Sur l'interaction entre les algorithmes de congestion de WebRTC et QUIC

David Baldassin, Guillaume Urvoy-Keller, Dino Martin Lopez Pacheco

Sur l'interaction entre les algorithmes de congestion de WebRTC et QUIC

David Baldassin[†⋆], Guillaume Urvoy-Keller[⋆] et Dino Martin Lopez Pacheco[⋆]

[†] *CoSMoSoftware,* [⋆] *Université Côte d'Azur, Nice, France*

Depuis récemment, QUIC dispose d'un mode datagramme non fiable avec un contrôle de congestion optionnel, adapté à la vidéo. Notre travail se concentre sur le transport des paquets RTP générés par WebRTC sur QUIC et examine l'intéraction des algorithmes de contrôle de congestion à ces deux niveaux.
QUIC étant implémenté en espace utilisateur, nous employons une approche expérimentale pour étudier diverses combinaisons d'algorithmes de contrôle de congestion de QUIC et WebRTC. Nos résulats montrent que dans une configuration à faible latence, l'utilisation du mode fiable de QUIC est bénéfique car il masque les pertes à WebRTC. Lorsque la latence est plus élevée, le mode datagramme avec un protocole de transport approprié au niveau de QUIC peut être intéressant. Néanmoins, dès que les conditions se dégradent, c'est le niveau applicatif qui prend la main.

Mots-clefs : WebRTC, QUIC, Congestion Control, RTP, Real-Time, WebTransport

1 Introduction

Recently, some efforts have been made to enable the delivery of real-time media and low latency streaming over the unreliable datagram extension of QUIC [P$^+$18, O$^+$23]. Unreliable means that losses are not recovered. However, a TCP-like congestion control algorithm (cc) can be used to control the tranport level rate. Our objective in this work is to assess the interplay between the congestion control (CC) mechanisms provided by WebRTC (that tunes the video encoder rate) and QUIC. We rely on an experimental approach that enables us to perform fully controlled experiments with a variety of QUIC implementations.

Our contributions are as follows: 1) We designed an interface to map the RTP flow produced by WebRTC onto QUIC streams. 2) We developed and publicly released a WebRTC over QUIC testbed based on a tunneling approach where the RTP packets sent by a WebRTC media server over UDP are injected in a highly controllable QUIC tunnel, in terms of CC algorithm or QUIC implementation. 3) Our results indicate that (i) in case of rapid change of path capacity, the WebRTC CC algorithm bypasses the QUIC one as it directly controls the video source rate, (ii) any QUIC CC algorithm can be used in datagram mode and (iii) the stream mode of QUIC (that recovers losses) can be interesting for small to medium latency (less than 50 ms) scenarios especially when combined with a delay based CC algorithm like COPA. 4) We uncovered some inconsistencies between and also inside the implementations, which can have a detrimental impact on the media flow.

2 A QUIC/WebRTC testbed

QUIC is a relatively new standard and there is no legacy implementation that could be used as unique reference point to perform experiments. We considered two implementations in this study: (1) **Mvfst** [Fac22] which is the QUIC implementation of Meta platforms for its support of the QUIC datagrams extension and several congestion algorithms : NewReno, Cubic, Copa, BBR or none. 2) **Quic-go** (v0.29.0) [S$^+$22] since it was used in previous experiments for RTP over QUIC [E$^+$21].

High level design: A WebApp is used to control the experimental set up and receive the media streams over a WebRTC peerconnection. The video stream is sent by a Medooze server [Mur22], an open source

WebRTC media server that the Webapp contacts through a websocket. WebRTC is emitting video packets in the form of RTP packets that are encapsulated into UDP packets. We intercept the RTP flow and inject it into a QUIC tunnel. The Webapp controls each QUIC tunnel endpoint and provides the ability to configure the test session. A public release of the testbed is available at: `https://github.com/dbaldassi/`.

WebRTC over QUIC: As the RTP over QUIC draft suggests [O+23], we encapsulate one RTP packet per QUIC datagram. For QUIC streams, we open one unidirectional stream for each RTP packet (and then close it), to avoid any head of line blocking.

3 Transport and Application Layer Congestion Control Algorithms

QUIC CC Algorithms: The QUIC implementations we considered feature the following algorithms: NewReno, Cubic, BBR [C+16] and COPA [V+18]. Mvfst implements all these algorithms while Quic-go only features NewReno. NewReno and Cubic are two emblematic representatives of loss based algorithms. As such, they are not well suited for multimedia transfers but will serve as reference. BBR and COPA, on the other hand, are clearly targeting time sensitive communications. They monitor the delay to avoid bufferbloat and adjust a target emission rate. COPA is a pure delay-based CC algorithm. There exists three versions of BBR that are either pure delay-based or also monitor loss events. Inspection of the mvfst BBR code shows that the version in our testbed relies both on the delay and loss signals.

Medooze (v0.120.0) bandwidth estimation: Several CC algorithms have been proposed for WebRTC, SCReAM [J+17], NADA [Z+20] and GCC [H+16]. Medooze CC algorithm borrows characteristics from COPA [V+18].

4 Experiments

We use the same video for all experiments. The video sequence is chosen in such a manner that the video encoder will produce Constant Bit Rate (CBR) stream, whatever the target bit rate picked by the WebRTC cc algorithm. Its nominal bit rate is 2000 kbps, meaning that if more bandwidth is available, the encoding rate will remain at 2 000 kbps.

To investigate the interplay between the two levels of CC, we performed the following experiments: (i) **Link capacity limitation**: the channel capacity varies over time. ; (ii) **Random losses**: as the video signal is relatively immune to random losses, we expect to trigger a response from the QUIC CC and not necessarily from WebRTC and (iii) **Latency**: as the WebRTC CC responds to latency variation and some of the QUIC CC algorithms to random losses, this scenario enables to explore the interplay between the two CC levels.

4.1 Link Capacity

In this experiment, the capacity of the link varies over time between 500 and 2500 kbps by periods of 30 seconds, see A key observation from fig. 1 is that we have no difference whether we use raw UDP transport layer with no safety mechanism, or QUIC in datagram mode or in the reliable stream mode. There is in fact **a clear dominance of the WebRTC CC algorithm over the transport layer one in this scenario.** This is because the WebRTC CC algorithm reacts immediately to this major change in the network conditions by reducing the target encoding rate, which limits the number of losses and delay increase that can be observed from the transport level CC algorithm. There are still some losses observed by QUIC but they are so few that the decrease of the congestion window does not affect the video flow.

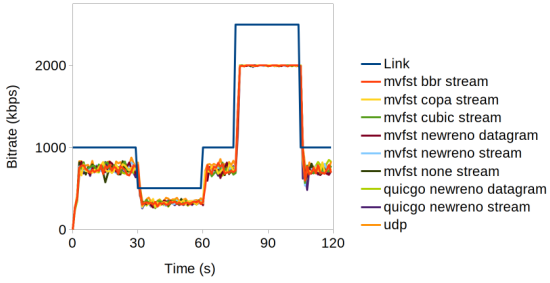
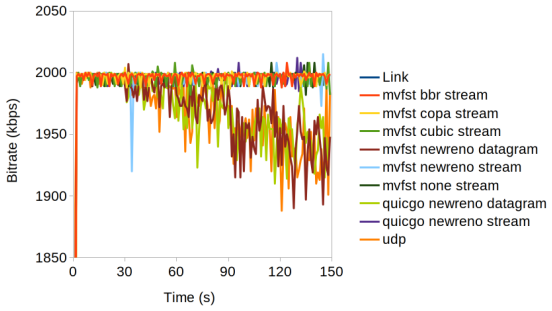


Fig. 1: Received media bitrate at WebRTC client

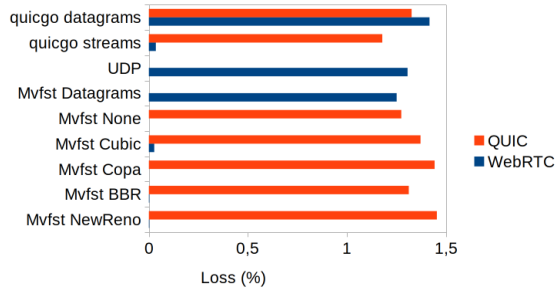
4.2 Random losses

To investigate how both congestion controllers react to random losses, we set a constant link limit to 2.5 mbps, i.e. higher than the maximum encoding rate. We then increase the random loss percentage each 30 seconds with the following steps: 0%, 0.5%, 1%, 2%, 3%.

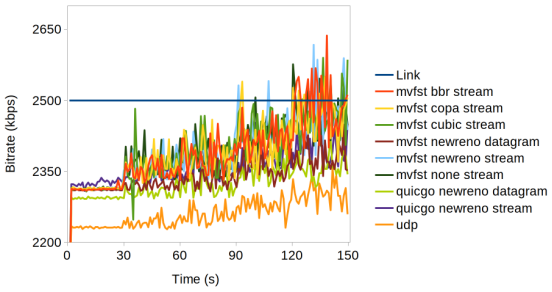
As we can see on fig. 2a, the bitrate decreases over time for QUIC in datagram modes (both with mvfst and quic-go) and UDP as in these cases, the RTX mode is triggered and part of the available bandwidth is reserved for retransmissions. In contrast, the stream mode retransmits lost packets within a round trip time so **the media server does not detect any loss**, whatever the CC algorithm is, including 'none'. This is illustrated by Figure 2b that reports QUIC and WebRTC detected losses, averaged over the full set of loss rates considered.



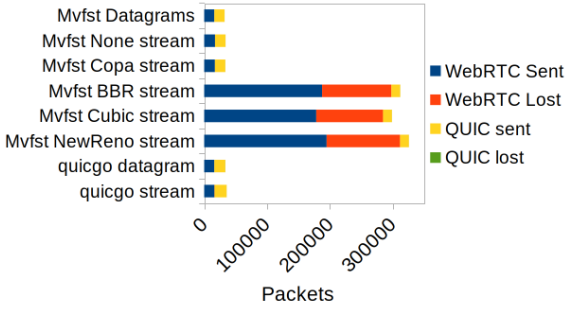
(a) Media bitrate by the client.



(b) Packets loss for QUIC and WebRTC



(c) QUIC bitrate at the QUIC server.



(d) Mvfst w/ 0.5% loss rate and 50ms latency.

Fig. 2

When moving to the transport layer, we notice that the QUIC bitrate increases with an increasing loss rate, as can be seen in fig. 2c. This is primarily because of retransmissions that can occur at the transport or application layer (RTX mode). Variations can be explained by the nature of the lost frame, key or reference.

To sum up, in this scenario, the stream mode of QUIC algorithms is beneficial as it hides the losses to the application layer. However, this result holds for low RTTs — 1 to 2ms in our experiments. We investigate more challenging scenarios in the next section.

4.3 Latency

In this section, we investigate the impact of latency, with four scenarios: 0ms, 25ms, 50ms and 100ms. We used a fixed loss rate of 0.5% that corresponds to a typical value observed in our production deployments.

For a latency below 25 ms, we observed no losses at the WebRTC level when QUIC is in stream mode, irrespectively of the exact CC algorithm. This means that the losses are compensated by QUIC. The picture is completely different above 50ms of latency, see fig. 2d that reports the loss rate for all scenarios and a 50ms latency.

These results can be explained as follows. First note that 25 to 30 ms is a typical buffer length in WebRTC. Hence, if the network latency is smaller than this value, the stream mode is efficient and hides the losses to WebRTC. Above 50 ms, the recovery mode of the stream mode can be detrimental to the video flow. This is typically what we observe in fig. 2d for the NewReno, BBR and Cubic cases: the media server enters in RTX mode and sends new key frames to reset the stream. Key frames are way larger than differential ones (by a factor of 10 approximately), which explains the rate increase (to about 18 mbps) that overshoots the channel capacity of 2.5 mbps and completely breaks the video flow.

However, the Quic-go implementation in stream mode with New Reno does not impact the video flow. This is also the case for Mvfst in stream mode with COPA, which we believe is caused by the latency increase that COPA perfectly detects and limits its transmission rate. Since there is a very complex interaction resulting from the mapping and adaptation of WebRTC onto QUIC, it is very hard to decide which one, Mvfst or Quic-go, provides the normal behavior. Hence, we conclude that **for clients with a good network latency, typically 5G or FTTH, using the stream mode of QUIC is a good solution. For larger latencies, the datagram mode is a better choice, if coupled with the appropriate CC algorithm.**

5 Conclusion

Our work sheds light on the interaction between QUIC and a live streaming mechanism like WebRTC. Our in-depth tests have uncovered how the CC algorithms at the two levels interact, as well as the impact of the QUIC mode (steam or datagram). Overall, the datagram mode has no detrimental effect on the WebRTC CC algorithm, which most of the time controls the situation as it directly tunes the encoding rate. The stream mode also has some merits, especially when combined with a pure delay-based CC algorithms like COPA, as it can alleviate the work of the WebRTC CC algorithm. We also uncovered some internal inconsistencies in mvfst implementation and discrepancies between mvfst and QUIC-go.

As future work, we would like to test other implementations of QUIC and do experiments in the wild.

References

[C+16] N. Cardwell et al. Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time. 14(5):20–53, 2016.

[E+21] M. Engelbart et al. Congestion control for real-time media over quic. In *EPIQ*, 2021.

[Fac22] Facebook. mvfst: An implementation of the QUIC transport protocol. `https://github.com/facebookincubator/mvfst`, 2022.

[H+16] S. Holmer et al. A google congestion control algorithm for real-time communication draft-ietf-rmcat-gcc-02. `https://datatracker.ietf.org/doc/html/draft-ietf-rmcat-gcc`, 2016.

[J+17] J. Johansson et al. Self-Clocked Rate Adaptation for Multimedia. RFC 8298, 2017.

[Mur22] S. G. Murillo. Medooze media server. `https://github.com/medooze/media-server`, 2022.

[O+23] J. Ott et al. Rtp over quic (roq). `https://datatracker.ietf.org/doc/html/draft-ietf-avtcore-rtp-over-quic`, 2023.

[P+18] C. Perkins et al. Real-time audio-visual media transport over quic. In *EPIQ*, 2018.

[S+22] M. Seemann et al. A QUIC implementation in pure go. `https://github.com/lucas-clemente/quic-go`, 2022.

[V+18] A. Venkat et al. Copa: Practical Delay-Based congestion control for the internet. In *NSDI*, 2018.

[Z+20] X. Zhu et al. Network-Assisted Dynamic Adaptation (NADA): A Unified Congestion Control Scheme for Real-Time Media. RFC 8698, 2020.