Congestion Control for Real-time Media over QUIC

Mathis Engelbart
Technical University Munich
Munich, Germany
mathis.engelbart@tum.de

Jörg Ott
Technical University Munich
Munich, Germany
ott@in.tum.de

ABSTRACT

QUIC is the new transport protocol for the Internet, designed for secure, reliable communication, especially with the web in mind. While multimedia streaming, allowing for some playout delay, has been widely run on top of reliable transport protocols, conversational multimedia usually requires unreliable ones, often using RTP over UDP. A recent extension to QUIC supports unreliable datagrams within QUIC connections so that also real-time media can be supported. In this paper, we investigate a recent design for RTP over QUIC with a focus on congestion control and the related signaling. We implement a strawman using Gstreamer and quic-go and evaluate different permutations of congestion control algorithms and signaling in a simple testbed.

CCS CONCEPTS

• **Networks → Transport protocols**.

KEYWORDS

RTP, QUIC, Congestion Control, SCReAM, New Reno

ACM Reference Format:
Mathis Engelbart and Jörg Ott. 2021. Congestion Control for Real-time Media over QUIC. In *Workshop on the Evolution, Performance and Interoperability of QUIC (EPIQ '21), December 7, 2021, Virtual Event, Germany.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3488660.3493801

1 INTRODUCTION

Carrying real-time multimedia traffic across the Internet requires transport protocols that keep media senders in control of transmission timing and error correction and give receivers immediate access to received media packets for playout management and error concealment. Therefore, real-time transport protocols such as RTP [25] run on top of UDP (or DTLS, DCCP) to avoid undesirable feature interactions with lower layers such as head-of-line (HoL) blocking. Without much support from the underlying transport, RTP implements its own control signaling (RTCP) to monitor media session quality [3, 25], perform error (e.g., [2, 14]) and congestion control [21, 23].

QUIC [11] offers secure, reliable transport on top of UDP. Numerous efforts explored how to allow QUIC to also carry real-time media [20]. Mapping traditional DASH media streaming to QUIC [1, 15, 16] appears rather straightforward as DASH is designed to run on top of reliable transports in the first place. To prevent HoL blocking when carrying conversational real-time media over QUIC, one suggestion included mapping each video frame to a new stream and canceling streams as soon as the frame can no longer be delivered in time.

With the advent of proposals that extend QUIC to support unreliable datagrams [19], carrying conversational real-time media over QUIC no longer requires the above workarounds. Instead, media packets can be mapped to QUIC datagram frames and thus RTP over QUIC can be easily implemented. After early discussions [18, 20], several concrete—and quite similar—mappings for RTP onto QUIC have been proposed [8, 17].

With such appropriate mapping in place, two main issues remain for carrying RTP over QUIC: (1) How to do proper real-time media congestion control, given that RTP realizes its own congestion control mechanisms [6, 13, 28] as does QUIC [10]? (2) For RTP-based congestion control, how to leverage information readily available within QUIC and avoid, or at least minimize, duplicate signaling in RTCP? In this paper, we use the RTP mapping defined in [17] and compare different approaches to congestion control—SCReAM [13] at the RTP level vs. NewReno in QUIC—as well as explicit RTCP signaling vs. inferring RTCP feedback information from QUIC. We implement the different schemes by extending GStreamer [5] and integrating it with quic-go [22] and evaluate 14 different permutations in a simple testbed.

2 MAPPING RTP TO QUIC

The above two Internet Drafts describing a mapping of RTP onto QUIC use QUIC's unreliable datagram extension [19], and a flow identifier prepended to each datagram to demultiplex multiple RTP sessions on the same QUIC connection.

QUIC datagram frames do not count to any flow control limits, which may lead to dropped datagram payloads if a receiver does not have the resources to handle them. Unlike flow control, connection-wide congestion control also applies to QUIC datagram frames, as sent datagram frames increase link utilization and may thus lead to link congestion. Datagram frames have to be acknowledged by a receiver because they are so-called ack-eliciting frames, although they are not retransmitted if lost [11].[1] Additionally, the current version of the draft allows a QUIC implementation to expose these acknowledgments to an application, a feature of which we will make use of in section 3.4. The draft also encourages implementations to provide an API for prioritizing datagram frames relative to each other and QUIC streams.

ACM ISBN 978-1-4503-9135-1/21/12.
https://doi.org/10.1145/3488660.3493801

[1] Note that some recent proposals also suggest non-ack-eliciting datagram frames but turn ACKs off is optional so that desirable reception feedback could be maintained. See https://github.com/quicwg/datagram/issues/42 for the corresponding discussion.

If the RTP to QUIC mapping is implemented as an additional layer on top of an existing QUIC implementation, the QUIC implementation must fulfill specific interface requirements. Since the proposed mappings use the QUIC unreliable datagram extension, it is obvious that the QUIC implementation has to implement that extension. Section 3.4 will introduce an optimization of real-time congestion control for RTP over QUIC, which adds further requirements to the QUIC implementation and its API.

Due to the bandwidth limitations of most links and the large data rates of raw video, video is encoded before being encapsulated in RTP packets. When using congestion control as outlined in section 3.2, the congestion controller produces a target bitrate, which will be fed back to the media encoder. The remainder of this work will focus on calculating a good bitrate to configure the media encoder to produce output that optimally utilizes the available bandwidth.

3 CONGESTION CONTROL

RTP and QUIC both provide some form of support for congestion control and it is important to carefully design congestion control when using both combined. This section explains the basics of congestion control in QUIC and RTP. We then discuss why it may be useful to use QUIC and RTP congestion control simultaneously and the impact it may have in 3.3 before looking at how QUIC features can help reduce RTCP overhead in section 3.4.

3.1 Congestion Control for QUIC

The QUIC specification does not mandate a certain congestion control algorithm to use. Instead, it suggests using an algorithm similar to TCP NewReno, while allowing implementations to use a different algorithm by providing connection statistics that may be used to implement different congestion control algorithms [10].

3.2 Congestion Control for Real-time Media

Real-time applications are delay-sensitive. This makes the loss-based congestion algorithms (e.g., NewReno or Cubic) as they are widely used in TCP less applicable for real-time traffic. Loss-based algorithms alternate between filling queues on network elements and draining them when congestion is detected. This causes a lot of delay variation and large bitrate changes, which is not acceptable for real-time media coding [4].

Congestion control for real-time media has been the topic of the RMCAT working group of the IETF in recent years. RTP itself does not include any form of congestion control. Still, congestion control can be added through an extension and using RTCP to fulfill the feedback requirements of any congestion control algorithm.

A way to avoid the problems of loss-based algorithms for real-time media is to use congestion control algorithms that are delay-based or a mix of delay-based and loss-based.

With Self-Clocked Rate Adaptation for Multimedia (SCReAM) [13] and Network-Assisted Dynamic Adaptation (NADA) [28], the RMCAT working group has published two real-time congestion control algorithms. A third one, Google Congestion Control (GCC) was published by Google [6]. The RMCAT working group additionally published an RTCP feedback format for congestion control, which can be used for all of the 3 algorithms [23].

Like many loss-based congestion control algorithms, all of these algorithms require the receiving side to provide feedback about the arrival of packets such as for whether a packet has been received or lost and when the packet has arrived. The receiver can then send the feedback to the sender via RTCP, which uses it to calculate the maximum available bandwidth and configure the media encoder to utilize the link as much as possible without exceeding its capacity.

In this paper, we use SCReAM as an example of a real-time congestion control algorithm.

3.3 Mixing QUIC- and Real-time Congestion Control

QUIC's streams and datagrams open up the possibility to send many data streams concurrently. A question that comes up is, how traffic should be congestion-controlled if the same connection is used to send real-time traffic as well as non-real-time traffic simultaneously. A simple option would be to prohibit the transmission of non-real-time data on the same connection, which is currently used for an RTP session. This RTP-only connection could then easily be congestion-controlled by replacing QUIC's default congestion control algorithm with one of the proposed real-time congestion control algorithms. However, it would be useful in many scenarios, to be able to carry reliable and real-time data within the same connection. An example would be a typical multimedia conferencing session comprising audio and video streams as well as a slide presentation or shared whiteboard. The audio and video streams should of course arrive in real-time, even if that comes at a cost of losing some packets. The presentation, on the other hand, does not have such strict real-time requirements but should ensure that the content is delivered reliably and lost packets should be retransmitted. Such a multimedia session, as described here, would benefit from a shared connection setup and avoid competing for the same resources independently. A downside of this approach is the need for possibly complex prioritization between different data transmissions on the same QUIC connection, as we will see in section 6.4.

To explore and evaluate the different options of using congestion control for RTP over QUIC, we implemented different combinations of congestion control algorithms. When we are not using a real-time congestion control algorithm, we still need to find a suitable bitrate to configure the media encoder. While real-time congestion control protocols explicitly expose such a sending rate, QUIC does not. To cope with this, we implemented a simple algorithm, which uses the growth of the RTP queue length, to decide whether to increase, decrease or keep the bitrate at the current level. The levels used by this algorithm are the following discrete values tailored to our use case: 256 kbit/s, 512 kbit/s, 768 kbit/s, 1024 kbit/s, 1280 kbit/s, which somewhat resembles a very simple DASH scheme, but without any intention to mimic the established complex DASH mechanisms.

We considered the following combinations of algorithms:

- **Only Trivial Rate Control** No congestion controller is used in QUIC, but we use the trivial rate controller to set the encoder bitrate.
- **Trivial Rate Control and NewReno** Enables QUIC's internal congestion control using NewReno, which applies to all outgoing traffic in addition to the trivial rate controller.

- **Only SCReAM** RTP is congestion-controlled using SCReAM, and congestion control feedback via RTCP.
- **SCReAM and NewReno** Combining QUIC and RTP congestion control using NewReno and SCReAM. NewReno applies to all outgoing traffic, while SCReAM only applies to the RTP datagrams.
- **SCReAM without RTCP** Only RTP is congestion-controlled using SCReAM, but instead of using RTCP feedback, only rely on the connection state provided by QUIC as described in section 3.4.
- **SCReAM and NewReno without RTCP** Combining QUIC and RTP congestion control using NewReno and SCReAM and again only rely on the connection state provided by QUIC instead of using RTCP.

3.4 RTP Congestion Signaling

As mentioned in section 3.1 and section 3.2, the QUIC congestion controller as well as the real-time congestion controller require some feedback information. Since the QUIC connection already gathers a lot of the feedback that is also needed by the real-time congestion controller, it should be easy to re-use some of this information, to reduce the RTCP overhead introduced when the real-time congestion controller has to gather all the feedback by itself. The inputs listed in the following table are required for the different real-time congestion controllers. Not all congestion controllers might need all the inputs, but all should be able to compute an optimal target bitrate using not more than these inputs.

Name	Content
`t_current`	A current timestamp.
`pkt_status_list`	A list of RTP packets that were acknowledged by the receiver.
`pkt_delay_list`	For each acknowledged RTP packet, the delay between send- and receive-timestamps of the packet.
`latest_rtt`	The latest RTT sample measured by QUIC.
`min_rtt`	The minimum observed RTT.
`smoothed_rtt`	An exponentially weighted moving average of the observed RTT values.
`rtt_var`	The mean deviation in the observed RTT values.
`ecn`	Optionally ECN marks if supported by the network.

To acquire this information, we need to extend the QUIC implementation interface, which we briefly discussed in section 2. Firstly, the unreliable datagrams implementation must also provide an interface to signal the datagram frame's acknowledgments or losses to the application. Secondly, QUIC needs to expose its connection statistics, such as the path RTT, to the application. Lastly, an optional enhancement would be signaling explicit congestion notifications (ECN) from QUIC to the RTP mapping. A congestion controller could use ECN signals for optimization, but this cannot be a hard requirement since they are not reliably available to begin with.

To locally generate the state information that RTCP would usually provide, the sender has to keep track of all RTP packets sent. Using the acknowledgement signals, the sender can produce the `pkt_status_list`. The `pkt_delay_list`, can be generated by calculating an estimation of the receive time of the packet by dividing the last known RTT by two. All of the other inputs listed above are either trivial to compute or gather from the QUIC implementation. Optimization of the `pkt_delay_list` can be made by replacing the receive time calculation using RTT by using an explicit one-way delay measurement as proposed in [7].

The generated feedback can now be passed to the congestion controller, either at regular intervals or when a particular packet or frame was acknowledged, depending on the needs of the congestion controller in use.

4 IMPLEMENTATION

In this section, we describe the implementations for our evaluation. We use Gstreamer as the video source and sink, all the way to the RTP encapsulation. We then implement two bridge modules to forward the RTP packets from/to Gstreamer: one that connects to the quic-go implementation for RTP-over-QUIC and one that simply sends and receives RTP in UDP packets. This structure simplifies switching between the different encapsulation formats. Since the plain UDP forwarding is straightforward—we simply encapsulate an RTP packet in a UDP datagram—this section focuses on RTP over QUIC, for which we use the packet format described in [17].

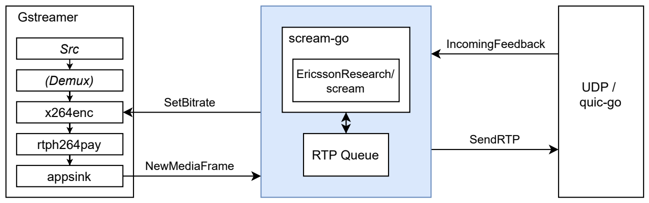

Figure 1: Sender side architecture for integrating Gstreamer, SCReAM and quic-go

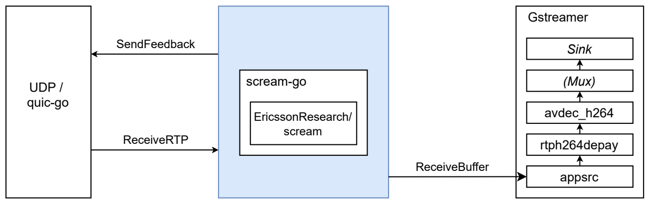

Figure 2: Receiver side architecture for integrating Gstreamer, SCReAM and quic-go

Figure 1 shows a conceptual view of the sender-side application: a Gstreamer pipeline reads a raw video from a source file, encodes it using H.264, and packetizes the encoded video in RTP packets. We use the Gstreamer `appsink` element to pass the resulting buffer to a Go-application, which uses an internal queue for RTP packets. Before the next packet from the queue is passed to quic-go, the application prepends the flow identifier.

On the receiving side, shown in figure 2, quic-go accepts each datagram and passes it to our application, which strips off the flow identifier that indicates the stream to which the packet belongs and passes it on to a corresponding Gstreamer pipeline. The pipeline depacketizes and decodes the video and stores the raw video in a file.

To integrate SCReAM, we build a Go-wrapper around the SCReAM implementation, the latter of which is provided as open source by Ericsson Research [12]. If SCReAM is enabled, the sending side notifies the SCReAM implementation about incoming packets from the Gstreamer pipeline and each packet sent by quic-go.

5 TEST ENVIRONMENT

In this section, we describe the test setup to evaluate the RTP over QUIC implementations. First, we sketch out the general architecture of our testbed and explain our choice of parameters to simulate different network conditions. In 5.3 we show which metrics we use to assess the results of our experiments.

5.1 Testbed Setup

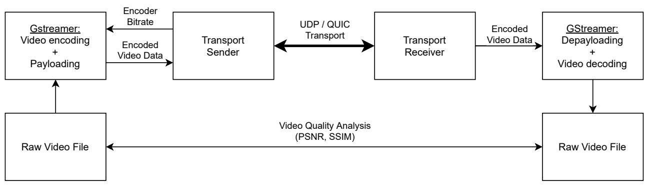

Figure 3: Testbed setup for automated evaluation of RTP over QUIC implementations

The testbed is shown in figure 3. We run our Sender and Receiver on the same physical host. To simulate the network conditions, each peer runs in a docker container with a separate virtual network interface. The host machine automatically applies different network conditions to both virtual network interfaces using Linux `tc`. Since there are no network elements between the peers, the round-trip delay is set as the sum of the delay applied to both interfaces.

We run all implementations twice to test the effects of different kinds of traffic on the same connection. Once only sending real-time traffic and once sending real-time traffic competing with a QUIC stream constantly sending data on the same connection.

There are two aspects to congestion control, real-time media or not: on the one hand, how an algorithm impacts the performance of a given data stream itself and, on the other hand, how fair the algorithm is against other connections (using the same or different congestion control). In this paper, we focus on the former one as we seek to understand the use of control loops at different layers and the feature interactions if multiple such loops exist. We leave the latter part for future work.

5.2 Test Parameters

The network parameters are configured based on the test cases defined in RFC8867 [24] and RFC8868 [26]. The link between the peers is limited to 1 Mbit/s using a Token Bucket Filter with a queueing latency of up to 400ms on each interface, i.e., 50 KB at 1 Mbit/s. To simulate a variable network capacity, we reduce the capacity to 500 kbit/s after 30 seconds. After another 30 seconds, we set the bandwidth back up to 1 Mbit/s until the end of the test. Each experiment is run with four different one-way propagation delay settings: 1, 50, 150, and 300ms.

5.3 Metrics

To evaluate our measurements, we use mainly the two metrics described in this section. The first metric evaluates the quality of the video received by the receiver compared to the original video file, and the second metric assesses the behavior of the congestion control in use.

SSIM. We use the structural similarity index (SSIM) [27] to evaluate the perceived quality of the received video stream. SSIM compares the original raw video file with the resulting raw video file, and we look at the average SSIM over all frames and the time series over all frames. [2]

Target Bitrate. To compare the performance of SCReAM and our trivial bitrate adaption algorithm in combination with or without QUIC's built-in congestion control, we analyze the target bitrate, which is output by the algorithms and signaled to the encoder.

6 EVALUATION

We organize discussing the results of our experiments following the different combinations of congestion controllers presented in section 3.3 used on the RTP over QUIC implementation. Results for the UDP reference implementation are not explicitly discussed, but mentioned where we compare QUIC and UDP. Since the one-way propagation delay makes no big difference for most experiments, we only look at the experiments using 50ms, except where otherwise noted.

6.1 Only Trivial Rate Control

The first experiments disabled QUIC's NewReno and only used our trivial rate adaption algorithm. Since QUIC does not perform any congestion control, the rate adaption algorithm sees an almost empty queue of outgoing RTP packets most of the time. It thus uses the highest bitrate option of 1280 kbit/s, causing very high packet loss, which results in a low average SSIM of 0.78. Since the algorithm does not detect the congestion of the link, the algorithm behaves very similarly when a QUIC stream is opened in parallel to send competing data. Due to more packet loss, the average SSIM decreases to 0.75.

6.2 Trivial Rate Control and NewReno

When using NewReno with our trivial rate controller, the average SSIM increases to about 0.78. The bitrate alternates between the highest and lowest level very fast as shown in 4. The NewReno congestion control causes the RTP queue to grow at high bitrates, leading to bitrate decreases. At low bitrates, which do not use the full capacity of the link, the queue of outgoing RTP packets shrinks, which then causes the bitrate to increase again. With a second stream sending data next to the RTP datagrams, the RTP queue never shrinks enough for the bitrate adaption algorithm to increase

[2]We also calculated the peak-signal-to-noise ratio (PSNR), another metric to assess the quality of a received video by comparing it with the source file. We decided only to include SSIM in this paper since it produces more significant differences between the experiments.

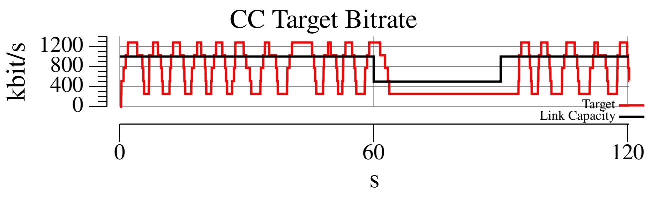

Figure 4: Target bitrate of our trivial rate controller used to configure the encoder when running on a QUIC connection with NewReno.

the target bitrate. Thus, the target bitrate stays at the lowest level most of the time, which, combined with some packet loss, leads to a very low average SSIM of 0.69.

6.3 Only SCReAM

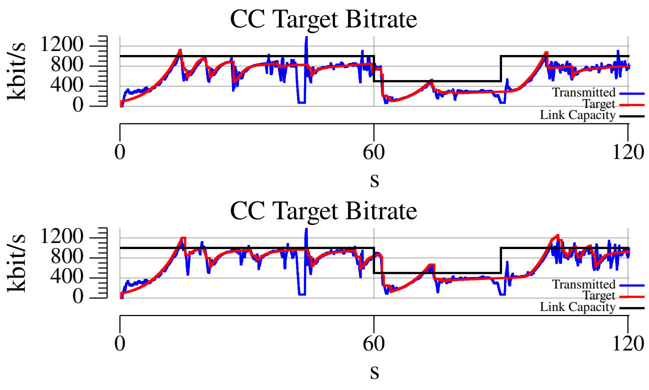

Figure 5: SCReAM target bitrate and rate transmitted using only SCReAM on a real-time media flow without any background traffic on QUIC (top) and UDP (bottom).

Next, we look at the SCReAM algorithm on the outgoing RTP traffic. In this scenario, RTP over QUIC reaches a very similar performance as the UDP reference. In both cases, RTP throughput comes close to the maximum bandwidth of 1 Mbit/s after the initial ramp-up phase. When the link capacity reduces, the target bitrate first drops to almost zero before starting a new ramp-up phase and reaching the maximum capacity. SCReAM then tries to keep the bitrate stable, as frequent changes can lead to a worse QoE, which leads to the delay between the increase of the link's capacity at 90 seconds and the next ramp up phase. The UDP implementation reaches an average SSIM of 0.8, while the RTP over QUIC implementation even reaches an average of 0.87. Figure 5 shows the target bitrate. Compared to UDP, the target bitrate over QUIC slightly lower, which is caused by a larger packet header overhead, but in general, this shows, that QUIC datagrams can be used as an alternative to RTP over plain UDP.

When introducing a parallel QUIC stream to send data concurrently to the RTP stream, we have a prioritization problem. The QUIC implementation has to decide how much bandwidth to allocate to the stream or the datagrams. Since our test constantly sends data on the stream, while the RTP packets vary in size, the non-RTP data streams received a significantly larger share of the bandwidth. This leads to SCReAM reducing the target bitrate even further which drops to zero almost immediately.

It might even be better for the RTP stream, to disable the real-time congestion controller, to receive a bigger share of the available bandwidth. However, this is a very artificial setup because, in real-world configurations, the background traffic is more likely on/off traffic, such as occasional HTTP request/response bursts of variable sizes instead of a constant data stream.

In future experiments, we plan to evaluate how the bandwidth would be shared between a more natural form of background traffic and whether a higher prioritization of datagrams could improve this scenario.

6.4 SCReAM and NewReno

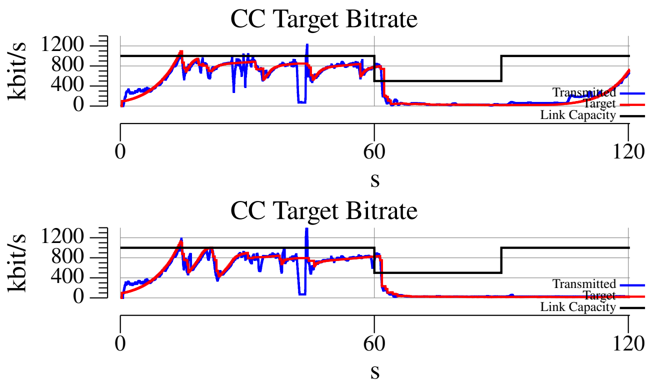

Figure 6: SCReAM target bitrate and rate transmitted using SCReAM and NewReno on a real-time media flow with one-way propagation delays of 1ms (top) and 50ms (bottom).

Now we enable QUIC's NewReno on all outgoing traffic, together with SCReAM on the RTP transmission. The sender initially behaves similarly to the previous case. Since SCReAM does not try to find the link capacity as aggressively as NewReno might try, the datagrams are sent too slow for QUIC even to try to increase its congestion window, leading to an underutilization of the link. A problem with this combination appears when the link capacity reduces. The combination of SCReAM and NewReno reduces the target bitrate to zero because both algorithms react to the reduced capacity. This is shown in figure 6. After the link capacity recovers, the target bitrate takes very long to recover because the ramp up happens much later than after the initial startup if it happens at all. This also has an impact on the video quality, which results in an average SSIM of 0.78.

When sending data on the parallel QUIC stream, the setup behaves similar to the one without NewReno in the previous section. The target bitrate drops to zero very fast due to the missing prioritization between real-time and non-real-time data.

6.5 SCReAM without RTCP

In this section, we look at the potential RTCP overhead reduction reached by the techniques described in 3.4. Since this scenario does not change much compared to the results of the previous subsections, we ignore the data transmission on parallel streams for now.

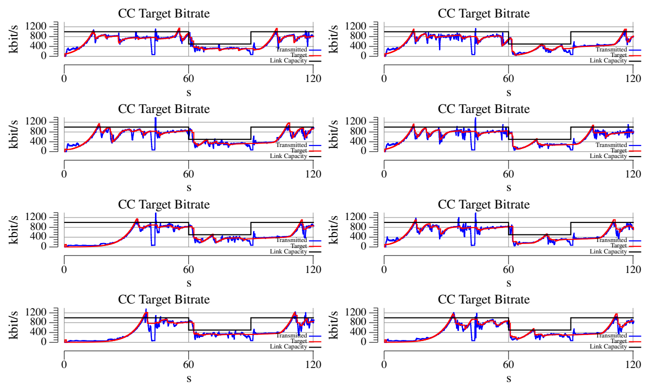

Figure 7: SCReAM target bitrate and rate transmitted using SCReAM with only feedback gathered from QUIC connection statistics (left) and RTCP feedback (right) at different one-way delay settings from top to bottom: 1ms, 50ms, 150ms, 300ms.

Figure 7 shows the target bitrate of video transmission using only the SCReAM congestion controller with and without RTCP feedback at different one-way delays. The implementation generates all required information as described in 3.4. To calculate the arrival time of RTP packets, we tried to use the `smoothed_rtt` and the `latest_rtt` sample. We found that using the `latest_rtt` generally yields better results, and thus, we only consider those results in our evaluation. Here we observe that the generated feedback can be used to calculate a target bitrate close to the target bitrate calculated using the RTCP feedback. However, the ramp-up phase takes a very long time to reach a good link utilization in some cases. In experiments with a shorter one-way propagation delay, SCReAM reaches a better link utilization, which is also reflected in the video quality. The two lower link delays lead to SSIM values equal or even higher than in the experiment using RTCP feedback. The two larger delays lead to slightly lower SSIM values of 0.85 and 0.83. This indicates that more fine-grained connection statistics may be required to replace the RTCP feedback.

The RTT measurements provided by QUIC may be inaccurate when acknowledgments are delayed on the return path, which is unlikely in our artificial test environment. However, a QUIC endpoint is allowed to delay and aggregate acknowledgments of multiple packets, and it will inform the sender about the delay since the last received packet. This may lead to situations in which the generated feedback does not include an arrival timestamp for the latest RTP packets, even though they have already arrived at the receiver. This feedback report is likely different from what the RTCP feedback would provide.

Alternatively to the RTT, precise one-way delay measurements may provide more accurate information of the characteristics of the path. A method to explicitly measure the one-way delay was proposed in[7]. The Acknowledgement Frequency draft[9] could provide another optimization. The draft allows a sender to control how frequent acknowledgments are sent by the receiver, improving the timing of acknowledgments and feedback generation.

6.6 SCReAM and NewReno without RTCP

In sections 6.4 and 6.5, we described the experiments using SCReAM without RTCP and SCReAM combined with NewReno. In both setups, we saw issues that all come together when using SCReAM without RTCP and combined with NewReno at the same time. The results are thus very unreliable. In some cases, it leads to a link utilization similar to the ones in the previous sections, in others, the target bitrate drops to almost zero without ever ramping up.

7 DISCUSSION AND CONCLUSION

In this paper, we have presented an implementation of RTP over QUIC. The focus of our work is on different schemes of congestion control for real-time media over QUIC. We have seen that it is possible to stream video in real-time over QUIC, using RTP encapsulated in QUIC datagrams with an appropriate congestion controller. While relying on QUIC's default NewReno algorithm combined with a trivial encoder rate adaption is possible, we achieved much better quality using a real-time congestion control algorithm like SCReAM. When we combined NewReno with a real-time congestion controller, they initially did not seem to interfere with each other significantly. However, as soon as the network conditions are bad enough for both algorithms to react to it, the link utilization gets much worse than when only one controller is enabled.

One issue we encountered when using RTP over QUIC appeared when non-real-time data was sent simultaneously on the same flow using QUIC streams. This issue was not caused by congestion control but by the lack of prioritization of datagrams and streams in our implementation.

We have shown that by leveraging the QUIC connection state, mainly datagram acknowledgments and RTT estimations, it is possible to generate feedback reports similar to RTCP, however, it does not provide the same quality. The quality might be improved in future experiments using explicit one-way delay measurements and letting the sender control the acknowledgment frequency of the receiver.

Future work needs to be done to evaluate other possible congestion control configurations and adding relative priorities to QUIC datagrams and streams. We have not yet looked at a setup to use two independent congestion controllers, one for real-time traffic only and the other one for non-real-time traffic only. The results of this setup will likely depend on the prioritization issue we already encountered in the designs we evaluated in this work. Other setups which need further experimentation involve other topologies such as multiple independent connections sharing a common bottleneck. The competing connections could be other RTP over QUIC streams using the same congestion control or other traffic types, e.g., TCP using different congestion control algorithms. Moreover, one could develop more sophisticated improvements to the input to the congestion control algorithms by combining observations from the real-time and non-real-time streams. E.g., one could try to use data that does not have any real-time requirements sent on streams to probe for higher bandwidths. Packet loss, in this case, would not be an issue, as retransmissions can be done at any time.

REFERENCES

[1] 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

[2] Carsten Burmeister, Jose Rey, Noriyuki Sato, Joerg Ott, and Stephan Wenger. 2006. Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF). RFC 4585. https://doi.org/10.17487/RFC4585

[3] Ramon Caceres, Alan Clark, and Timur Friedman. 2003. RTP Control Protocol Extended Reports (RTCP XR). RFC 3611. https://doi.org/10.17487/RFC3611

[4] Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio Mascolo. 2016. Analysis and Design of the Google Congestion Control for Web Real-Time Communication (WebRTC). In *Proceedings of the 7th International Conference on Multimedia Systems* (Klagenfurt, Austria) *(MMSys '16)*. Association for Computing Machinery, New York, NY, USA, Article 13, 12 pages. https://doi.org/10.1145/2910017.2910605

[5] GStreamer Team. 2021. *GStreamer: open source multimedia framework*. Retrieved September 24, 2021 from https://gstreamer.freedesktop.org/

[6] Stefan Holmer, Henrik Lundin, Gaetano Carlucci, Luca De Cicco, and Saverio Mascolo. 2016. *A Google Congestion Control Algorithm for Real-Time Communication*. Internet-Draft draft-ietf-rmcat-gcc-02. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-rmcat-gcc-02 Work in Progress.

[7] Christian Huitema. 2021. *Quic Timestamps For Measuring One-Way Delays*. Internet-Draft draft-huitema-quic-ts-06. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-huitema-quic-ts-06 Work in Progress.

[8] Sam Hurst. 2021. *QRT: QUIC RTP Tunnelling*. Internet-Draft draft-hurst-quic-rtp-tunnelling-01. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-hurst-quic-rtp-tunnelling-01 Work in Progress.

[9] Jana Iyengar and Ian Swett. 2021. *QUIC Acknowledgement Frequency*. Internet-Draft draft-ietf-quic-ack-frequency-00. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-quic-ack-frequency-00 Work in Progress.

[10] Jana Iyengar and Ian Swett. 2021. QUIC Loss Detection and Congestion Control. RFC 9002. https://doi.org/10.17487/RFC9002

[11] Jana Iyengar and Martin Thomson. 2021. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000. https://doi.org/10.17487/RFC9000

[12] Ingemar Johansson. 2014. Self-Clocked Rate Adaptation for Conversational Video in LTE. In *Proceedings of the 2014 ACM SIGCOMM Workshop on Capacity Sharing Workshop* (Chicago, Illinois, USA) *(CSWS '14)*. Association for Computing Machinery, New York, NY, USA, 51–56. https://doi.org/10.1145/2630088.2631976

[13] Ingemar Johansson and Zaheduzzaman Sarker. 2017. Self-Clocked Rate Adaptation for Multimedia. RFC 8298. https://doi.org/10.17487/RFC8298

[14] Adam H. Li. 2007. RTP Payload Format for Generic Forward Error Correction. RFC 5109. https://doi.org/10.17487/RFC5109

[15] Abhijit Mondal and Sandip Chakraborty. 2020. Does QUIC Suit Well With Modern Adaptive Bitrate Streaming Techniques? *IEEE Networking Letters* 2, 2 (2020), 85–89. https://doi.org/10.1109/LNET.2020.2991867

[16] Minh Nguyen, Hadi Amirpour, Christian Timmerer, and Hermann Hellwagner. 2020. Scalable High Efficiency Video Coding Based HTTP Adaptive Streaming over QUIC. 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, 28–34. https://doi.org/10.1145/3405796.3405829

[17] Joerg Ott and Mathis Engelbart. 2021. *RTP over QUIC*. Internet-Draft draft-engelbart-rtp-over-quic-01. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-engelbart-rtp-over-quic-01 Work in Progress.

[18] Joerg Ott, Roni Even, Colin Perkins, and Varun Singh. 2017. *RTP over QUIC*. Internet-Draft draft-rtpfolks-quic-rtp-over-quic-01. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-rtpfolks-quic-rtp-over-quic-01 Work in Progress.

[19] Tommy Pauly, Eric Kinnear, and David Schinazi. 2021. *An Unreliable Datagram Extension to QUIC*. Internet-Draft draft-ietf-quic-datagram-04. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-quic-datagram-04 Work in Progress.

[20] Colin Perkins and Jörg Ott. 2018. Real-Time Audio-Visual Media Transport over QUIC. In *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC* (Heraklion, Greece) *(EPIQ'18)*. Association for Computing Machinery, New York, NY, USA, 36–42. https://doi.org/10.1145/3284850.3284856

[21] Colin Perkins and Varun Singh. 2017. Multimedia Congestion Control: Circuit Breakers for Unicast RTP Sessions. RFC 8083. https://doi.org/10.17487/RFC8083

[22] quic-go authors and Google, Inc. 2021. *A QUIC implementation in pure Go*. Retrieved September 24, 2021 from https://github.com/lucas-clemente/quic-go

[23] Zaheduzzaman Sarker, Colin Perkins, Varun Singh, and Michael A. Ramalho. 2021. RTP Control Protocol (RTCP) Feedback for Congestion Control. RFC 8888. https://doi.org/10.17487/RFC8888

[24] Zaheduzzaman Sarker, Varun Singh, Xiaoqing Zhu, and Michael A. Ramalho. 2021. Test Cases for Evaluating Congestion Control for Interactive Real-Time Media. RFC 8867. https://doi.org/10.17487/RFC8867

[25] Henning Schulzrinne, Stephen L. Casner, Ron Frederick, and Van Jacobson. 2003. RTP: A Transport Protocol for Real-Time Applications. RFC 3550. https://doi.org/10.17487/RFC3550

[26] Varun Singh, Joerg Ott, and Stefan Holmer. 2021. Evaluating Congestion Control for Interactive Real-Time Media. RFC 8868. https://doi.org/10.17487/RFC8868

[27] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing* 13, 4 (2004), 600–612. https://doi.org/10.1109/TIP.2003.819861

[28] Xiaoqing Zhu, Rong Pan *, Michael A. Ramalho, and Sergio Mena de la Cruz. 2020. Network-Assisted Dynamic Adaptation (NADA): A Unified Congestion Control Scheme for Real-Time Media. RFC 8698. https://doi.org/10.17487/RFC8698