Cross that boundary: Investigating the feasibility of cross-layer information sharing for enhancing ABR decision logic over QUIC

Joris Herbots
Hasselt University – tUL – EDM
Diepenbeek, Belgium
joris.herbots@uhasselt.be

Arno Verstraete
Hasselt University – tUL – Flanders
Make – EDM
Diepenbeek, Belgium
arno.verstraete@uhasselt.be

Maarten Wijnants
Hasselt University – tUL – Flanders
Make – EDM
Diepenbeek, Belgium
maarten.wijnants@uhasselt.be

Peter Quax
Hasselt University – tUL – Flanders
Make - EDM
Diepenbeek, Belgium
peter.quax@uhasselt.be

Wim Lamotte
Hasselt University – tUL - EDM
Diepenbeek, Belgium
wim.lamotte@uhasselt.be

ABSTRACT

With HTTP Adaptive Streaming (HAS), client-side Adaptive Bitrate (ABR) algorithms drive the (quality-variant) scheduling and downloading of media segments. These ABR algorithms are implemented in the application layer and can therefore base their logic only on relatively coarse and/or inaccurate application-layer metrics. The recently standardized QUIC transport protocol has many userspace implementations, which paves the way for cross-layer optimizations by exposing transport-layer metrics to application-layer algorithms. In this paper, we investigate whether the availability of fine-grained transport-level throughput metrics can positively impact the operation of ABR algorithms and hence the Quality of Experience (QoE) of HAS users in Video on Demand (VoD) settings. Our results show that QUIC-level throughput data can indeed aid ABR algorithms to more accurately predict playout buffer underruns, which in turn allows the ABR logic to take reactive measures in a timely fashion such that playback stalls can be avoided under challenging network conditions. Overall, our work presents a step towards improving ABR operation via cross-layer data exchange.

CCS CONCEPTS

• **Networks** → **Transport protocols**; **Application layer protocols**; • **Information systems** → **Multimedia streaming**.

KEYWORDS

MPEG-DASH, HAS, QoE, playback stall prediction

ACM Reference Format:
Joris Herbots, Arno Verstraete, Maarten Wijnants, Peter Quax, and Wim Lamotte. 2023. Cross that boundary: Investigating the feasibility of cross-layer information sharing for enhancing ABR decision logic over QUIC. In *The 33rd edition of the Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV '23), June 7–10, 2023, Vancouver, BC, Canada.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3592473.3592563

NOSSDAV '23, June 7–10, 2023, Vancouver, BC, Canada
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *The 33rd edition of the Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV '23), June 7–10, 2023, Vancouver, BC, Canada*, https://doi.org/10.1145/3592473.3592563.

1 INTRODUCTION

HTTP Adaptive Streaming (HAS), with its standardized MPEG-DASH specification, is the state-of-the-practice solution for on-demand video streaming across the Internet. Due to its adoption by major video streaming platforms like YouTube and Netflix, HAS data dominates the Internet traffic mix nowadays [27]. In the HAS paradigm, video content is temporally segmented in chunks that are made available in multiple bitrates, which are differentiated by their resolution, framerate, picture quality etc. Client-side Adaptive BitRate (ABR) algorithms, which operate integrally in the application layer of the OSI reference model, dynamically steer the HAS streaming session by deciding which media segments to fetch and in what quality to do so. Measurements or estimations of network conditions typically play a prominent role in ABR implementations. As an example, if segment bitrates drastically exceed network capacity for prolonged periods of time, the client-side playback buffer will drain. This will yield playback stalls, which are extremely detrimental to the Quality of Experience (QoE) [19].

Broadly speaking, ABR algorithms can be categorized in heuristics-based and (machine) learning-based approaches [6]. The former category is the focus of this paper and can be further subdivided into throughput-based, buffer-based and hybrid schemes. Throughput- and buffer-based schemes base their operation on network throughput measurements, and the state of the client-side playback buffer, respectively; hybrid ABR algorithms mix throughput- and buffer-based mechanisms. An important challenge that all types of heuristics-based ABR algorithms face is that they have to ground their segment quality selection decisions on a limited set of data points. In effect, since ABR algorithms are implemented in the application layer, their network throughput estimations (and indications of playback buffer changes) use integral HAS media segments (typically containing multiple seconds of video) as a data unit. Such application-layer metrics are coarse-grained and highly periodic (i.e., with relatively large temporal gaps between consecutive data points). Especially for network throughput estimations, this information is available in a more granular and timely fashion in lower layers of the network stack. For instance, at the transport layer, network throughput can

be measured at the TCP segment or UDP datagram level, which is much more granular compared to application-layer media segments (one HAS segment easily spans over 100 transport-layer packets).

This work investigates the feasibility of improving HAS ABR algorithms in Video on Demand (VoD) settings by exploiting cross-layer information exchange between the OSI transport and application layers. In more detail, our contribution is three-fold. Firstly, we contribute a proof-of-concept cross-layer MPEG-DASH client, implemented in `quic-go` and `goDASH`, where fine-grained throughput measurements are exposed from the transport layer (i.e., QUIC) to the MPEG-DASH client. Secondly, we exploit the resulting transport-layer data to integrate stall prediction in the BBA-2 ABR algorithm. Finally, testbed-based experimental results show the ability of the cross-layer stall prediction algorithm to prevent video playback stalls when working with small HAS playback buffers under varying throughput-constrained network conditions.

2 RELATED WORK

2.1 Cross-layer network optimization

Fu et al. [10] provide a survey of cross-layer designs in wireless networks, hereby discussing various architectures and methods for exchanging information between network layers. According to their proposed taxonomy, our work would be classified as a *non-manager cross-layer method* (i.e., a direct data exchange between two adjacent layers). Recent work by Xixi et al. [29] proposes a cross-layer joint scheduling scheme between the physical and data link OSI layers to improve resource scheduling for infrastructure-less device-to-device communication in cellular networks. The cross-layer architecture proposed by Piri et al. [22] allows for real-time (i.e., RTP-like) adaptive streaming over heterogeneous wireless networks using the IEEE 802.21 Media Independent Handover (MIH) Function. Their cross-layer approach, called Triggering Engine, collects and exchanges information across multiple OSI layers and performs MIH with an access point. An application controller connected to this cross-layer signaling system can subsequently adjust video transmission parameters based on the transmission channel characteristics. Later work by Piri et al. [23] uses the Triggering Engine to exchange cross-layer network information in an end-to-end fashion between client and server for Scalable Video Coding (SVC), again in a real-time (i.e., non-HAS) video conferencing setting.

As the above works illustrate, the majority of the cross-layer prior art focuses on low(er) OSI layers, whereas our work is concerned with the OSI transport- and application layer. In addition, the idea of cross-layer HAS optimization is relatively underexplored, especially in the VoD context that this paper focuses on.

2.2 Exploiting QUIC features for HAS

Compared to TCP, QUIC introduces multiple new features in the transport layer that can be utilized to improve adaptive streaming protocols. Two relevant examples are VOXEL and Days of Future Past (DoFP). VOXEL [20] leverages the unreliable extension of QUIC [21] to create a partially reliable streaming architecture. An a priori calculation decides what video frames can be streamed unreliably (i.e., without recovery for lost packets) while still delivering a high QoE and lowering the need for rebuffering. DoFP [16] uses QUIC's multiplexing feature to improve the QoE by redownloading already buffered (but not yet played) HAS segments in a higher quality. Their work, similar to ours, makes use of the abort mechanism of QUIC-HTTP/3 to halt scheduled HAS segments that will not arrive in time for playback (in the DoFP case: higher-quality redownloads of already buffered segments).

2.3 ON-OFF behavior

The ON-OFF behavior is a well-discussed HAS topic [5, 12], where it is known to cause poor bandwidth utilization and fairness issues for TCP and QUIC [7]. ON-OFF behavior can also be used to explain the sequential nature of HAS. ABR algorithms fetch segments during the ON periods and then execute their ABR scheduling logic during the OFF period. The authors of BOLA [28] proposed a variation of their ABR algorithm called BOLA-FINITE that uses a segment abandonment strategy to prevent stalls. BOLA-FINITE, similar to our work, proposes to monitor during the ON period, instead of executing ABR logic exclusively during the OFF period. Contrary to our work, BOLA-FINITE solely employs application layer metrics.

3 EXPERIMENTAL METHODOLOGY

To investigate the feasibility of sharing transport layer metrics with ABR algorithms, we needed to create a setup where such an information exchange could occur. Traditional OSI transport layer protocols, most notably TCP, are implemented in kernel space. While this yields performance benefits, it is unfortunately circuitous to apply changes or extract metrics from kernel-space implementations. As such, in this paper, we look at QUIC and HTTP/3, recent additions to the transport layer and HTTP protocol collections standardized by the IETF [8, 14]. Unlike TCP, all current QUIC implementations are available as user-space libraries [1], allowing for extracting transport layer metrics with less overhead. Also, most available QUIC implementations support the structured `qlog` format [18] – a unified way of logging QUIC and HTTP/3 events. This includes, but is not limited to, metrics we are interested in, such as congestion control metrics, flow control metrics and per-packet information.

3.1 Client Implementation

Ideally, our setup would use a reference player like the `dash.js` web implementation [2]. However, extracting transport layer metrics from the browser's QUIC implementation is, similarly to kernel space implementations, a circuitous route. Instead, we have forked the open-source `goDASH` implementation [25], a headless framework for streaming MPEG-DASH video content. `goDASH` is implemented in the Go programming language and supports QUIC-HTTP/3 video streaming by utilizing the open-source `quic-go` library [3] that implements the QUIC protocol, also written in the Go programming language. `quic-go` adopts the `qlog` logging format.

Because both projects are written in Go, we implemented the Go message channel concept between the `quic-go` and the `goDASH` GoRoutines (i.e., a message pipe between two threads) to exchange `quic-go`'s `qlog` metrics with `goDASH`. The message channel effectively creates a unidirectional cross-layer information exchange from the transport layer to the application layer. Additionally, we changed `goDASH`'s handling of HTTP/3 requests to reuse the QUIC connection instead of creating a new QUIC-HTTP/3 connection

for each MPEG-DASH request. This more closely resembles the desired behavior for QUIC connections.

goDASH comes equipped with a range of MPEG-DASH ABR algorithms, including sophisticated buffer-based algorithms such as BBA-1 by Huang et al. [13]. Unfortunately, BBA-2 – which provides a better QoE during the startup phase by being more aggressive than BBA-1 [13] – is unavailable. Additionally, the BBA-1 implementation of goDASH uses HTTP HEAD requests to determine the individual segment sizes. During our testing, this approach yielded poor results. Each HTTP HEAD request incurs one extra RTT, which amounts to lower bitrate choices, especially in high-latency situations. Considering the shortcomings above, we decided to implement the BBA-2 algorithm for our proof-of-concept, in which the segment sizes are known at the start by specifying them in the MPEG-DASH manifest. BBA-2 is a hybrid heuristics-based algorithm that uses rate-based estimations during the startup phase and transitions towards buffer-based logic when the buffer-based logic deems it safe to schedule higher quality segments. Similar to other ABR algorithms, BBA-2's logic executes at discrete moments right after finishing the download of the last scheduled segment and before fetching the next segment. As a result, BBA-2, similarly to other ABR algorithms, has no mechanism to detect the changes in prevailing network conditions in between MPEG-DASH segment scheduling. We will augment BBA-2's algorithm to monitor the stage of the network by inspecting the currently scheduled segment's more fine-grained QUIC packet flow. In the event the network deteriorates in such a way that stalling is imminent, a readjustment procedure will be started. This entails aborting the requested segment and resetting BBA-2's logic into its rate-based starting mode.

```
1  Function PredictStall(packet_queue,
     representation_bitrate, segment_duration, segment_size,
     buffer_level, bba2_config):
2      sum_bits = packet_queue.sum_bits()
3      window_time = time() - packet_queue.first()
4      outstanding_bits = segment_size - sum_bits
5      window_bitrate = sum_bits / window_time
6      if window_time ≤ 0 then
7          return false
8      end if
9      required_bits = segment_size × minimum_fraction
10     if required_bits ≤ sum_bits < segment_size then
11         estimated_req_time = outstanding_bits /
             window_bitrate
12         estimated_req_time_lowest_quality =
             bba2_config.lowest_representation /
             window_bitrate
13         if buffer_level ≤ bba2_config.lower_reservoir then
14             return estimated_req_time > buffer_level &&
                 estimated_req_time >
                 estimated_req_time_lowest_quality
15         end if
16     end if
17     return false
18 end
```

Algorithm 1: BBA2-CL stall algorithm pseudocode. All sizes are expressed in bits, and the durations in milliseconds.

3.2 Stall Prediction

Algorithm 1 showcases our approach to stall prediction using fine-grained throughput measurements acquired from the transport layer with our cross-layer message channel. A benefit of using BBA-2 is that its algorithm requires knowledge about each segment's size to combat oscillations in the advertised bitrates of the MPEG-DASH manifest introduced by Variable BitRate (VBR) encoding. This information allows us to make accurate stall predictions (line 4). Our algorithm only abandons a segment when three conditions are met. ❶ Firstly, during the segment download window, we require at least 10% of the total segment size to be downloaded (represented by `minimum_fraction` on line 9) as this produces stabler decisions. The reasoning is that the instantaneous throughput measurements of individual packets tend to fluctuate a lot, especially at the start of a download. ❷ Secondly, we only trigger our behavior when BBA-2 is at or below its lower reservoir to avoid false positives (line 13). The lower reservoir is the amount of buffer BBA-2 itself considers an unsafe territory, where network variations might cause it to choke. ❸ Finally, if the estimated download time of the currently scheduled segment exceeds the amount of buffer we have left, and the algorithm estimates that downloading the same segment in the lowest available quality is possible before the buffer runs out, it will give the go-ahead for abandonment (line 14). The client then sends a request cancellation frame, prompting the server to abruptly terminate the HTTP/3 stream carrying our initial segment request, as per the HTTP/3 specification. A new request will subsequently be made for the same segment but in the lowest available representation. Additionally, an abort resets the BBA-2 algorithm to regain its lower reservoir and restart its algorithm at the lowest available quality representation. A successful readjustment occurs if the observed throughput does not drop below the required bitrate for the lowest representation. We shall refer to this enhancement to BBA-2 as BBA2-CL (BBA2 Cross-Layer) from now on.

Furthermore, we implemented an extension to BBA2-CL, called BBA2-CLDouble, that checks for a specific edge case as a result of BBA-2's stepwise quality transitions. In cases where the playout buffer is heading towards a stall, but the current segment will finish before the buffer crosses the lower reservoir threshold, BBA-CL will not trigger. This causes the next segment to be scheduled with a dangerously low buffer, too low for safe readjustment by aborting. BBA2-CLDouble will, if condition ❸ is unmet, branch its logic to consider if the current segment and the next segment – downshifted by one quality level – can be safely fetched without stalling. If this branching logic predicts stalling, the same abandonment logic of BBA2-CL will occur.

4 EXPERIMENTAL SETUP

In order to evaluate the impact of BBA2-CL and BBA2-CLDouble, we set up an emulated network scenario using Linux TC with the `netem` and `tbf` queueing disciplines. The orchestration of all experiments was done with the Vegvisir [11] QUIC-HTTP/3 testing

Software	
QUIC HTTP/3 Servers	`ngtcp2`, `quic-go`, `aioquic`
QUIC HTTP/3 Client	`quic-go`
Network Software	`Linux TC` (`netem` and `tbf`)
Settings	
ABR algorithms	BBA-2, BBA2-CL, BBA2-CLDouble
BBA-2 max buffer	60 seconds
BBA-2 lower reservoir	10% of maximum buffer
Congestion control	New Reno (`quic-go`, `aioquic`), Cubic (`ngtcp2`)
`netem` properties	40 ms round trip time, 0% loss
tbf rate	10s @ 2Mbps, 30s @ 100kbps, 10s @ 1Mbps, 30s @ 100kbps, 20s @ 2Mbps
tbf parameters	buffer 5k limit 10k

Table 1: Summary of used software, settings, video datasets.

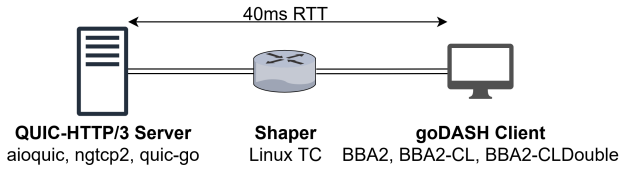

Figure 1: Experimental test-bed setup.

framework. Table 1 provides a summary of the software and settings used, while Figure 1 provides an overview of the network setup. The server and client assume their prototypical roles, connected via a network router called the shaper, which applied our network emulation. The client, shaper and server were all represented by docker containers interconnected via two docker-compose networks routed via the shaper.

The work by Marx et al. [17] shows that different QUIC implementations can exhibit different performance behavior – depending on the context – due to underlying differences. In our setup, we therefore tested our client against three unmodified QUIC-HTTP/3 server implementations (to investigate inter-implementation performance differences): `aioquic`, `ngtcp2` and `quic-go`. As recommended by the `quic-go` implementation, the UDP receive buffer was increased to 2.5MB [4].

The server hosts three MPEG-DASH audiovisual datasets [15]: Big Buck Bunny (BBB), Elephants Dream (ED) and Of Forrest and Men (OFM). For each dataset, a temporal segmentation of 2-seconds, 4-seconds and 6-seconds was provided, resulting in 9 different manifests being tested. Each dataset contained 20 representations targeting 240p, 360p, 720p and 1080p with an average bitrate ladder ranging between 45kbps and 4.2Mbps.

Based on the bitrate ladder adopted by the utilized MPEG-DASH datasets, we constructed an artificial network scenario (see the dotted purple line in Figure 2), enforced by the shaper, that compels an ABR algorithm toward a worst-case scenario. The shaper starts the connection at 2Mbps, allowing our client to experience "sufficient" throughput in which the ABR algorithm can learn the network conditions and scale its target quality accordingly. The shaper then drops the experienced throughput to 100kbps at the 10 seconds mark and maintains this setting for 30 seconds. Since most HAS clients have no direct control over the transport layer, ABR algorithms cannot cancel an ongoing request[1] and will, most likely, be stuck fetching a segment at a bitrate that exceeds the current throughput. Depending on how much buffer buildup occurred during the first 10 seconds, a potential stall can occur during this constrained throughput window. Next, the shaper will raise the experienced throughput to 1Mbps for 10 seconds before, once again, introducing a drop to 100kbps for 30 seconds. Finally, the shaper increases the experienced throughput to 2Mbps for the remaining 20 seconds of the test. Thus, a single test totals 100 seconds, containing two intermittent drops to 100kbps that last for a total of 60 seconds. The shaper was additionally configured with a static 40ms Round Trip Time (RTT). The queues employed by the shaper were sized to the bandwidth-delay product.

Our experimental setup entails 81 simulations (3 QUIC-HTTP/3 servers × 9 datasets × 3 ABR algorithms).

5 EXPERIMENTAL RESULTS

During our experiment, we collected the client's experienced buffer occupancy, quality switches in subsequent MPEG-DASH media segments, stall periods, and stall prediction timestamps (only for our enhanced ABR algorithms). Figure 2 shows the results obtained from the setup described in Section 3. The results show the interactions between the `goDASH` client and the `quic-go` server only, as no performance differences were witnessed between the three tested QUIC-HTTP/3 servers.

There is a total of six rows. The first two represent the *Big Buck Bunny (BBB)* dataset, the third and fourth rows represent the *Elephants Dream (ED)* dataset, and the last two rows represent the *Of Forrest and Men (OFM)* dataset. The first column represents 2-second segments, the middle column 4-second segments and the third column 6-second segments of each dataset. The first row of each dataset contains the results with the base implementation of BBA-2, while the second row shows the results of our BBA2-CL enhancement. The results obtained from simulations with BBA2-CLDouble did not provide additional insights compared to BBA2-CL. Thus, we omitted them from Figure 2 due to space constraints, except for a single result in Figure 3.

The results in Figure 2 (a)-(c), (g)-(i), and (m)-(o) show the effects of BBA-2 in our constrained network scenario. During the first 10 seconds, the buffer occupancy (solid blue line) rises as the BBA-2 algorithm fills its lower reservoir and switches from rate-based to buffer-based bitrate targetting. The buffer growth differences between the columns are attributed to the differences in MPEG-DASH temporal segmentation, similar bitrate ladders used for their encoding and BBA-2's stepwise quality transitions. A shorter temporal segmentation implies less data, and these will thus be retrieved faster than longer segments. This, in turn, allows BBA-2 to upscale its quality faster. A higher bitrate implies fetching more data and, thus, longer download times. We can see this phenomenon happening in all the results. The dashed green line targets higher qualities during the startup phase for shorter temporally segmented datasets, resulting in less buffer fill. During the two periods of deteriorated throughput – 10 to 40 seconds and 50 to 80 seconds – a drop in the

[1]Support for abandonment exists, for example in the JavaScript fetch API, but is rarely used by existing ABR implementations.

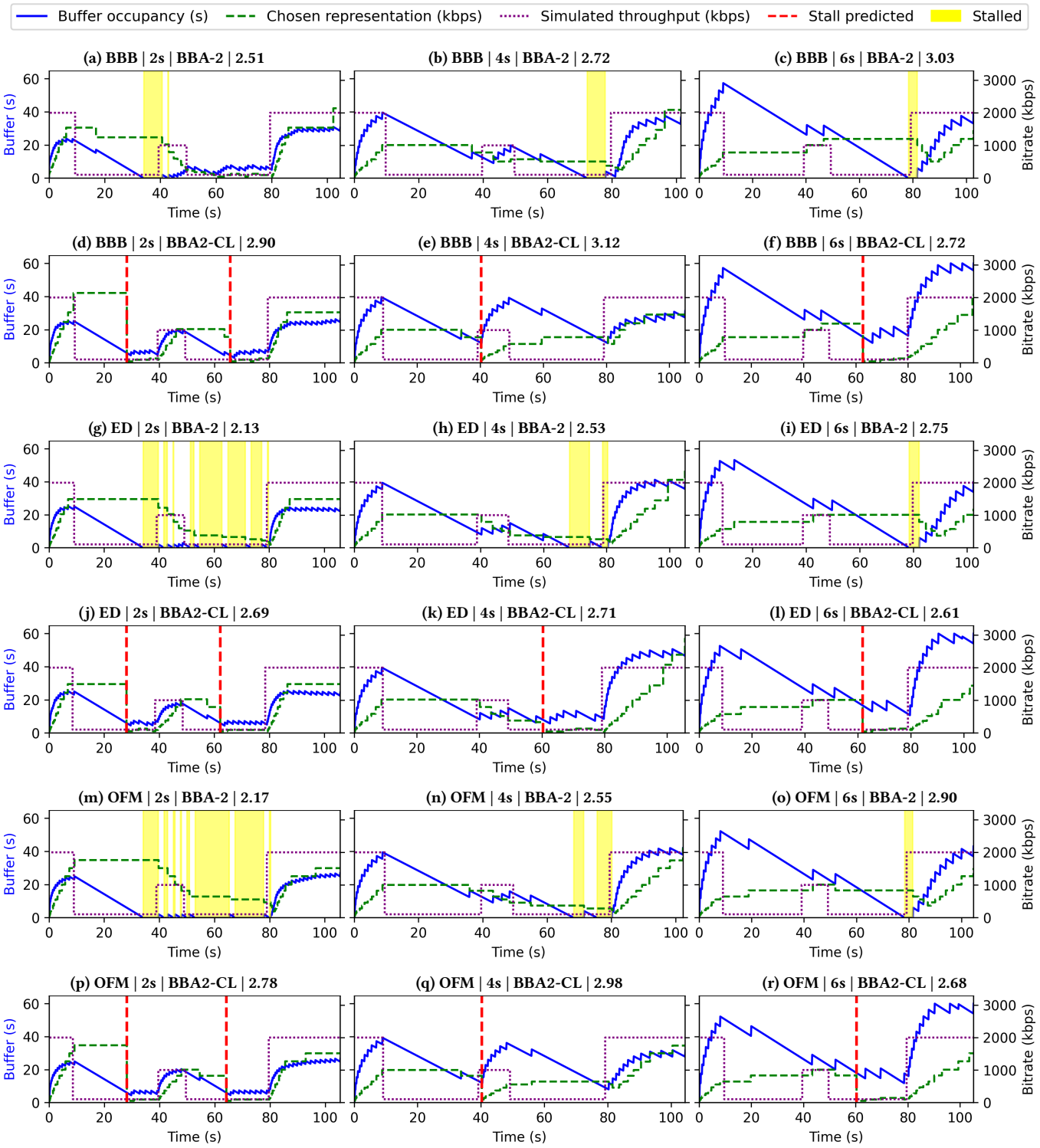

Figure 2: Client metrics collected from the experiment setup. Each graph has a subtitle in the form of "`dataset` | `segment size` | `ABR algorithm` | P.1203 MOS". The tested datasets are *Big Buck Bunny (BBB)*, *Elephants Dream (ED)* and *Of Forrest and Men (OFM)*. Solid blue lines represent the buffer occupancy in seconds. Dashed green lines represent the targeted MPEG-DASH quality bitrate in kbps, while dotted purple lines represent our artificial network scenario in kbps. BBA2-CL graphs (d)-(f), (j)-(l) and (p)-(r) additionally contain red dashed vertical lines indicating BBA2-CL's stall prediction. Vertical zones highlighted in yellow indicate stalls.

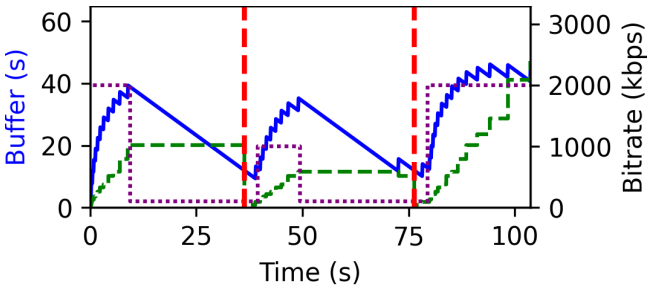

Figure 3: ED | 4s | BBA2-CLDouble | 2.63

buffer occupancy can be seen as the client tries to fetch an MPEG-DASH segment that is encoded at a bitrate (dashed green line) that the network cannot sustain during these periods (dotted purple line). As a result, the buffer drains completely in all nine simulations (yellow highlighted zones). Due to a lower initial buffer buildup, the 2-second datasets experience significantly more prolonged buffer underruns than the 4-second and 6-second datasets.

The results in Figure 2 (d)-(f), (j)-(l), and (p)-(r) show the effects of our stall prediction enhancement BBA2-CL. No stalling is observed in any of the tested conditions, as can be seen by the absence of the yellow highlighted zones and the buffer occupancy never reaching zero. The datasets with 2-second segments experience two stall predictions in both deteriorated throughput periods. The 4-second and 6-second segments experience only one stall prediction each; also, contrary to the 2-second results, these stall predictions only occur after the first deterioration period has already ended (i.e., after the 40-second mark). Results (e) and (q) experience a false-positive stall prediction at the beginning of the first throughput recovery period (i.e., between 40 and 50 seconds) due to a combination of passing the lower reservoir threshold and not having seen enough packets experience the new throughput window between 40 and 50 seconds, thus falsely triggering a stall prediction. The BBA2-CL results show that the application-layer ABR algorithm can benefit from the more fine-grained measurements from the transport layer, effectively predicting and preventing buffer underruns.

Finally, we also quantified the QoE impact of BBA2-CL and BBA2-CLDouble by calculating the ITU-T P.1203 Mean Opinion Score (MOS) in mode 0, which ranges between 1 and 5 [24, 26]. As can be seen in Figure 2, BBA2-CL consistently scored higher in all 2- and 4-second simulations compared to BBA2, but underachieved for all 6-second simulations. This latter result can be attributed to two factors. Firstly, in the 6-second simulations, the witnessed stall durations were much shorter compared to the 2- and 4-second cases; shorter stalls penalize the P.1203 score less than longer stalls. Secondly, aggressive MPEG-DASH quality switches (which happen when BBA2-CL's stall prediction triggers) have a more negative P.1203 score impact than stepwise quality transitions (cf. BBA-2 where two subsequent MPEG-DASH media segments may differ at most one quality level). BBA2-CLDouble on average produced slightly lower P.1203 scores compared to BBA2-CL. Due to BBA2-CLDouble's more pessimistic behavior, it tends to induce more stall predictions than BBA2-CL (see Figure 3 versus Figure 2 (k)); this in turn causes BBA2-CLDouble to introduce multi-step quality changes more frequently.

6 SUMMARY, DISCUSSION AND NEXT STEPS

Based on a proof-of-concept implementation in `quic-go` and `goDASH`, we have investigated the impact of cross-layer information sharing on application-layer HAS performance. In particular, we have extended the BBA-2 ABR algorithm with a stall prediction and prevention component that exploits network throughput measurements collected at the QUIC packet level. Our testbed-based experimental results demonstrate that our cross-layer ABR approach aids to prevent buffer underruns during MPEG-DASH video playback under fluctuating network conditions where network throughput intermittently suffers severe throttling. The key contributing element to these results is that our cross-layer approach allows ABR mechanisms (like stall prediction) to operate on transport-layer metrics that are more accurate, fine-grained and timely compared to application-layer network measurements.

Preventing stalls, however, requires us to reduce MPEG-DASH segment quality abruptly; as P.1203 MOS shows, this does not always yield a better QoE. We argue, however, that P.1203 MOS only tells part of the story. For Video on Demand (VOD) content (e.g., Netflix and Disney+), aiming for the highest QoE makes sense. For time-sensitive content, such as live content or (critical) instructional content (e.g., emergency procedures or first aid instructions), keeping playback going – even with reduced quality – makes more sense. In future work, we would like to explore the feasibility and impact of cross-layer metrics sharing for the (low latency) live streaming use case (i.e., LL-DASH).

We recognize that our implementation is rudimental in its decision logic. Our focus for this paper was to investigate the feasibility of tapping into a more fine-grained information stream for ABR algorithms rather than in implementing and advanced stall prediction and prevention mechanism. In future work, we want to explore additional metrics, such as RTT calculations. Borrowing from congestion control developments (e.g., BBR [9]), fluctuating RTT measurements could indicate poor buffer management on the delivery path (i.e., bufferbloat), providing us with more detailed information to include in our ABR decision logic.

While this paper presents a first step in cross-layer techniques, it only explores a unidirectional cross-layer mechanism exchanging information from the OSI transport layer to the OSI application layer. A bidirectional approach would require exposing application layer knowledge to the transport layer or providing a mechanism to steer the transport layer. As is the case for most HAS clients, having no direct control over the transport layer socket is one of the contributors to the well-known ON-OFF problem. As a future work, we would like to explore an approach where the HAS client can directly control the transport layer's flow control and pacing mechanics utilizing cross-layer information exchange. Such an exchange could lead to more fair bandwidth usage scenarios with multi-client setups.

ACKNOWLEDGMENTS

Joris Herbots is a Ph.D. candidate at Hasselt University, supported by the Special Research Fund (BOF19OWB07). The research leading to these results has received funding from the European Union's Horizon Europe Programme under grant agreement 101070072, MAX-R (Mixed Augmented and eXtended Reality media pipeline).

REFERENCES

[1] 2023. Active QUIC implementations. https://github.com/quicwg/base-drafts/wiki/Implementations.

[2] 2023. dash.js: A reference client implementation for the playback of MPEG DASH via Javascript and compliant browsers. https://github.com/Dash-Industry-Forum/dash.js.

[3] 2023. quic-go: A QUIC implementation in pure go. https://github.com/quic-go/quic-go.

[4] 2023. quic-go: UDP receive buffer size recommendations. https://github.com/quic-go/quic-go/wiki/UDP-Receive-Buffer-Size.

[5] Saamer Akhshabi, Lakshmi Anantakrishnan, Ali C. Begen, and Constantine Dovrolis. 2012. What Happens When HTTP Adaptive Streaming Players Compete for Bandwidth?. In *Proceedings of the 22nd International Workshop on Network and Operating System Support for Digital Audio and Video* (Toronto, Ontario, Canada) *(NOSSDAV '12)*. Association for Computing Machinery, New York, NY, USA, 9–14. https://doi.org/10.1145/2229087.2229092

[6] Abdelhak Bentaleb, Bayan Taani, Ali C. Begen, Christian Timmerer, and Roger Zimmermann. 2019. A Survey on Bitrate Adaptation Schemes for Streaming Media Over HTTP. *IEEE Communications Surveys & Tutorials* 21, 1 (2019), 562–585. https://doi.org/10.1109/COMST.2018.2862938

[7] Divyashri Bhat, Amr Rizk, and Michael Zink. 2017. Not so QUIC: A Performance Study of DASH over QUIC. In *Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video* (Taipei, Taiwan) *(NOSSDAV'17)*. Association for Computing Machinery, New York, NY, USA, 13–18. https://doi.org/10.1145/3083165.3083175

[8] Mike Bishop. 2022. *HTTP/3.* RFC 9114. Internet Engineering Task Force.

[9] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2017. BBR: Congestion-Based Congestion Control. *Commun. ACM* 60, 2 (jan 2017), 58–66. https://doi.org/10.1145/3009824

[10] Bo Fu, Yang Xiao, Hongmei Deng, and Hui Zeng. 2014. A Survey of Cross-Layer Designs in Wireless Networks. *IEEE Communications Surveys & Tutorials* 16, 1 (2014), 110–126. https://doi.org/10.1109/SURV.2013.081313.00231

[11] Joris Herbots, Mike Vandersanden, Wim Lamotte, and Peter Quax. 2023. Vegvisir: A testing framework for HTTP/3 media streaming. In *Proceedings of the 14th ACM Multimedia Systems Conference* (Vancouver, BC, Canada) *(MMSys '23)*. Association for Computing Machinery, New York, NY, USA, 7 pages. https://doi.org/10.1145/3587819.3592550

[12] Te-Yuan Huang, Nikhil Handigol, Brandon Heller, Nick McKeown, and Ramesh Johari. 2012. Confused, Timid, and Unstable: Picking a Video Streaming Rate is Hard *(IMC '12)*. Association for Computing Machinery, New York, NY, USA, 225–238. https://doi.org/10.1145/2398776.2398800

[13] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2014. A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (Chicago, Illinois, USA) *(SIGCOMM '14)*. Association for Computing Machinery, New York, NY, USA, 187–198. https://doi.org/10.1145/2619239.2626296

[14] Jana Iyengar and Martin Thomson. 2021. *QUIC: A UDP-Based Multiplexed and Secure Transport.* RFC 9000. Internet Engineering Task Force.

[15] Stefan Lederer, Christopher Müller, and Christian Timmerer. 2012. Dynamic Adaptive Streaming over HTTP Dataset. In *Proceedings of the 3rd Multimedia Systems Conference* (Chapel Hill, North Carolina) *(MMSys '12)*. Association for Computing Machinery, New York, NY, USA, 89–94. https://doi.org/10.1145/2155555.2155570

[16] Daniele Lorenzi, Minh Nguyen, Farzad Tashtarian, Simone Milani, Hermann Hellwagner, and Christian Timmerer. 2021. Days of Future Past: An Optimization-Based Adaptive Bitrate Algorithm over HTTP/3 *(EPIQ '21)*. Association for Computing Machinery, New York, NY, USA, 8–14. https://doi.org/10.1145/3488660.3493802

[17] Robin Marx, Joris Herbots, Wim Lamotte, and Peter Quax. 2020. Same Standards, Different Decisions: A Study of QUIC and HTTP/3 Implementation Diversity. In *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC* (Virtual Event, USA) *(EPIQ '20)*. Association for Computing Machinery, New York, NY, USA, 14–20. https://doi.org/10.1145/3405796.3405828

[18] Robin Marx, Luca Niccolini, Marten Seemann, and Lucas Pardue. 2023. *Main logging schema for qlog.* Internet-Draft draft-ietf-quic-qlog-main-schema-05. Internet Engineering Task Force.

[19] Hyunwoo Nam, Kyung-Hwa Kim, and Henning Schulzrinne. 2016. QoE matters more than QoS: Why people stop watching cat videos. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. IEEE, San Francisco, CA, USA, 1–9. https://doi.org/10.1109/INFOCOM.2016.7524426

[20] Mirko Palmer, Malte Appel, Kevin Spiteri, Balakrishnan Chandrasekaran, Anja Feldmann, and Ramesh K. Sitaraman. 2021. VOXEL: Cross-Layer Optimization for Video Streaming with Imperfect Transmission. In *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies* (Virtual Event, Germany) *(CoNEXT '21)*. Association for Computing Machinery, New York, NY, USA, 359–374. https://doi.org/10.1145/3485983.3494864

[21] Tommy Pauly, Eric Kinnear, and David Schinazi. 2022. *An Unreliable Datagram Extension to QUIC.* RFC 9221. Internet Engineering Task Force.

[22] Esa Piri, Tiia Sutinen, and Janne Vehkaperä. 2009. Cross-layer architecture for adaptive real-time multimedia in heterogeneous network environment. In *2009 European Wireless Conference*. 293–297. https://doi.org/10.1109/EW.2009.5357979

[23] Esa Piri, Mikko Uitto, Janne Vehkaperä, and Tiia Sutinen. 2010. Dynamic Cross-Layer Adaptation of Scalable Video in Wireless Networking. In *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*. IEEE, Miami, FL, USA, 1–5. https://doi.org/10.1109/GLOCOM.2010.5683143

[24] Alexander Raake, Marie-Neige Garcia, Werner Robitza, Peter List, Steve Göring, and Bernhard Feiten. 2017. A bitstream-based, scalable video-quality model for HTTP adaptive streaming: ITU-T P.1203.1. In *Ninth International Conference on Quality of Multimedia Experience (QoMEX)*. IEEE, Erfurt. https://doi.org/10.1109/QoMEX.2017.7965631

[25] Darijo Raca, Maelle Manifacier, and Jason J. Quinlan. 2022. goDASH - GO accelerated HAS framework for rapid prototyping. In *2th International Conference on Quality of Multimedia Experience (QoMEX)*. IEEE, Athlone, Ireland.

[26] Werner Robitza, Steve Göring, Alexander Raake, David Lindegren, Gunnar Heikkilä, Jörgen Gustafsson, Peter List, Bernhard Feiten, Ulf Wüstenhagen, Marie-Neige Garcia, Kazuhisa Yamagishi, and Simon Broom. 2018. HTTP Adaptive Streaming QoE Estimation with ITU-T Rec. P.1203 – Open Databases and Software. In *9th ACM Multimedia Systems Conference*. Amsterdam. https://doi.org/10.1145/3204949.3208124

[27] SANDVINE. 2023. *2023 Global Internet Phenomena Report.* Technical Report.

[28] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K. Sitaraman. 2016. BOLA: Near-optimal bitrate adaptation for online videos. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. IEEE, San Francisco, CA, USA, 1–9. https://doi.org/10.1109/INFOCOM.2016.7524428

[29] Bi Xixi, Qin Zhiliang, and Ma Ruofei. 2022. Cross-Layer Joint Scheduling for D2D Communication in Cellular Systems. In *6GN for Future Wireless Networks*, Shuo Shi, Ruofei Ma, and Weidang Lu (Eds.). Springer, Cham, Huizhou, China, 130–144. https://doi.org/10.1007/978-3-031-04245-4_12

A ARTIFACT APPENDIX

A.1 Abstract

This artifact contains the source files (written in Go) for our custom quic-go [3] and goDASH [25] client, each containing the necessary communication mechanisms for the unidirectional cross-layer information exchange explained in Section 3.1. The BBA2, BBA2-CL and BBA2-CLDouble ABR algorithms are contained within the goDASH source files.

Additionally, this artifact contains Docker build files (i.e., dockerfiles) for our client and simulated network environment, to be used in the open source QUIC-HTTP/3 testing framework Vegvisir [11]. We also provide scripts for Vegvisir to automatically generate graphs from the experiment's collected metrics and to calculate the ITU-P.1203 O46 score.

Finally this artifact also contains the logs from our results, containing system information, metrics and graphs generated from the collected metrics.

A.2 Artifact check-list (meta-information)

- **Algorithm:** BBA2-CL, BBA2-CLDouble
- **Data set:** Audiovisual dataset by Lederer et al. [15]
- **Hardware:** X86-64 Linux
- **Output:** Vegvisir collects system log files and metrics captured by the client-shaper-server setup
- **How much time is needed to prepare workflow (approximately)?:** 25 minutes (excluding video dataset download times)
- **How much time is needed to complete experiments (approximately)?:** 1 hour
- **Publicly available?:** Yes
- **Code licenses (if publicly available)?:** GPL-3.0 license
- **Workflow framework used?:** Git clone artifact, git clone Vegvisir testing framework, configure Vegvisir, download and prepare video datasets, run experiment, observe results

A.3 Description

A.3.1 How delivered. All source files, scripts and configurations are available over at the artifact repository on GitHub: https://github.com/EDM-Research/cross-that-boundary-mmsys23-nossdav. The README provides an explanation of the repository structure.

A.3.2 Hardware dependencies. Any system capable of running Docker and Docker Compose should suffice. The results of this paper were produced on Arch Linux 6.1.7 X86-64 with Docker version 23.0.1, Docker Compose version 2.16.0 and Vegvisir version 2.0.0.

A.4 Installation

A step-by-step installation guide is provided over at the GitHub repository: https://github.com/EDM-Research/cross-that-boundary-mmsys23-nossdav. This guide provides information on how to create Docker images from the provided source files, how to configure the Vegvisir testing framework and how to prepare the audiovisual datasets.

A.5 Evaluation and expected result

After creating the required Docker images (i.e. the quic-go and goDASH client along with the network shaper) and configuring Vegvisir; from the root of the Vegvisir folder, run:

```
$ python -m vegvisir run paper_experiment_full.json
```

Vegvisir will display ongoing tests and show a general progress bar, the provided configuration should run for about an hour. Afterwards the results can be found in the `vegvisir/logs/cross_layer_paper/{datetime of run}` folder. A convenience script is available to quickly display all generated graphs in a web browser. See the GitHub repository README for more information.

The results acquired can be compared to our own results provided with this artifact in the `paper-logs/` folder.

A.6 Experiment customization

The included dockerfile for the quic-go and goDASH client is steerable through command line parameters. Providing custom arguments can be done via the Vegvisir experiment configuration file (see `paper_experiment_full.json` file in the `paper-utilities/vegvisir-configurations/` folder). The following parameters can be changed:

- **REQUESTS** URL to the MPEG-DASH manifest
- **ABR** Name of the ABR algorithm to be used in the experiment: `bba2`, `bba2XL-base`, `bba2XL-double` (note that these are internal names, they map to the BBA2, BBA2-CL and BBA2-CLDouble algorithms – as mentioned in the paper – respectively).
- **MAX_BUFFER** BBA2 maximum buffer size in seconds
- **INIT_BUFFER** Initial number of segments to download before starting playback
- **STREAM_SPEED** Factor applied to playback rate (1 = normal rate)
- **MAX_HEIGHT** Maximum height resolution in pixels to stream