IBC2023

A FRESH LOOK AT LIVE SPORTS STREAMING WITH PRIORITIZED MEDIA-OVER-QUIC TRANSPORT

Zafer Gurel[1], Tugce Erkilic Civelek[1], Ali C. Begen[1,2] and Alex Giladi[2]

[1] Ozyegin University (Turkiye), [2] Comcast (USA)

ABSTRACT

A QUIC-based low-latency delivery solution for media ingest and distribution in browser and non-browser environments is currently being developed for various use cases such as live streaming, cloud gaming, remote desktop, videoconferencing and eSports. Operating in an HTTP/3 environment (i.e., using WebTransport in browsers) or using raw QUIC transport, QUIC has the potential to revolutionize the media industry overcoming the limitations we face with the traditional approaches that impose TCP. This study explains the design methodology and explores possible gains with QUIC's stream prioritization features.

INTRODUCTION

Live sports broadcasting, where fans can stream live content on their connected devices or cloud gaming, where users can play together connecting from different parts of the world, has many demanding requirements. Nobody wants to hear a neighbor's cheers when a goal is scored before seeing it on the screen, making low-latency transport and playback indispensable. Similarly, high latency is intolerable when playing games in the cloud.

The existing HTTP ecosystem comprises solid foundational components such as distributed caches, efficient client applications and high-performant server software glued with HTTP. This formation allows efficient live media delivery at scale. However, the two popular approaches, DASH and HLS, are highly tuned for HTTP/1.1 and 2 running on top of TCP [1, 2]. The downside is the latency caused by the head-of-line (HoL) blocking experienced due to TCP's in-order and reliable delivery. The latest version of HTTP (HTTP/3) uses QUIC underneath instead of TCP. QUIC can carry different media types or parts in different streams. These streams can be multiplexed over a single connection avoiding the HoL blocking [1, 3]. The streams can also be prioritized (or even discarded) based on specific media properties (e.g., dependency structure and presentation timestamp) to trade off reliability with latency. DASH and HLS can readily run over HTTP/3, but they can only recoup the benefits if they use its unique features [6].

In the IETF, a new working group, Media over QUIC (MOQ), was formed in 2022 to study further the possible enhancements that QUIC may bring for low-latency live streaming [5]. The initial implementation of MOQ Transport (MOQT) is discussed during the IETF meetings, which is on its way to standardization [7]. As the initial discussions reveal, MOQT may improve the scalability of real-time, interactive media applications and the interactivity of live streaming applications.

This paper explains the MOQT design considerations and summarizes the enhancements we implemented in a previous study [8, 9]. Then, we investigate the prioritization schemes

that can be performed over this first implementation and show results for the on-time-display ratio under different bandwidth constraints.

MOQT DESIGN METHODOLOGY

MOQT offers a latency-configurable delivery protocol for transmitting content from one or more senders to the receiver(s) over zero or more relays utilizing either WebTransport (in browsers) or raw QUIC (otherwise), as shown in Figure 1.

Relays scale media delivery by forwarding incoming media to one or more relays or receivers without requiring a unique encoding for every recipient. In order to adapt to congestion and meet the application's latency requirements, relays choose what to deliver in what order or what to drop, depending on the particular metadata disclosed in the envelope of the incoming packets. Receivers can also compromise on quality and latency by determining the ideal time to wait for media, depending on their network conditions and user expectations.

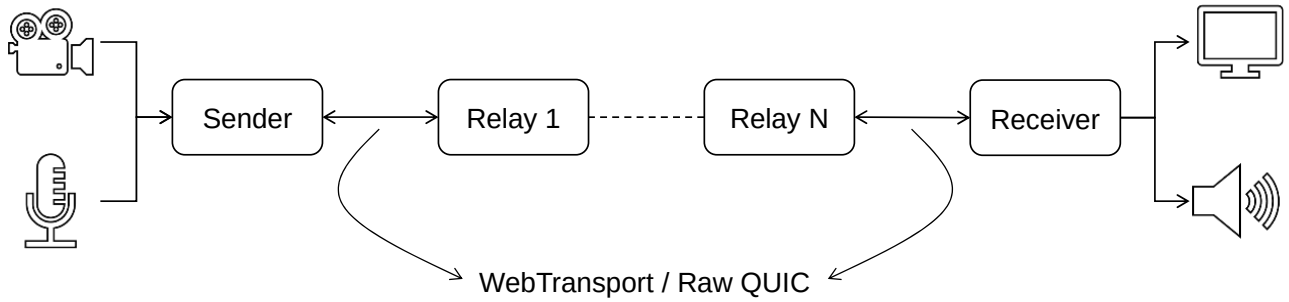

Figure 1: Simplified end-to-end deployment of MOQ.

The server side of the first MOQT demo (initially named Warp[1]) makes use of a customized quic-go[2] library with features like stream prioritization and APIs that expose the bandwidth and QUIC connection [10]. The server mimics live streaming by releasing a pre-encoded and pre-packaged media file chunk by chunk as soon as a client connects. In this implementation, each media frame is packaged as a CMAF chunk using the Common Media Application Format (CMAF, ISO/IEC 23000-19 [4]) standard.

The client application is a Web page that plays the live feed and includes a video element and establishes a connection to the server using the WebTransport API after the page has loaded and begins to receive QUIC streams. The CMAF segments are parsed and appended to the source buffer as they are received. The client uses a unidirectional, single QUIC stream to send the server control messages (such as play, pause and resume).

Enhancements to MOQT Demo

The first MOQT demo provided the necessary elements, but numerous others were required for a complete study. The improvements we made to create a testbed for experimenting with MOQT proposals are outlined below [9] and can be reached at [11]:

- addition of data keys under server-to-client informational messages to calculate end-to-end latency on the client side and throughput estimation on the server side,
- wall-clock time synchronization between the server and client,

[1] https://tinyurl.com/initial-warp-demo
[2] https://github.com/kixelated/quic-go

- addition of client-to-server control messages for transmitting client preferences to the server,
- addition of passive and active bandwidth measurement methods, and
- enhanced user interface to compare the real-time bandwidth measurements on both sides.

Prioritization Schemes

Prioritization is an important feature yet to be studied by the IETF MOQ working group. The network usually cannot maintain the intended order for media content – the order to decode the material and play it. Due to several reasons, a receiver cannot expect packets to be received in the order they were sent. The sending of specific streams may be delayed by packet loss or flow control.

Latency budget is the maximum acceptable delay between when a media unit is generated and when it will be consumed in a given application. The latency budget allows a client to buffer and re-order the incoming and possibly out-of-order packets for decoding. This, in turn, smooths the viewer experience by avoiding frame drops and increasing the on-time-display ratio (of the frames).

The ideal latency budget differs from application to application. For instance, the quality of experience is directly tied to the end-to-end latency for live sports. Therefore, a lower latency budget should be used for applications requiring low-latency delivery. Aiming for high-quality streaming using a small latency budget is a challenging task. It requires careful planning of each step in the streaming process, from encoding to transmission to decoding. Each step's small latency gain helps keep the latency budget low. Using a tool such as prioritization can make a difference.

Developing effective prioritization techniques for different types of applications is possible with MOQT. For low-latency live streaming, an approach may be to group I and P-frames with a higher priority than B-frames. For non-low-latency live streaming, high-resolution frames can be prioritized over low-resolution frames. Prioritization should not be limited to just the type of frames. For example, a user can also be given the option to choose to prioritize low latency over high quality or vice versa.

The current MOQT draft [7] describes two prioritization options (Send Order and Ordering with Priorities) and, in this study, we explore and compare the following two schemes.

Implicit Prioritization

The video frames are transmitted without specific prioritization in the delivery process. A single unidirectional QUIC stream is used to deliver the frames, and their send order is the same as their encoding order. This Implicit Prioritization is also called First Encode, First Send (FEFS). The delivery order is the same as the send order because QUIC guarantees the delivery order for the objects sent over the same stream, as illustrated in Figure 2. Nonetheless, in cases where there is congestion on the link or not enough available bandwidth to send all the frames (e.g., see Figure 3), queueing will occur and later frames will experience an increased delay.

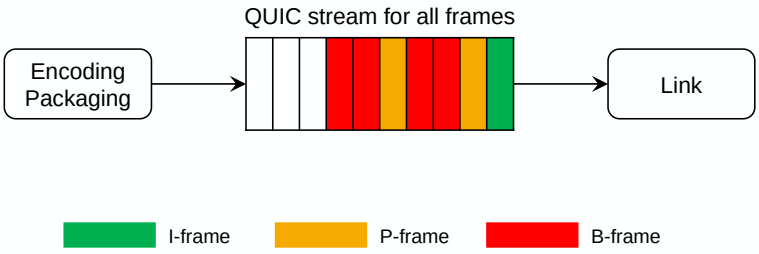

Figure 2: Illustration of implicit prioritization without congestion.

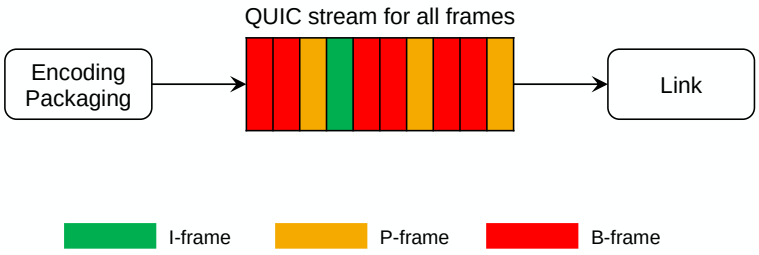

Figure 3: Illustration of implicit prioritization with congestion.

Prioritization by Frame Type

The video frames are prioritized according to their types (e.g., I, P and B). For each frame type, a separate unidirectional QUIC stream is created, and each stream is assigned a priority based on that type (as illustrated on the left in Figure 4), where the I-frames and B-frames have the highest and lowest priority, respectively. This way, it is expected that in a congested network, the timely delivery of I-frames is more likely than that of P and B-frames. The timely delivery of P-frames is also more likely than that of B-frames (as illustrated on the right in Figure 4).

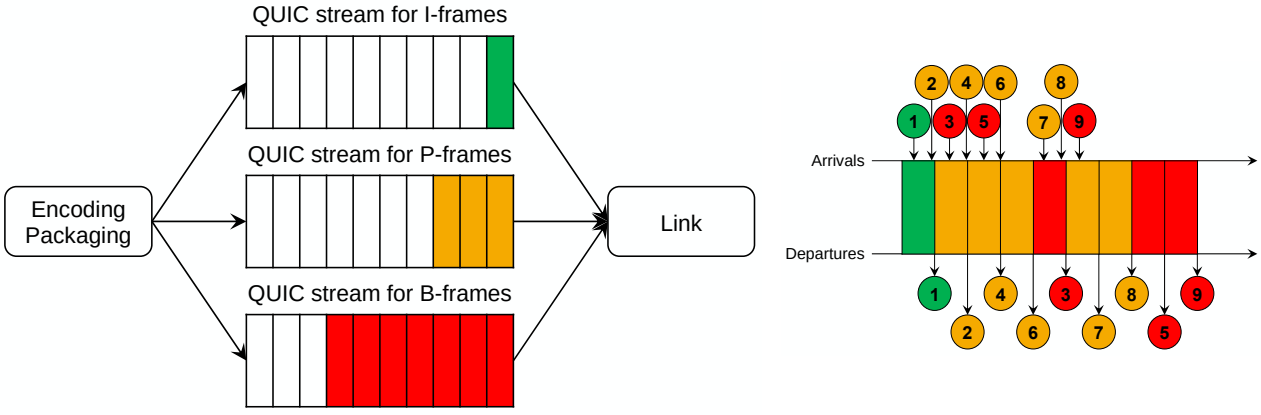

Figure 4: Illustration of prioritization by frame type.

EXPERIMENTS AND RESULTS

Experimental Setup

We ran several experiments with different configurations to compare the prioritization schemes. These experiments were conducted on a computer with an Intel Core i7-8750H

CPU (6 cores, 12 threads) and 32 GB of RAM, running Ubuntu 22.04.2 LTS with a kernel version 5.15.0. Node.JS and NPM versions were 18.13 and 9.3.1, respectively. The Go runtime version was 1.19.5. As the browser, Chrome 113.0.5672.63 was used to run the MOQT demo. For network emulation (e.g., applying a bandwidth constraint), tc NetEm[3] was used.

We used a pre-encoded (1280x720) test video to simulate live streaming. It was displayed for approximately two minutes in each experiment. The frame rate was 25 fps and the group-of-pictures (GoP) length was 50 frames (two seconds). Each frame was packaged into a CMAF chunk and each GoP was packaged into a CMAF fragment. The GoP structure was a sequence of frames where two B-frames were interleaved between the I and P-frames (i.e., IBBPBBP…).

The bitrate of the test video with the associated overheads was approximately 2.7 Mbps. Therefore, the frame latency variations were measured at three different bandwidth constraints: 2.7, 2.85 and 3 Mbps.

Performance Metrics

The metrics to evaluate the effectiveness of the prioritization are (i) latency variation depending on the bandwidth constraint, and (ii) the on-time-display ratio (OTDR) under different latency budgets. OTDR indicates the ratio of the number of frames displayed on time over the total number.

In multimedia applications, not all the received frames are necessarily displayed on time. A P-frame can be displayed only if the I-frame and any other preceding P-frame have been received. Similarly, B-frames can be displayed only if the referenced I and P-frames have been received.

When determining whether a frame is displayed on time, it is checked whether the frame(s) it depends on are received and decodable by this frame's presentation deadline. The methodology is summarized in the flowchart given in Figure 5.

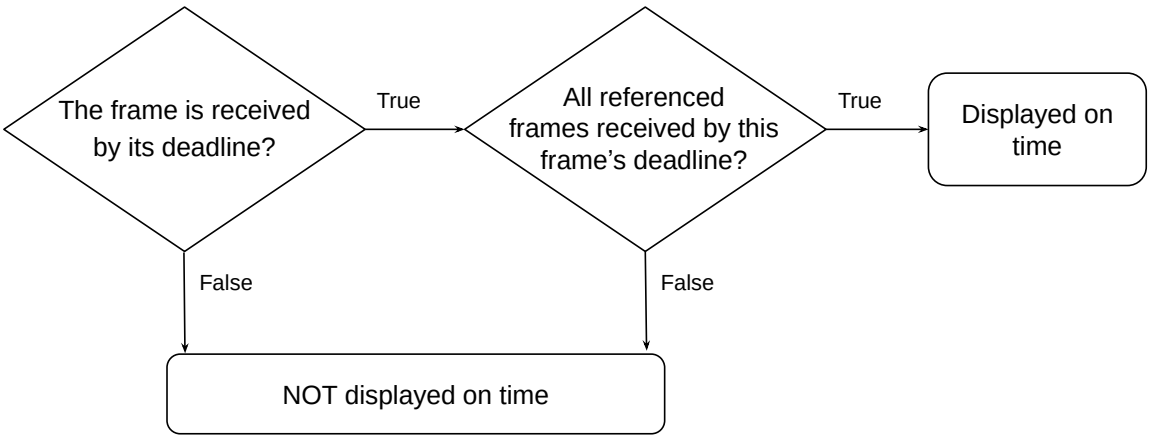

Figure 5: Calculation of the on-time-display ratio (OTDR) metric.

Results

In the tests, we measured the latency for different frame types, which were prioritized differently. Figures 6, 7 and 8 present the individual latency values for the I, P and B-frames,

[3] https://man7.org/linux/man-pages/man8/tc-netem.8.html

respectively. First, while the results show a latency variation among the frames of any given type, the variation is smallest for the I-frames and largest for the B-frames. At the same time, the latency variation among the I and P-frames does not change between the scenarios of bandwidth being constrained to 2.7 Mbps vs. 3 Mbps. On the other hand, the latency variation for the B-frames increases substantially when the bandwidth is constrained more. These are expected as prioritization limits the impact of the network congestion and the amount of latency experienced. Figure 6 also shows that if the latency budget is 700 ms or more, all I-frames can be displayed on time. At this latency budget (700 ms), Figure 7 reveals that some P-frames will not be displayed on time and Figure 8 reveals that most B-frames will not be displayed on time.

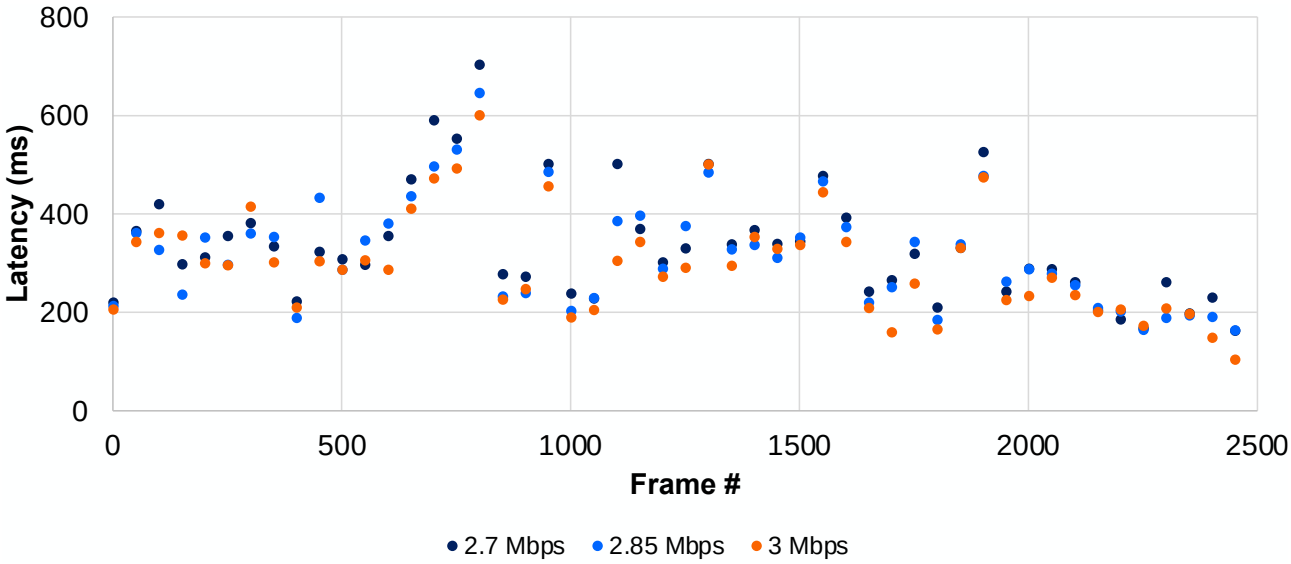

Figure 6: Latency variation for I-frames.

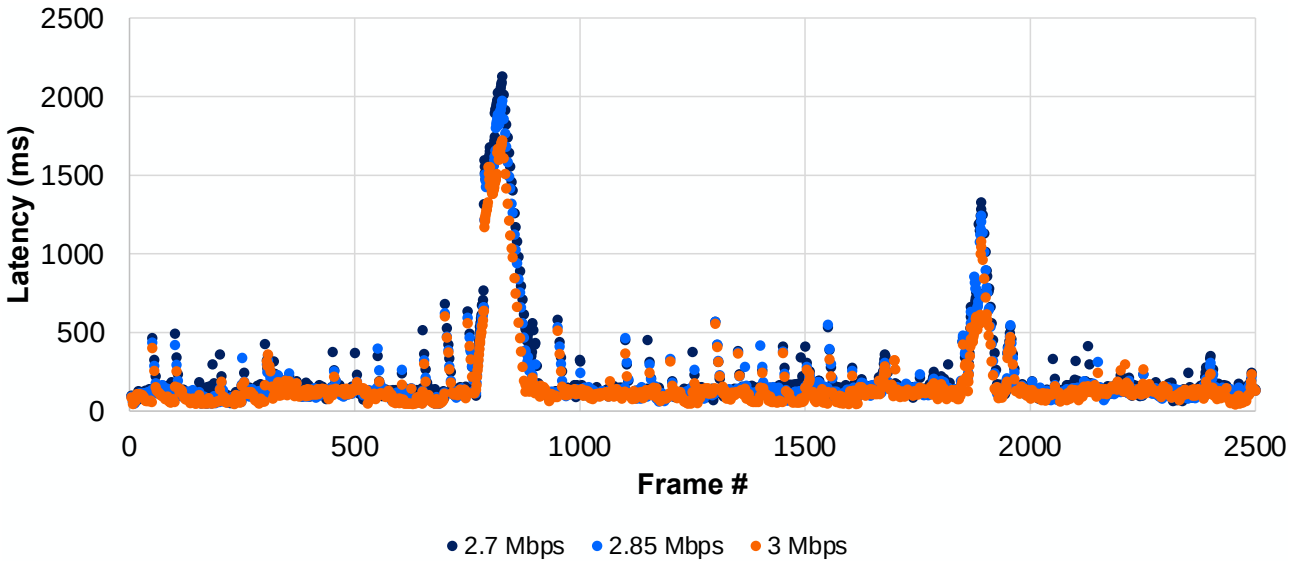

Figure 7: Latency variation for P-frames.

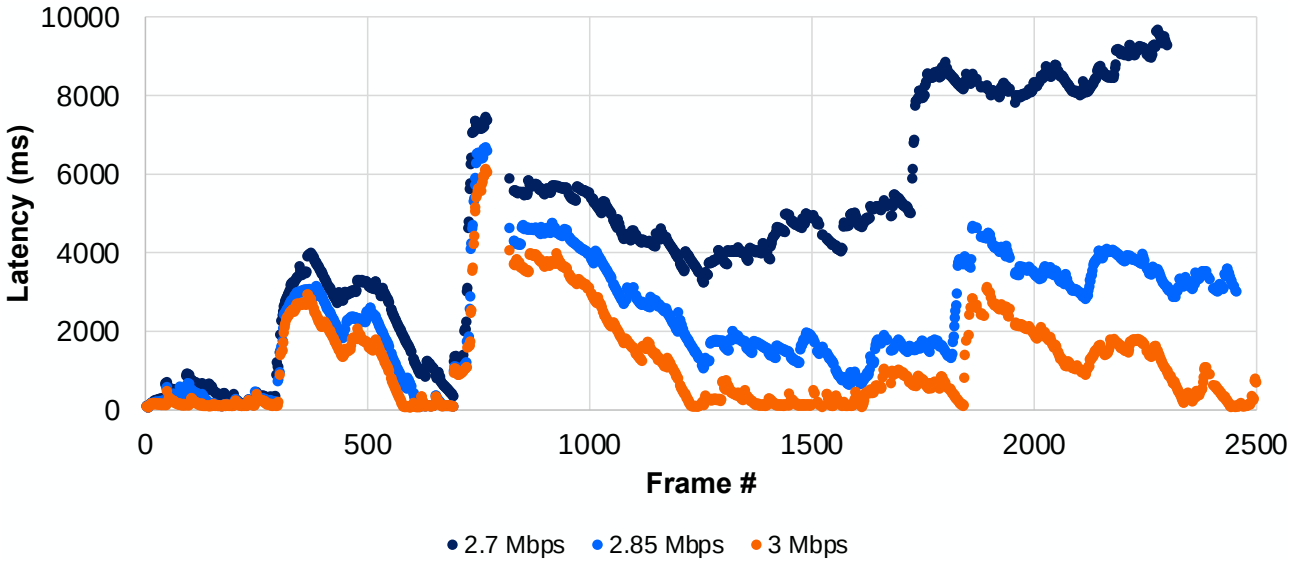

Figure 8: Latency variation for B-frames.

Figure 9 reveals the latency variation of all frame types when the bandwidth is constrained to 2.7 Mbps. We observe that latency values for the I and P-frames vary in a smaller range. In contrast, the variation for the B-frames is significantly larger.

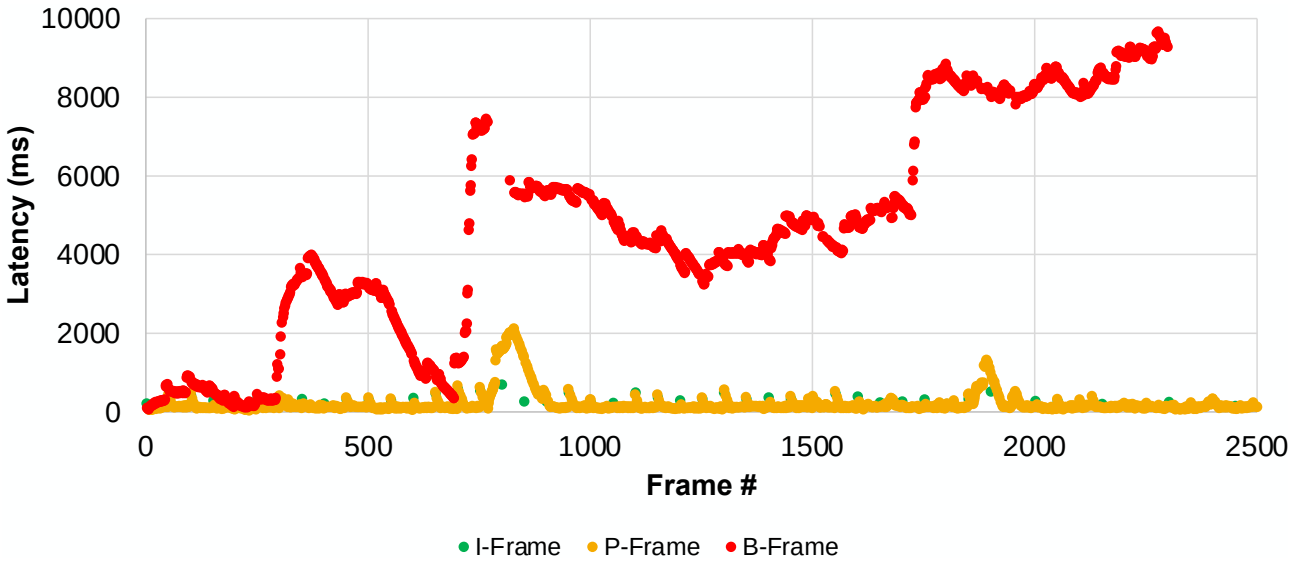

Figure 9: Latency variation for all frames at a bandwidth constraint of 2.7 Mbps.

Figure 10 shows the OTDRs for different bandwidth constraints under a latency budget of up to 3000 ms. While calculating the OTDR (of all frames of all types), the main idea is to check whether any given frame is received and can be displayed within that latency budget. The dashed lines illustrate the OTDRs with implicit prioritization (First Encode, First Send – FEFS). The dashed lines also represent the OTDR performance if we had used a single TCP connection (rather than one QUIC stream in a single QUIC connection). On the other hand, the solid lines illustrate the OTDRs with prioritization by frame type (FT) (using three QUIC streams in a single QUIC connection). The results show that MOQT enables us to improve the OTDR performance through better prioritization under the same bandwidth

constraint. Said differently, the same OTDR performance can be achieved with lower latency budgets if we apply a better prioritization scheme.

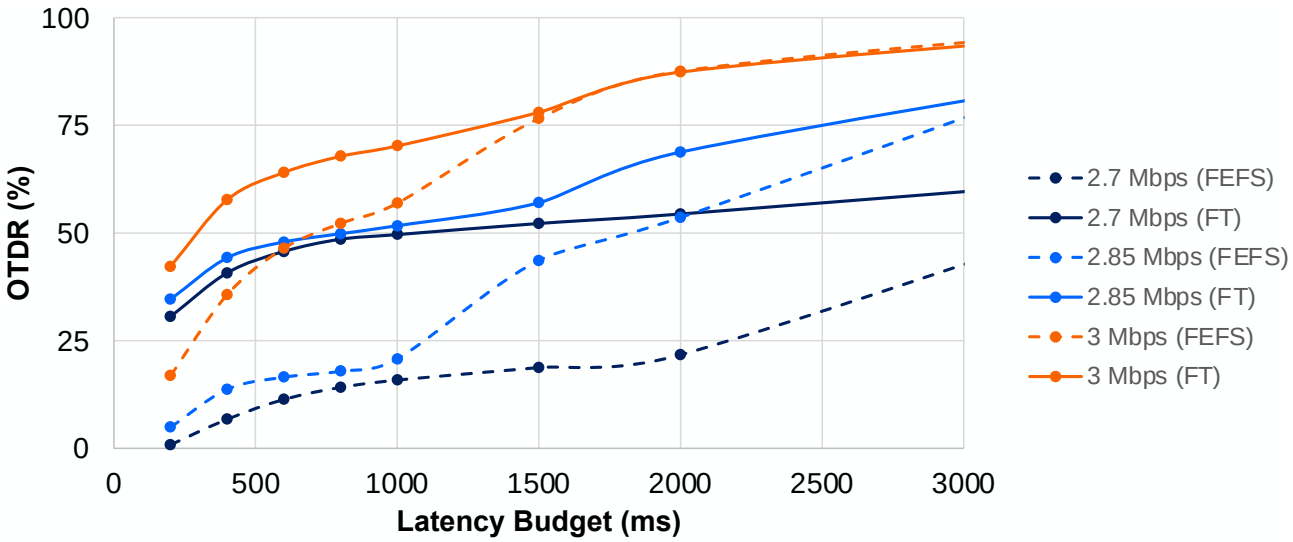

Figure 10: OTDR with FEFS and FT prioritization schemes.

CONCLUDING REMARKS

MOQT is currently under development and it potentially comes with many advantages. With the enhancements we implemented to MOQT, we can provide the client with better performance under the same resources. For low-latency use cases, prioritization always provides higher OTDRs, resulting in a better quality of experience or a fairer use of available resources. The next step in our research is to expand the testing to multi-client scenarios.

REFERENCES

[1] S. Arisu and A. C. Begen. Quickly starting media streams using QUIC. In ACM Packet Video Wksp., 2018 (DOI: 10.1145/3210424.3210426).

[2] R. Pantos, Ed. HTTP live streaming 2nd edition. [Online] Available: https://datatracker.ietf.org/doc/draft-pantos-hls-rfc8216bis/. Accessed on June 1, 2023.

[3] T. Shreedhar, R. Panda, S. Podanev, and V. Bajpai. Evaluating QUIC performance over web, cloud storage, and video workloads. IEEE Trans. Network and Service Management, June 2022.

[4] ISO/IEC 23000-19:2020 Information technology — Multimedia application format (MPEG-A) — Part 19: Common media application format (CMAF) for segmented media. [Online] Available: https://www.iso.org/standard/79106.html. Accessed on June 1, 2023.

[5] IETF. Media Over QUIC (moq). [Online] Available: https://datatracker.ietf.org/wg/moq/about/. Accessed on June 1, 2023.

[6] M. Nguyen, C. Timmerer, S. Pham, D. Silhavy, and A. C. Begen. Take the red pill for H3 and see how deep the rabbit hole goes. In ACM MHV, 2022 (DOI:10.1145/3510450.3517302).

[7] L. Curley, K. Pugin, S. Nandakumar, and V. Vasiliev. Media over QUIC Transport. [Online] Available: https://datatracker.ietf.org/doc/draft-lcurley-moq-transport/. Accessed on June 1, 2023.

[8] Z. Gurel, T. E. Civelek, and A. C. Begen. Need for low latency: media over QUIC. In ACM MHV, 2023 (DOI: 10.1145/3588444.3591033).

[9] Z. Gurel, T. E. Civelek, A. Bodur, S. Bilgin, D. Yeniceri, and A. C. Begen. Media over QUIC: initial testing, findings and results. In ACM MMSys, 2023 (DOI: 10.1145/3587819.3593937).

[10] Luke Curley. kixelated/warp-demo: Demo server and web player for the Warp live video protocol. [Online] Available: https://github.com/kixelated/warp-demo. Accessed on May 1, 2023.

[11] Streaming University. streaming-university/public-moq-demo: MOQ testbed. [Online] Available: https://github.com/streaming-university/public-moq-demo. Accessed on June 1, 2023.