On the benefits and caveats of exploiting Quality on Demand Network APIs for video streaming

Tuan Tran, Dylan Gageot, Christoph Neumann, Guillaume Bichot
Broadpeak
France

Abderrahmen Tlili, Karim Boutiba, Adlen Ksentini
Eurecom
France

ABSTRACT

The mobile industry - via forums such as the O-RAN Alliance and Linux Foundation CAMARA - is working on network APIs that allow a mobile network operator to expose network capabilities to application developers. One of these APIs is the Quality on Demand (QoD) API, which enables the application to ask for additional network resources for improved latency or bandwidth. In this work, we show how an intelligent content delivery network (CDN) can exploit these APIs to improve the quality of experience (QoE) of video streaming despite difficult network conditions by boosting the available network bandwidth at precise moments in time. As the bandwidth boost is only applied whenever necessary, we avoid the caveat of constantly and statically assigning network resources to a service. We propose two boosting strategies both relying on information provided by the video player via Common Media Client Data (CMCD). We implemented the approach and evaluated it on an emulation testbed and on top of an actual 5G O-RAN compliant network capable of running xApps and the CAMARA QoD API. Our evaluation shows the gains in terms of QoE but also highlights possible caveats and adverse interactions with the ABR algorithm of the video player.

CCS CONCEPTS

• **Information systems → Multimedia streaming**.

ACM Reference Format:
Tuan Tran, Dylan Gageot, Christoph Neumann, Guillaume Bichot and Abderrahmen Tlili, Karim Boutiba,, Adlen Ksentini. 2024. On the benefits and caveats of exploiting Quality on Demand Network APIs for video streaming. In *The 34th edition of the Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV '24), April 15–18, 2024, Bari, Italy*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3651863.3651882

1 INTRODUCTION

The network conditions experienced by end-users of a cellular network can greatly vary over time because of changing coverage, due to the user's mobility, interference, or because of highly loaded cells at peak hours. In the context of video streaming, Adaptive BitRate (ABR) can compensate for some of these difficult network conditions by having the video player requesting lower video bitrates. However, in the best case, this comes at the price of lower video quality and, at worst, may still lead to video freezes and rebuffering.

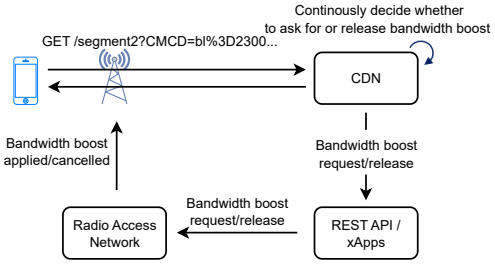

Figure 1: High-level architecture for bandwidth boost

The mobile industry is working on mechanisms and APIs - we call "Network APIs" - that allow an application to dynamically control and reconfigure the radio or the core network of a cellular network. Such control is made possible at a low-level using application called "xApps" deployed at the RAN Intelligent Controller (RIC) as proposed by the O-RAN Alliance [3], or at a high-level using REST APIs as specified by Linux Foundation (LF) CAMARA [9]. These APIs are not specifically tailored towards video streaming but are open to any application provider that needs fine control over a telecom operator's network.

In this work, we explore how an intelligent CDN (Content Delivery Network) can exploit these Network APIs to improve the quality of experience (QoE) of a given video streaming session despite difficult network conditions by temporarily requesting additional network bandwidth (that we call "bandwidth boost") at precise moments in time. The CDN hereby relies on information provided by the video player via CMCD [5], such as the buffer length. As the bandwidth boost is only applied whenever necessary, we avoid the caveat of constantly allocating a network slice to a service, which avoids reserving and wasting underlying network resources. The radio and network resources allocated to a bandwidth boost are immediately released when the video streaming session is back to safe conditions.

This concept is depicted in Figure 1. While serving the client and using the provided CMCD information, the CDN continuously evaluates whether to ask for a bandwidth boost for this particular user. If needed, the CDN requests a bandwidth boost using either the high-level REST APIs or the low-level xApps. Upon reception of the request, the mobile radio access network allocates temporarily more radio resources to this mobile user. The CDN releases the bandwidth boost as soon as it considers that the boost is not required anymore.

NOSSDAV '24, April 15–18, 2024, Bari, Italy

ACM ISBN 979-8-4007-0613-4/24/04…$15.00
https://doi.org/10.1145/3651863.3651882

The contributions of this work are as follows:

- We provide an overview of the different Network APIs that can be exploited by an intelligent CDN to ask for additional resources when needed. We hereby focus on two initiatives: (i) LF CAMARA with its high-level Quality on Demand (QoD) API [16] that has been demonstrated and used by some tier-one mobile network operators [13, 18, 19], and (ii) the low-level and radio-centric "xApps" that controls the radio access network (RAN) as proposed by the O-RAN Alliance.
- We propose two bandwidth boost strategies for a CDN to dynamically ask for additional bandwidth when needed: one strategy that solely focuses on avoiding buffer underruns; and another strategy that also takes into account the video quality retrieved by the player, therefore trying to guarantee a certain level of quality.
- We implemented the bandwidth boost strategies within a caching server. We deployed and evaluated this caching server and it's bandwidth boost strategies within an emulated network that exposes QoD API as defined by LF CAMARA. We also deployed and evaluated the caching server within a 5G network for which we developed an O-RAN xApp using the FlexRIC framework [17]. Our xApp is able to provide additional radio resources upon request.
- We evaluate the proposed strategies, demonstrate the overall benefits of the approach and show that we are able to improve video QoE in degraded network conditions. We also highlight limitations of some bandwidth boost strategies and configurations due to the bad interaction with the ABR adaptation algorithm of the video player. This opens perspectives to further study and design bandwidth boost strategies.

2 BACKGROUND

The LF CAMARA project defines a set of network APIs, allowing a mobile network operator to expose network capabilities to application developers. One of the specified APIs is the QoD API [16]. A high-level description of the QoD API is provided in Figure 2. The QoD API allows an authorized application to request a given QoS profile for a specified user equipment or network flow. The QoS profile defines the type and amount of network guarantees that the application asks for: an amount of network bandwidth and/or some latency guarantees. Once a QoD request is set, it can either be explicitly removed with a specific delete API call or the request is removed after an optional duration set in the initial QoD request. Several commercial offers provide the QoD API [13, 18, 19, 22].

LF CAMARA focuses on specifying northbound APIs; how these APIs are implemented or mapped to internal mechanisms to provide the requested network capability is implementation specific. For the QoD API, different approaches exist on how to implement a QoS profile: (i) the QoS profile is either applied on the existing bearer and PDU-session of the mobile device, or (ii) a new dedicated bearer and PDU-session is created that solely transports the traffic of the requesting application. A description of an implementation of the QoD API is provided in [14]. In this specific setup, each of the QoS profiles is mapped to a specific set of 3GPP specific parameters

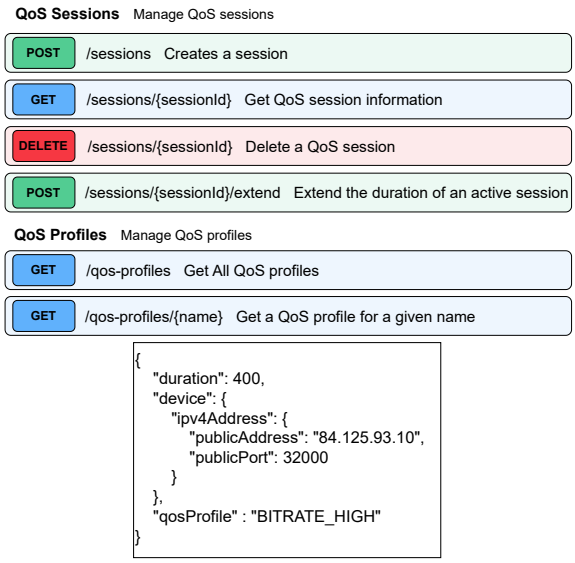

Figure 2: QoD API description. (top) Overall overview of API, (bottom) Example Json body of a new QoD session request.

that are applied to the existing bearer and PDU-session of the end-device. These 3GPP parameters typically include the 5G Quality Indicator (5QI) amongst others.

The advantage of re-using the existing bearer is that the profile can be applied quickly and that it does not require specific support from the mobile device, hence faster response time that is critical for some applications. The drawback is that the QoS profile is applied on all traffic of the mobile device, which includes traffic from other applications that did not request the QoD. While this is probably not a major issue with a mobile phone (the number of applications running in parallel and sharing the bearer is limited), it can be more problematic in the case of Fixed Wireless Access (FWA) as the connection is shared amongst many different devices in the home network. Creating a dedicated bearer and PDU-session may take more time (in the order of a few seconds) and most mobile devices do not support such a feature as of today. The advantage of such approach is that the traffic is well isolated from the rest and the network can provide better quality guarantees.

The Open Radio Access Network (O-RAN) is an initiative to develop an open and interoperable RAN ecosystem with open interfaces and specification to enable mobile network operators to deploy and manage RAN from different vendors. This helps to reduce the cost and complexity of deploying RANs, and also enable operators to innovate more quickly [15].

One of key components in the O-RAN architecture [4] is the RAN Intelligent Controller (RIC); it runs applications that can reconfigure RAN specific parameters and behaviors in "near-realtime". Doing so it is possible to reconfigure the scheduler of a base station, the number of resource blocks allocated to a specific user equipment etc. An xApp can therefore implement a QoD logic and APIs that are exposed to a specified application.

3 VIDEO STREAMING BOOST STRATEGIES AND USE CASES

We describe how to exploit QoD network APIs in the case of video streaming and detail the different bandwidth boost strategies.

We suppose that a CDN cache server is authorized to use a QoD network API of a cellular network, either via a CAMARA-like network API or by relying on xApps that are deployed within the cellular network. The cache server is either deployed within the cellular network, e.g. in regional edge locations, or runs as part of a global CDN [20]. When using the network API, the cache server uses the public IP address and port as an end-user device identifier. Such an identifier allows the network API to determine for which device the QoS profile must be applied.[1]

We describe two scenarios where each has its own strategy:

- *Avoiding buffer underruns:* this scenario solely focuses on avoiding video freezes (because of buffer underruns) while accepting a quality degradation due to players retrieving lower qualities in the ABR bitrate ladder. This scenario typically targets mobile phones that are subject to higher fluctuations of the radio signal, interference and coverage; a degradation of video quality is also less visible due to the reduced screen size of such devices. The main metric used in this scenario is the buffer length of the video player; the algorithm for this scenario is described in subsection 3.1.
- *Ensuring video quality:* this scenario tries to maintain a certain video quality for the end-user. This scenario targets connected TVs and set-top boxes that access the video content via FWA. Video quality variations are very visible on such devices and should thus be avoided. Further, as the receiving wireless device is not moving, the variation of the wireless signal will mostly depend on the load of the cell. The two metrics used in this scenario are the buffer length of the video player and the retrieved video quality; the algorithm for this scenario is described in subsection 3.2.

A cache server may implement only one or both of the above strategies. This may depend on the type of deployment (e.g. a deployment within the edge location of a mobile operator) and the capabilities and types of subscriptions offered by the mobile operator. For a network operator that does not offer any FWA subscription, the cache server should only implement the first strategy. For network operators that offer both modes, the cache server may either use information provided by the network operator to determine the type of wireless access, or exploit HTTP User-Agent to detect SmartTVs (hence connected via FWA) and use appropriate boost strategy.

The video players provide the information required for the above strategies via CMCD. Namely, we exploit the fields `bl`, specifying the video player buffer length in milliseconds and `br`, specifying the encoded bitrate of the requested object. For the latter information it is also possible for the server to exploit the URL path or file name which often (but not necessarily) provides hints on the video quality being requested by the player.

For all proposed boost strategies we try to minimize the "budget" spent and to release a bandwidth boost as soon as it is not more required to reach one of the above objectives. The budget may be expressed as the number of requests or a duration during which we requested a certain bandwidth. We detail and evaluate different budget metrics in the evaluation (see subsection 4.1).

[1]Note that other identifiers, including the private IP address or the IMSI number are also supported but these identifiers may require additional end-user device support or code to provide this information to the cache server.

Table 1: Experimental setup. (top) QoS profiles and (bottom) video ladder.

QoS-Profile	Requested bandwidth (Mbps)
High	6
Medium	4
Low	2
Live video encoding ladder (Mbps)	0.8, 1.5, 2.5, 3.5, 5

3.1 Buffer-based boost strategy

We propose a simple buffer-based boost strategy. We define two thresholds: *bl_min* and *bl_max*. *bl_min* defines the minimum buffer length upon which the bandwidth boost should be triggered. *bl_max* defines the buffer length upon which an ongoing bandwidth boost can be withdrawn. At reception of an HTTP request for a video segment the cache server extracts the CMCD information containing the buffer length attribute (`bf`). If the buffer length is below *bl_min* the cache server asks for a bandwidth boost: it invokes the network API to request a certain QoS profile for the end-user. Once the bandwidth boost has been requested, the cache server continues to observe the buffer length attribute (`bl`) of this streaming session. Once the buffer length is greater or equal than *bl_max* the cache server asks for the cancellation of the bandwidth boost. In the evaluation (subsection 4.1) we evaluate and discuss the impacts of the QoS profile chosen, and the values *bl_min* and *bl_max*.

3.2 Quality-based boost strategy

The above buffer-based strategy can be extended to take into account the video quality being retrieved: in addition to the bandwidth boost triggered by the buffer length, additional bandwidth boost might be triggered because of the video quality. We define three parameters: *min_quality*, *max_quality* and *nb_quality_hit*. *nb_quality_hit* defines the number of consecutive segments in a quality below or equal *min_quality* that the cache server can tolerate before asking for a bandwidth boost at a specified QoS profile. *nb_quality_hit* also defines the number of consecutive segment downloads that must have occurred in a quality of at least *max_quality* before releasing the bandwidth boost.

4 IMPLEMENTATION AND EVALUATION

We implemented a cache server that is able to interpret CMCD information provided by a video player and that implements the two bandwidth boost strategies discussed in this paper.

We also implemented the LF CAMARA QoD API that is called by our caching server according to the used bandwidth boost strategy. The QoD API allows us to abstract the underlying network and mechanisms that apply the requested QoS profile. We tested three different QoS profiles that can be requested via the QoD API as described in Table 1. We implemented two approaches for the actual

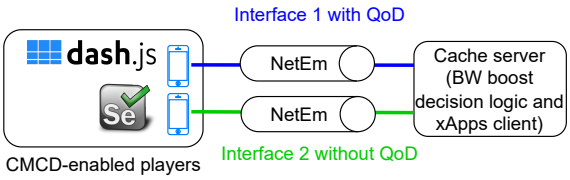

Figure 3: Emulated mobile network

underlying network and the application and enforcement of the requested QoS profile within the network: (i) *tc*-based network emulation as described in subsection 4.1; and (ii) an O-RAN compliant 5G RAN as described in subsection 4.2.

During our evaluation, we measured and report different metrics that reflect the QoE as experienced by the end-user and the budget spent by performing bandwidth boosts:

- *Stall duration:* Cumulative duration of video freezes because the player buffer is empty.
- *Average bitrate:* Average video bitrate retrieved by the video player.
- *Number of boosts:* Number of times the cache server requested a bandwidth boost.
- *Duration of boosts:* Cumulative duration in seconds of bandwidth boost.

For each reported metric we performed 10 test runs. Each run lasts for 30 minutes and average results are reported.

We relied on the dash.js [8] player (version 4.7.3), which provides buffer length and bitrate information via CMCD. We used the default dash.js' ABR strategy called "Dynamic". We customized the player to report QoE video metrics at the end of the streaming session. The tests were performed automatically using Python Selenium [1]. We used a live video that we produced by looping an approximately 12 minutes long video file "Tears of Steel" encoded with a 5-layer video ladder as described in Table 1. The segment duration is 2 seconds.

4.1 Emulation

We used *tc* NetEm to emulate network conditions. To mimic realistic network conditions, we based our evaluation on the network bandwidth traces *Cascade*, *Intra-cascade* and *Spike* provided with the ACM MMSys 2020 Twitch Grand Challenge [21]. Similarly to [6], we adapted the bandwidth values of the traces resulting in a bandwidth pattern described in Table 2. *tc* applies the bandwidths and durations specified in the traces with a RTT of 50ms. Episodes with low bandwidth (1Mbps) emulate the difficult wireless conditions. While a little above the lowest video encoding bitrate of 800kbps, such bandwidth is sufficient to slowly drain the video player buffer (due to the additional audio of 128kbps and manifest requests combined plus a small variability in video encoding rate).

The cache server exposes two interfaces to the players (Figure 3). We applied the same bandwidth patterns on both interfaces. When the cache server requests a QoS profile via the network API, we overrode the bandwidth of the target interface with the bandwidth corresponding to the QoS profile. When the QoD request is deleted, *tc* falls back to the bandwidth set in the network bandwidth pattern.

Table 2: Repeated network pattern used in the evaluation derived from [21] with a RTT of 50ms.

Pattern	Bandwidth pattern (Mbps)	Duration (s)
Cascade	6, 1	30
Intra-cascade	6, 3, 1, 3	15
Spike	6, 1, 5	10

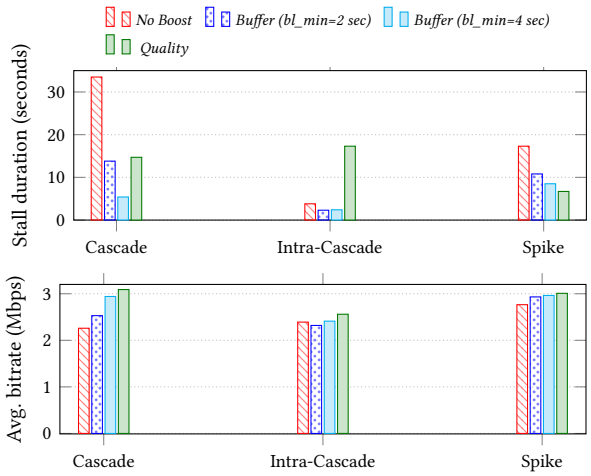

Figure 4: Average performance metrics of the buffer and quality-based bandwidth boost strategies. The buffer-based strategy is defined as a function of the minimum buffer length threshold *bl_min* (*bl_max*=8 seconds). The quality strategy uses the parameters *min_quality*=1.5Mbps, *max_quality*=2.5Mbps, *nb_quality_hit*=3. For all strategies the requested QoS profile is *Medium* (4Mbps).

This setup allows us to run tests with and without bandwidth boost in parallel, while ensuring that both video players exhibit the same network conditions but in separate environments.

Figure 4 reports the stall duration and average bitrate without bandwidth boost and with buffer-based and quality-based bandwidth boost strategies. *We observe that all buffer-based boost strategies reduce the stall duration.* Without any bandwidth boost the video player experience stalls for about 33 seconds, 4 seconds and 17 seconds for the patterns *Cascade*, *Intra-Cascade* and *Spike* respectively. The buffer-based bandwidth boost strategies allow to reduce the stall duration down to about 6 seconds, 2 seconds and 8 seconds respectively. We also observe that increasing *bl_min* in the buffer-based bandwidth boost decreases the stall duration. This is expected, as the boost is called earlier before the buffer is at risk of being completely empty. In terms of average bitrate, the bandwidth boost tends to improve the average bitrate, but the magnitude of improvements largely depend on the traffic pattern.

The quality-based strategy improves the bitrate the most; this was expected as it's objective was to avoid video bitrate degradation. However, the quality-based strategy is less efficient in reducing the number of stalls than the buffer-based strategy. Worse, in some scenarios, the stall duration increases significantly. We suspect that

Table 3: Budget spent for different strategies and scenarios.

Buffer-based boost (medium, bl_max=8)						
	Cascade		*Intra-Cascade*		*Spike*	
bl_min (sec)	2	4	2	4	2	4
Nb. boosts	26	28	3	14	16	52
Duration boost (seconds)	228	214	13	66	92	388

Quality-based boost (medium, min_quality=1.5Mbps, max_quality=2.5Mbps, nb_quality_hit=3)			
	Cascade	*Intra-Cascade*	*Spike*
Nb. boosts	31	38	62
Duration boost (sec)	814	1024	753

the quality-based boost made the ABR adaptation algorithm switch to high encoding bitrates; at the time of release of the bandwidth boost the player experiences a sudden and strong degradation of the network conditions that the ABR algorithm cannot compensate for anymore. This highlights the importance of carefully designing the bandwidth boost strategy. Further work is required to propose an efficient quality-based bandwidth boost strategy.

The budget associated with each of these strategies in our experiments is shown in Table 3. We can notice that the number of boosts increases while increasing the *bl_min* threshold. The bandwidth boost is triggered earlier and more often, giving the player less time to drain its buffer. The quality strategy is more expensive than the buffer-based strategy, especially when considering the duration of boosts. In light of the little benefits in average bitrate with mitigated results on the stall duration, we recommend to rely on the more simple yet effective buffer-based strategies.

The effect of changing the *bl_max* parameter and the requested QoS profile is shown in Figure 5. First we can observe - as already shown previously - that the players that benefit from a bandwidth boost have smaller stall duration and a higher average bitrate.

The highest QoS profile does not provide the best results. Stall duration and average bitrate are worse than boost strategies relying on lower QoS profiles. This may be explained by the ABR adaptation algorithm of the video player. As the video player is measuring a high available bandwidth it switches to higher video qualities in the video layer. When the bandwidth boost is released, the player will observe a sudden drop in available bandwidth and may not react immediately (i.e., change video quality), which results in draining the buffer more rapidly. This behavior highlights the difficulties when having two different control loops (bitrate adaptation by the video client and bandwidth boost by the server), especially in varying network conditions. It's a subject for further work.

Increasing the *bl_max* buffer duration threshold upon which the bandwidth boost is released has similar effects than increasing the requested QoS profile. A high *bl_max* can increase the stall duration and decrease the average video bitrate compared to lower ones. Similarly to the previous analysis, we suspect that this is due to the interaction with the ABR adaptation algorithm of the video player that decides to switch to higher video quality which has an adverse effect when the boost is released.

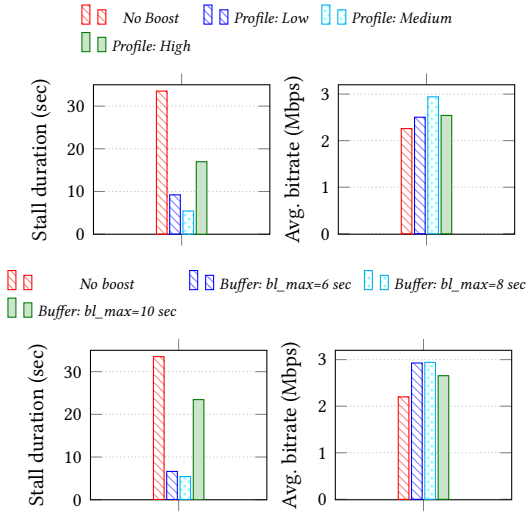

Figure 5: Average performance metrics of buffer-based bandwidth boost strategies with the Cascade pattern (top) as a function of the QoS profile requested with *bl_max*=8 and (bottom) as a function of the maximum buffer length threshold *bl_max* with a medium QoS profile (*bl_min*=4 seconds).

4.2 xApp-enabled 5G RAN

Our experiments were conducted on top of an OpenAirInterface 5G RAN [12] and 5G Core network [11]. The cache server is deployed within an edge cluster of the 5G network. The cache server has access to an LF CAMARA API which interacts with an xApp that controls the 5G RAN.

We implemented an xApp capable of handling bandwidth boost requests using the FlexRIC [17] framework. We apply the requested QoS profile by reusing the existing bearer as described in section 2. More specifically, the xApp overrides the decisions of the scheduler of the radio base station (gNB) by assigning a number of Physical Resource Blocks (PRBs) that corresponds to the requested bandwidth. Each PRB has multiple Resource Elements (RE) which is the smallest radio resource unit. The Modulation Coding Scheme (MCS) defines how many bits are carried by a single RE over the air. MCS varies depending on the network condition. When the MCS changes, the xApp dynamically adapts the number of assigned PRBs in order to meet the requested bandwidth.

When the bandwidth boost is deactivated, the mobile terminal gets a fixed number of radio resource blocks, set to 5 PRBs in our experiments. A change in the MCS will therefore directly impact the available bandwidth.

We ran the following experiments. We used a mobile phone that streams the same live video described before. To generate varying radio conditions we varied the MCS of the radio according to the Twitch traffic pattern as shown in Table 4. This also allows us to run repeatable experiments. We ran this experiment several times with and without the proposed bandwidth boost. We only focus on buffer-based boost strategies, as we showed previously the limitations of quality-based strategies. How the used MCS values map to available

Table 4: Repeated MCS radio network pattern used in the evaluation derived from [21].

Pattern	MCS pattern	Duration of one MCS value (s)
Cascade 8	27, 8	30
Cascade 9	27, 9	30
Intra-Cascade	27, 16, 8, 16	15
Spike	27, 8, 16	10

Table 5: Theoretical mapping of MCS values to radio bandwidth using 5 PRBs [2].

MCS	max throughput	MCS	max throughput
27	4.7Mbps	9	1.1Mbps
16	2.27Mbps	8	970kbps
12	1.45Mbps	6	700kbps

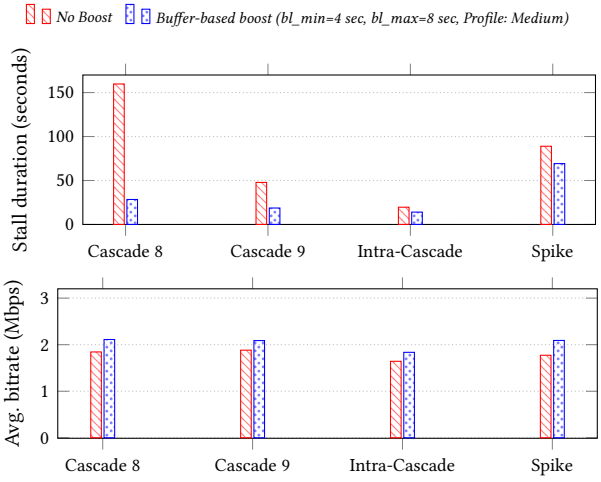

Figure 6: Average performance metrics of the buffer-based bandwidth boost strategies on 5G O-RAN testbed. The target bitrate of the xApp when the boost is enabled is 4Mbps.

bandwidth using the default number of radio resource blocks (when the boost is not activated) is shown in Table 5.

Figure 6 reports the stall duration and average bitrate with and without buffer-based bandwidth boost on our 5G network. We observe that without bandwidth boost the 30 minutes video playout shows significant stalls with up to 158 seconds of stalls with the network pattern *Cascade 8* and up to 50 seconds with *Cascade 9*. When the bandwidth boost is activated the stall duration is largely decreased to 28 seconds and 18 seconds respectively. For *Intra-cascade* and *Spike*, the benefit of the bandwidth boost is smaller: the stall duration decreases from 89 seconds to 69 seconds for *Spike* and from 20 seconds to 14 seconds for *Intra-cascade*. The effect on the average bitrate is less visible; the average bitrate increases about 0.2 Mbps in all network patterns. Overall, the experiments on the actual 5G network are in-line the ones using network emulation and show the clear benefit of using a buffer-based boost strategy.

5 RELATED WORK

The use of CMCD to improve the performance of adaptive video streaming has been discussed in [6]. More specifically, the authors implement a buffer-aware bandwidth allocation algorithm on the video streaming server that exploits buffer information provided by the video players via CMCD. The server implements a buffer-length-to-rate mapping function, i.e. the server calculates a rate that is allocated to a given video player as a function of the current buffer length and the minimum and maximum bandwidth supported by the player. The authors show that they are able to significantly reduce rebuffering events across different scenarios. Lim et al.[10] extend this work by relying on Common Media Server Data (CMSD). The server adapts its scheduling based on the buffer-level signaled via CMCD by serving video players being at risk of a buffer underrun first. At the same time, the server signals via CMSD to the delayed video players how long its requests have been delayed, such that the players can adapt their rate calculation accordingly. Our work also exploits CMCD buffer length information in order to detect and act upon possible bandwidth bottlenecks in the 5G cellular radio access network. Our approach and the two above related work are not exclusive to each other and it may even be beneficial to combine them, i.e. to design a cache server that implements a rate allocation buffer-aware rate allocation algorithm while at the same time boosting the radio bandwidth when needed.

Use cases for QoD APIs have been demonstrated at several showcases by different operators and partnering application providers [7, 13, 18, 19]. The approach adopted in these demonstrations is to continuously allocate a request quality (latency and bandwidth) to an application. E.g. the bandwidth would be guaranteed from the beginning until the end during the usage of the requesting application such as a game streaming session. In contrast, in our approach the bandwidth requests are only temporary and are released as soon as the buffer length is sufficient. This avoids the caveats of constantly reserving radio resources, therefore freeing them for other end-users and applications.

6 CONCLUSIONS

In this paper, we presented a mechanism to improve the QoE of video streaming on top of a cellular network with difficult network conditions. We relied on recent network APIs - such as the QoD LF CAMARA API and the O-RAN xApps - exposed by cellular network operators. These network APIs allow an application such as a video streaming CDN to ask for additional bandwidth resources when needed. We leveraged CMCD to provide buffer and bitrate information to the cache server. This allows the cache server to ask for a bandwidth boost at moments where the player encounters degraded download conditions. We implemented a cache server that interfaces with the LF CAMARA QoD API and evaluated the benefits on top of an emulation testbed and a 5G RAN controlled via an xApps. We show that simple bandwidth boost strategies allows us to improve the overall QoE. We also showed that a too aggressive bandwidth boost strategy may have adverse effects on the ABR adaptation algorithm of the video player, which opens perspectives to further study and design bandwidth boost algorithms.

ACKNOWLEDGEMENT

This work is funded by the french government within the framework "France 2030".

REFERENCES

[1] 2023. *Selenium with Python.* https://selenium-python.readthedocs.io/
[2] 3GPP. 2023. 3GPP TS 38.214 Release 17. TSG RAN; NR; Physical Layer Procedures For Data.
[3] The O-Ran Alliance. 2023. *O-RAN ALLIANCE Specifications.* https://www.o-ran.org/specifications
[4] The O-RAN ALLIANCE. October 2023. O-RAN Architecture Description R003-v10.00.
[5] Consumer Technology Association et al. 2020. Web Application Video Ecosystem-Common Media Client Data. CTA-5004. *Retrieved June* 7 (2020), 2021.
[6] Abdelhak Bentaleb, May Lim, Mehmet N Akcay, Ali C Begen, and Roger Zimmermann. 2021. Common media client data (cmcd) initial findings. In *Proceedings of the 31st ACM Workshop on Network and Operating Systems Support for Digital Audio and Video.* 25–33.
[7] Blacknut. 2023. *Operators are opening up 5G networks to application developers to drive innovation.* https://www.blacknut.biz/press-release/operators-are-opening-up-5-g-networks-to-application-developers-to-drive-innovation
[8] Dash Industry Forum. 2023. *Dash.js source code.* https://github.com/Dash-Industry-Forum/dash.js
[9] The Linux Foundation. 2023. *Camara - The Telco Global API Alliance.* https://camaraproject.org/
[10] May Lim, Mehmet N Akcay, Abdelhak Bentaleb, Ali C Begen, and Roger Zimmermann. 2022. The benefits of server hinting when DASHing or HLSing. In *Proceedings of the 1st Mile-High Video Conference.* 52–55.
[11] OpenAirInterface. 2023. *5G CORE NETWORK.* https://openairinterface.org/oai-5g-core-network-project/
[12] OpenAirInterface. 2023. *OpenAirInterface 5G Radio Access Network Project.* https://openairinterface.org/oai-5g-ran-project/
[13] Orange. 2023. *CAMARA - Quality on Demand.* https://developer.orange.com/apis/camara-quality-on-demand
[14] Jose Ordonez-Lucena and Felix Dsouza. 2022. Pathways towards network-as-a-service: the CAMARA project. In *Proceedings of the ACM SIGCOMM Workshop on Network-Application Integration.* 53–59.
[15] Michele Polese, Leonardo Bonati, Salvatore D'Oro, Stefano Basagni, and Tommaso Melodia. 2022. Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges. *CoRR* abs/2202.01032 (2022). arXiv:2202.01032 https://arxiv.org/abs/2202.01032
[16] The Linux Foundation CAMARA project. 2023. *Quality on Demand.* https://camaraproject.org/quality-on-demand/
[17] Robert Schmidt, Mikel Irazabal, and Navid Nikaein. 2021. FlexRIC: an SDK for next-generation SD-RANs. In *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies.* 411–425.
[18] Telefonica. 2023. *QoD Mobile API.* https://opengateway.telefonica.com/en/apis/qod-mobile
[19] Deutsche Telekom. 2023. *Telekom commercially launches network APIs.* https://www.telekom.com/en/media/media-information/archive/telekom-commercially-launches-network-apis-1049276
[20] Tuan Tran, Christoph Neumann, and Guillaume Bichot. 2023. Elastic Video Content Delivery Networks at the Edge. In *Proceedings of the 2nd Mile-High Video Conference* (Denver, CO, USA) *(MHV '23).* Association for Computing Machinery, New York, NY, USA, 91–96. https://doi.org/10.1145/3588444.3591010
[21] Twitch. 2023. *Twitch's ACM MMSys 2020 Grand Challenge.* https://github.com/twitchtv/acm-mmsys-2020-grand-challenge
[22] Vonage. 2023. *Vonage QoD.* https://developer.vonage.com/en/api/qod