. (IMIS)*, Jun./Jul. 2011, pp. 330–335.

[2] M. Jarschel, D. Schlosser, S. Scheuring, and T. Hoßfeld, "Gaming in the clouds: QoE and the users' perspective," *Math. Comput. Model.*, vol. 57, pp. 2883–2894, Dec. 2013.

[3] *State of the Internet Connectivity Reports*, Akamai, Cambridge, MA, USA, 2017.

[4] J. Gettys, "Bufferbloat: Dark buffers in the Internet," *ACM Queue-Virtualization*, vol. 9, pp. 40–54, Nov. 2011.

[5] Z.-Y. Wen and H.-F. Hsiao, "QoE-driven performance analysis of cloud gaming services," in *Proc. 16th IEEE Int. Workshop MultiMedia Signal Process. (MMSP)*, Sep. 2014, pp. 22–24.

[6] M. Claypool and K. Claypool, "Latency can kill: Precision and deadline in online games," in *Proc. 1st Ann. ACM SIGMM Conf. Multimedia Systems (MMSys)*, Feb. 2010, pp. 215–222.

[7] K. Raaen, R. Eg, and C. Griwodz, "Can gamers detect cloud delay?" in *Proc. 13th Annu. Workshop Netw. Syst. Support Games (NetGames)*, Dec. 2014, pp. 1–3.

[8] V. Clincy and B. Wilgor, "Subjective evaluation of latency and packet loss in a cloud-based game," in *Proc. 10th IEEE Int. Conf. Inf. Technol., New Gener. (ITNG)*, Apr. 2013, pp. 473–476.

[9] M. Claypool and D. Finkel, "The effects of latency on player performance in cloud-based games," in *Proc. 13th Annu. Workshop Netw. Syst. Support Games (NetGames)*, Dec. 2014, pp. 1–6.

[10] A. Sackl, R. Schatz, T. Hossfeld, F. Metzger, D. Lister, and R. Irmer, "QoE management made uneasy: The case of cloud gaming," in *Proc. IEEE Int. Conf. Commun. Workshops (ICC)*, May 2016, pp. 492–497.

[11] W. Cai, R. Shea, C.-Y. Huang, K.-T. Chen, J. Liu, V. C. Leung, and C.-H. Hsu, "A survey on cloud gaming: Future of computer games," *IEEE Access*, vol. 4, pp. 7605–7620, Aug. 2016.

[12] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," *Netw. Congestion*, vol. 14, pp. 20–53, Dec. 2016.

[13] R. Stewart and C. Metz, "SCTP: New transport protocol for TCP/IP," *IEEE Internet Comput.*, vol. 5, no. 6, pp. 64–69, Nov. 2001.

[14] E. Kohler, M. Handley, and S. Floyd, "Datagram congestion control protocol (DCCP)," Tech. Rep., 2006.

[15] C. Jin, D. X. Wei, S. H. Low, J. Bunn, H. D. Choe, J. C. Doylle, H. Newman, S. Ravot, S. Singh, F. Paganini, G. Buhrmaster, L. Cottrell, O. Martin, and W.-C. Feng, "FAST TCP: From theory to experiments," *IEEE Netw.*, vol. 19, no. 1, pp. 4–11, Jan. 2005.

[16] M. Claypool, J. W. Chung, and F. Li, "BBR'—An implementation of bottleneck bandwidth and round-trip time congestion control for ns-3," in *Proc. 10th ACM Workshop ns-3 (WNS3)*, Jun. 2018, pp. 1–8.

[17] T. Stockhammer, "Dynamic adaptive streaming over HTTP–: Standards and design principles," in *Proc. 2nd Ann. ACM Conf. MultiMedia Syst. (MMSys)*, Feb. 2011, pp. 133–144.

[18] M. Baldi and Y. Ofek, "End-to-end delay analysis of videoconferencing over packet-switched networks," *IEEE/ACM Trans. Netw.*, vol. 8, no. 4, pp. 479–492, Aug. 2000.

[19] S. Jarvinen, J.-P. Laulajainen, T. Sutinen, and S. Sallinen, "QoS-aware real-time video encoding how to improve the user experience of a gaming-on-demand service," in *Proc. 3rd IEEE Consum. Commun. Netw. Conf. (CCNC)*, vol. 2, Jun. 2006, pp. 994–997.

[20] S. Wang and S. Dey, "Addressing response time and video quality in remote server based Internet mobile gaming," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2010, pp. 1–6.

[21] S. Wang and S. Dey, "Rendering adaptation to address communication and computation constraints in cloud mobile gaming," in *Proc. IEEE Global Telecommun. Conf. (GLOBECOM)*, Dec. 2010, pp. 1–6.

[22] S. Wang and S. Dey, "Modeling and characterizing user experience in a cloud server based mobile gaming approach," in *Proc. IEEE Global Telecommun. Conf. (GLOBECOM)*, Nov. 2009, pp. 1–7.

[23] H.-J. Hong, C.-F. Hsu, T.-H. Tsai, C.-Y. Huang, K.-T. Chen, and C.-H. Hsu, "Enabling adaptive cloud gaming in an open-source cloud gaming platform," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 12, pp. 2078–2091, Dec. 2015.

[24] C.-Y. Huang, C.-H. Hsu, Y.-C. Chang, and K.-T. Chen, "GamingAnywhere: An open cloud gaming system," in *Proc. 4th ACM Int. Conf. MultiMedia Syst. (MMSys)*, Feb./Mar. 2013, pp. 36–47.

[25] M. Li, M. Claypool, and R. Kinicki, "WBest: A bandwidth estimation tool for IEEE 802.11 wireless networks," in *Proc. IEEE Conf. Local Comput. Netw. (LCN)*, Oct. 2008, pp. 374–381.

[26] M. Gerla and L. Kleinrock, "Flow control: A comparative survey," *IEEE Trans. Commun.*, vol. 28, no. 4, pp. 553–574, Apr. 1980.

[27] I. Slivar, L. Skorin-Kapov, and M. Suznjevic, "Cloud gaming QoE models for deriving video encoding adaptation strategies," in *Proc. 7th ACM Int. Conf. MultiMedia Syst. (MMSys)*, May 2016, p. 18.

[28] PlayGiga SL. *PlayGiga: Next Generation Cloud Gaming*. [Online]. Available: https://www.playgiga.com

[29] L. Wang, M. J. Suarez, and R. A. Domanico, "Adaptive bitrate streaming in cloud gaming," B.Sc. thesis, Worcester Polytech. Inst., Worcester, MA, USA, 2017.

[30] V. Jain, V. Mittal, and M. P. Tahiliani, "Design and implementation of TCP BBR in ns-3," in *Proc. 10th ACM Workshop ns-3 (WNS3)*, Jun. 2018, pp. 16–22.

[31] B. Hubert, T. Graf, G. Maxwell, R. van Mook, M. van Oosterhout, P. Schroeder, J. Spaans, and P. Larroy. (2004). *Linux Advanced Routing & Traffic Control*. [Online]. Available: http://lartc.org/

[32] L. De Cicco, S. Mascolo, and V. Palmisano, "Skype video responsiveness to bandwidth variations," in *Proc. 18th ACM Int. Workshop Network OS Support Digit. Audio Video (NOSSDAV)*, May 2008, pp. 81–86.

ALBERTO ALÓS received the Ingeniero de Telecomunicación degree (five-year engineering program) from the Universidad de Granada, Spain, in 2014. He is currently pursuing the Ph.D. degree with Grupo de Tratamiento de Imágenes (Image Processing Group), Universidad Politécnica de Madrid, Spain.

He worked for a few years in telecom companies. Since 2017, he has been a Researcher with the Grupo de Tratamiento de Imágenes (Image Processing Group), Universidad Politécnica de Madrid. His research interests include video coding and streaming with adaptive bitrate control.

FRANCISCO MORÁN received the Ingeniero de Telecomunicación degree (six-year engineering program) and the Doctor Ingeniero de Telecomunicación (Ph.D.) degree in communications from the Universidad Politécnica de Madrid (UPM), Spain, in 1992 and 2001, respectively.

Since 1992, he has been a member of the Grupo de Tratamiento de Imágenes (Image Processing Group), UPM. Since 1997, he has been a member of the Faculty of UPM, where he is currently an Associate Professor of signal theory and communications. He has been actively involved in European and Spanish research projects. Since 1996, he has also participated in the International Standardization Activities of the Moving Picture Experts Group (MPEG, formally ISO/IEC JTC 1/SC 29/WG 11), where he has been the Head of the Spanish Delegation, since 2006 and has served as an Editor and a Co-Editor for several standards. His research interests include modeling and coding of 3D objects and their adaptive transmission, and rendering for mixed/augmented reality applications.

PABLO CARBALLEIRA received the Ingeniero de Telecomunicación degree (five-year engineering program) and the Doctor Ingeniero de Telecomunicación (Ph.D.) degree in communications from the Universidad Politécnica de Madrid (UPM), Spain, in 2007 and 2014, respectively.

From 2008 to 2017, he was a member of the Grupo de Tratamiento de Imágenes (Image Processing Group), UPM. Since 2017, he has been an Assistant Professor with the Universidad Autónoma de Madrid, where he has also been a member of the Video Processing and Understanding Laboratory. He has been actively involved in European and Spanish research projects and in the International Standardization Activities of Moving Picture Experts Group (MPEG, formally ISO/IEC JTC 1/SC 29/WG 11) related to free-viewpoint and immersive 3D video. His research interests include video coding, computer vision, and quality of experience evaluation for immersive visual media, such as free-navigation and lightfield video.

DANIEL BERJÓN received the Ingeniero de Telecomunicación degree (five-year engineering program) and the Doctor Ingeniero de Telecomunicación (Ph.D.) degree in communications from the Universidad Politécnica de Madrid (UPM), Spain, in 2005 and 2016, respectively.

Since 2008, he has been a member of the Grupo de Tratamiento de Imágenes (Image Processing Group), UPM. He has been actively involved in European and Spanish research projects. His research interests include image processing, parallel processing, computer graphics, and real-time systems.

NARCISO GARCÍA received the Ingeniero de Telecomunicación degree (five-year engineering program), with the Spanish National Graduation Award, and the Doctor Ingeniero de Telecomunicación (Ph.D.) degree in communications, with the Doctoral Graduation Award, from the Universidad Politécnica de Madrid (UPM), Spain, in 1976 and 1983, respectively.

Since 1977, he has been a member of the Faculty of the UPM, where he is currently a Professor of signal theory and communications. He leads the Grupo de Tratamiento de Imágenes (Image Processing Group), UPM. He has been actively involved in Spanish and European research projects, also serving as an Evaluator, a Reviewer, an Auditor, and an Observer of several research and development programs of the European Union. He was a Co-Writer of the EBU Proposal, base of the ITU standard for digital transmission of TV at 34–45 Mb/s (ITU-T J.81). He was an Area Coordinator of the Spanish Evaluation Agency (ANEP), from 1990 to 1992, and the General Coordinator of the Spanish Commission for the Evaluation of the Research Activity (CNEAI), from 2011 to 2014. He was the Vice-Rector for International Relations of the UPM, from 2014 to 2016. His current research interests include digital video compression, computer vision, and quality of experience.

Dr. García was a recipient of the Junior and Senior Research Awards of the UPM, in 1987 and 1994, respectively.