University of Glasgow | School of Computing Science

Optimising QUIC for use with Real-Time Multimedia Traffic

Vivian Band (2038561)

April 22, 2019

ABSTRACT

The vast majority of IP traffic (80%) is multimedia traffic. Teleconferencing and online gaming are commonly used applications on the modern Internet; these need to appear to respond in real-time to user input and require low latency to achieve this.

QUIC is seeing more adoption as a transport protocol for client-server applications, accounting for a signficant portion of Google's outbound traffic and making up 7% of total Internet traffic as of 2017. However, it is not currently suitable for real-time traffic due to high latency caused by guaranteed reliability and in-order delivery.

QUIC is a userspace protocol; these can be modified more easily than kernel-level implementations of transport protocols, like TCP. This flexibility allows QUIC to be significantly adapted, allowing the protocol to be used for real-time multimedia applications.

The modifications made in this project allow QUIC to achieve low-latency by allowing some packet loss, subject to playback deadlines, in order to improve the throughput of data to the receiver application. Video playback quality is further improved by taking advantage of existing retransmission behaviours in QUIC, allowing the retransmission of frames which will still be useful when they reach the receiver as well as any dependent frames required for live ones.

Our solution shows that partial reliability through the use of deadline-aware retransmissions minimises the occurrence of stalls by improving the throughput of useful ('live') multimedia data. This use of selective retransmissions is particularly important in loss-prone environments (e.g. wireless links).

1. INTRODUCTION

In this paper, we ...

We structure the remainder of this paper as follows.

2. BACKGROUND

2.1 Motivation

2.2 Related Work

3. DESIGN

4. RESULTS

Each test was performed using a simulated network on mininet, consisting of a server node connected to a client node over a single link with a specifically defined latency. This link latency was varied between 50ms, 100ms, and 150ms, and loss rates of 0%, 0.01%, 0.1%, and 1% were introduced for tests on each latency. Each test ran for 300 seconds with client and server both operating at a framerate of 60 fps; an I-frame was sent every 10th frame, consisting of 4 QUIC packets to represent the fact that I-frames are too large to be sent as a single QUIC payload.

Tests for guaranteed reliability used a QUIC stack with standard retransmission behaviours, while tests for partial reliability used the selective retransmission mechanisms described above [TODO: write design section].

4.1 Differences in Stack Latency

Stack latencies remain consistently within 15ms of the path latency in the partially reliable implementation for all loss rates (figure 1), but have slightly more variance than stack latencies in the guaranteed reliability implementation in equivalent conditions. This is due to the extra checks which are performed by the server after sending a packet: each item in the retransmission buffer for the relevant stream is checked to see if it has become stale and needs to be removed.

When loss is introduced on the link, the guaranteed reliability implementation begins to show an increase in the number of higher stack latencies 2. QUIC determines a packet is lost when it receives an acknowledgement for a higher packet number; this requires a minimum of the time taken to send the packet over the link, plus a multiple of the total round trip time depending on how many times the packet in question was lost in transit. This causes the latency data to gather in bands: the latencies of lost packets will be multiples of the total round-trip time on the link. In contrast, the partially reliable implementation maintains similar performance all rates of loss; this is due to the server removing packets which will not be useful to the client upon arrival from its retransmit buffer without receiving an acknowledgement.

4.2 Differences in Application Latency

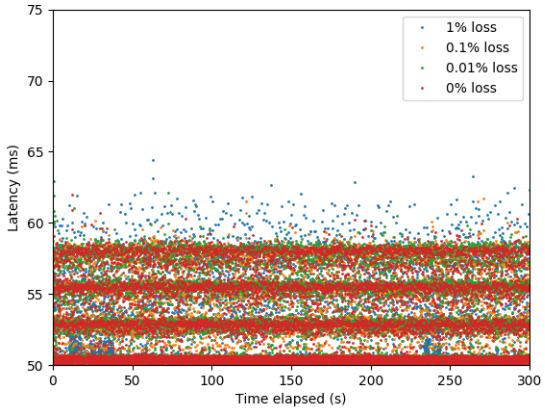
Partially reliable QUIC achieves consistently lower application latencies for all link latencies compared to standard QUIC with guaranteed reliability (figure 3). Application latencies for partially reliable QUIC have a greater range than stack latencies due to the frame buffer: if there is a gap in received packets, the stack will wait for the missing packet; if the packet doesn't arrive before its predicted playback deadline expires, the read offsets for the stream are advanced and a sequence of video frames with contiguous RTP timestamps is passed to the application (this may only be a single frame). This buffer was intended to be set as 4 frames, corresponding to an additional latency of 66ms at a 60fps playback rate, but was incorrectly set at a single

frame during the 50ms and 100ms tests. A single frame at 60fps is equivalent to 16.7ms; this is added to application latency in the event of loss, with decreasing latencies following as subsequent blocks of received frames are passed to the application. In the case of 150ms tests, the buffer limit was set to the intended 4 frames; application latencies can be seen in bands at multiples of 16.7ms to around a maximum of 66.6ms greater than the link latency as a result.

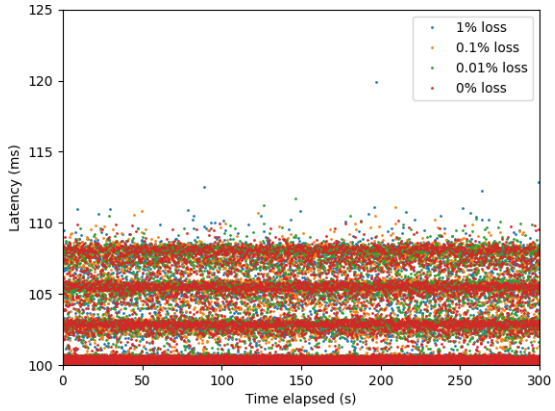
The distinctive four-band pattern present in all instances of partially reliable QUIC is due to I-frames being sent as 4 QUIC packets without a 16.7ms delay between them; four packets must be delivered to the application within the same call, but iterating through the reorder buffer and removing relevant items takes time to execute.

The bands in the guaranteed reliability application latencies are caused by video frames being sent by the server at 16.7ms intervals; the latency of each frame is increased by a multiple of this until the missing packet is received at the client, even though all of these frames are delivered to the application within the same function call. The occasional spikes in 1% loss test are caused by a packet being lost twice; these occur at twice the round-trip time, plus two frame playback times at 60fps, beyond the initial link latency.

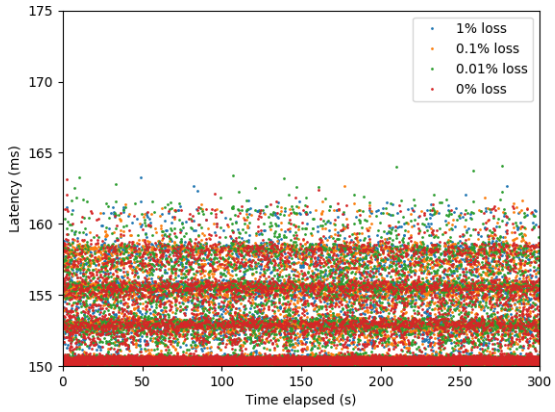
This head-of-line blocking behaviour associated with guaranteed reliability does not affect 0% loss links, where standard QUIC achieves consistently lower application latencies than partially reliable QUIC, and has minimal impact on average application latency in 0.01% loss links. However, for 0.1% and 1% loss scenarios, partially reliable QUIC achieves consistently lower application latencies: the latencies for fully reliable QUIC in these circumstances are frequently a round-trip time higher than the link latency, compared to a maximum of 66.6ms when a 4-frame buffer is set.



(a) Partial reliability, 50ms

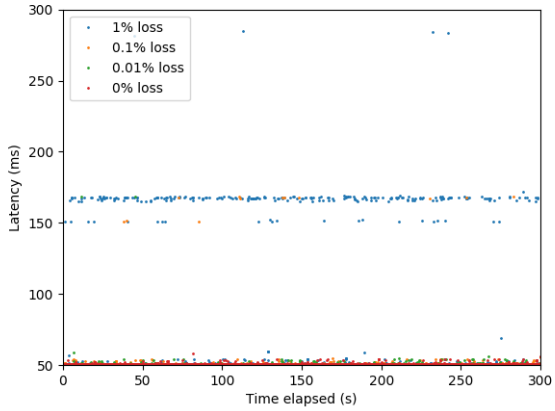


(b) Partial reliability, 100ms

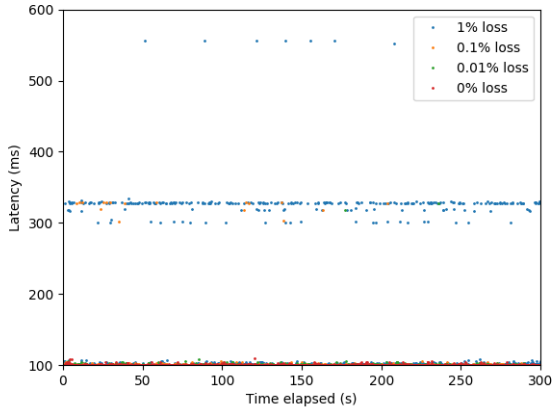


(c) Partial reliability, 150ms

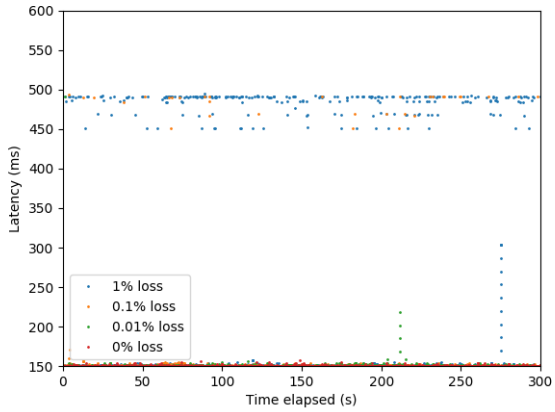
Figure 1: Latency between server and client stack (partial reliability)



(a) Guaranteed reliability, 50ms

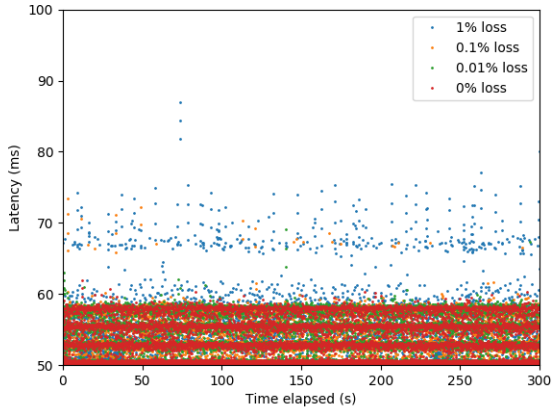


(b) Guaranteed reliability, 100ms

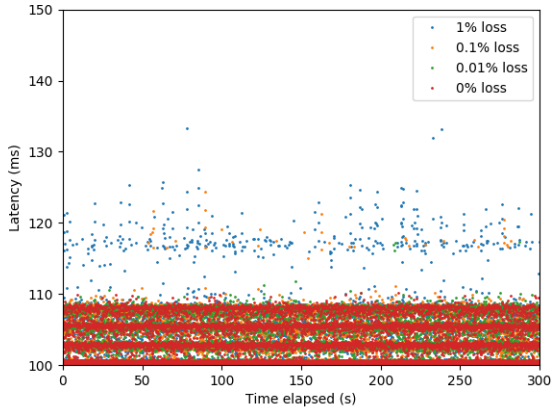


(c) Guaranteed reliability, 150ms

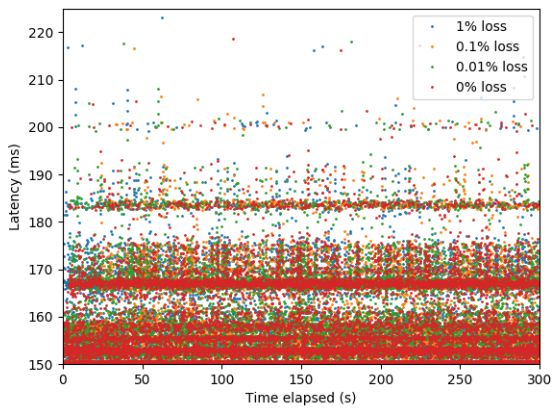
Figure 2: Latency between server and client stack (guaranteed reliability)



(a) Partial reliability, 50ms

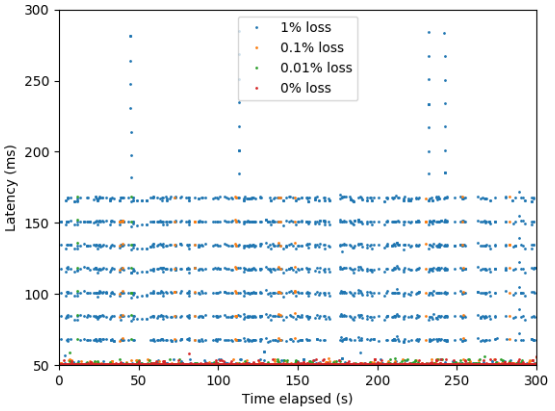


(b) Partial reliability, 100ms

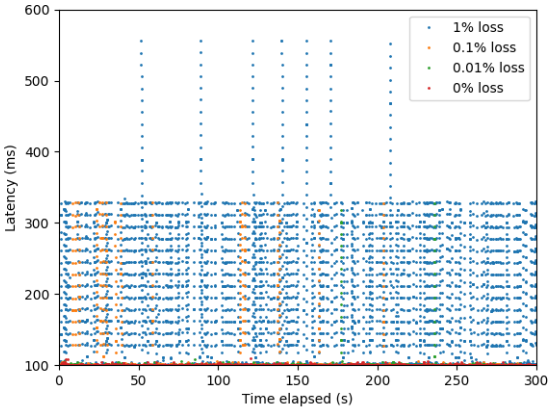


(c) Partial reliability, 150ms

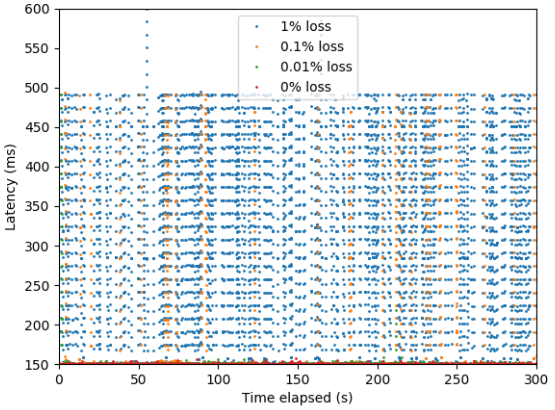
Figure 3: Latency between server and client application (partial reliability)



(a) Guaranteed reliability, 50ms



(b) Guaranteed reliability, 100ms



(c) Guaranteed reliability, 150ms

Figure 4: Latency between server and client application (guaranteed reliability)

4.3 Differences in Playback Time

If a packet has been lost in transit, the head-of-line blocking behaviours within standard QUIC will delay delivering all subsequent data until this missing packet has been received. If there is no video data held in the application to play during this delay (i.e. buffered data), the video will appear to stall. The exact delay caused by a packet lost once is proportional to the path latency: figure 5a shows a delay of 100ms, the round trip time of a 50ms link. Figure single-100 shows a delay of 216ms on a 100ms link, equivalent to a round trip time plus one frame playback time; this extra 16ms is likely due to these measurements being obtained in terms of RTP playback timestamps, with a granularity of 16ms per increment at 60fps. The delay for a loss on a 150ms link shown in figure 5c is 333ms, equivalent to one round-trip time plus one frame playback time.

As the rate of loss on a link increases, the number of stalls also increases. Table 1 shows the number of stalls which occured in the 100ms tests; naturally, the number of stalls between all latencies remain similar for each loss rate, but the difference between intended playback time and actual playback time for each subsequent video frame becomes larger at higher latencies (table 2). The graphs shown in figure 6 illustrate the magnitude of these differences.

Loss Rate	Stalls	Frames Played
0%	0	100.0%
0.01%	2	100.0%
0.1%	12	100.0%
1%	198	100.0%

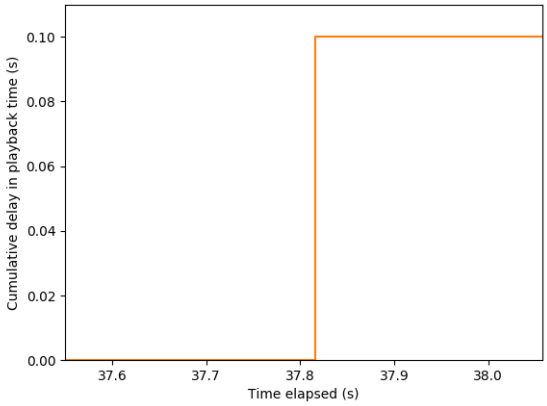
Table 1: Number of stalls and percentage of frames played for guaranteed reliability, 100ms

Loss Rate	Latency (ms)	Discrepancy (s)	Total latency (s)
0.01%	50	0.2	50.2
0.1%	50	1.2	50.12
1%	50	19.8	51.98
0.01%	100	0.43	100.43
0.1%	100	2.6	102.6
1%	100	42.8	142.8
0.01%	150	0.66	150.66
0.1%	150	4.0	150.4
1%	150	66.0	216.0

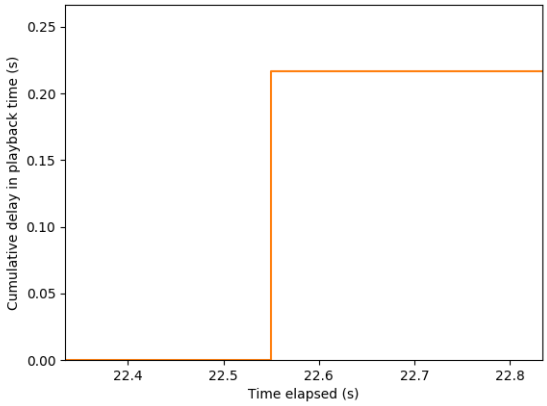
Table 2: Playback discrepancy and total latency for a live-recorded frame to be played by the client application at the end of each test run for each loss rate and latency for guaranteed reliability

Table 2 shows the cumulative discrepancy between intended playback time and actual playback time, as well as the delay between a live-recorded frame being sent and being played by the client application by the end of a 300 second test run. The increase in this delay over time is illustrated in figure 7.

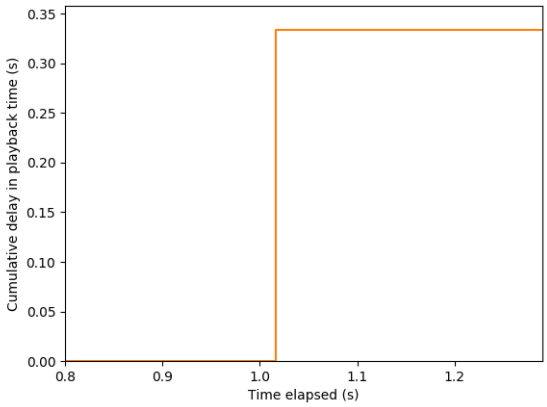
Although 100% of the video frames in the media content being sent are delivered to the client application with guar-



(a) Single instance playback delay, 50ms

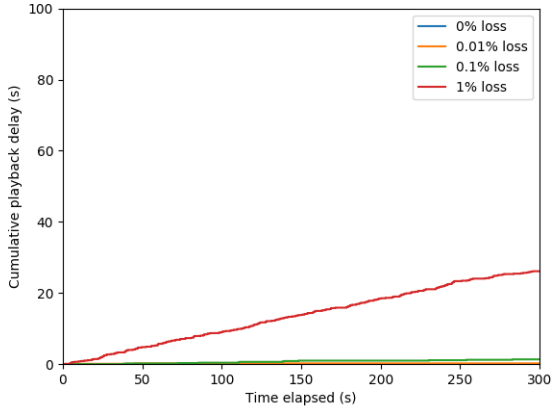


(b) Single instance playback delay, 100ms

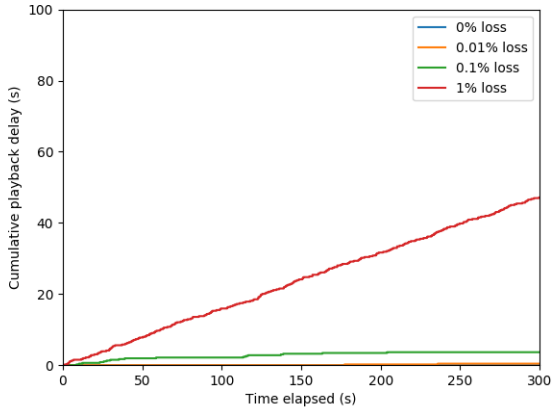


(c) Single instance playback delay, 150ms

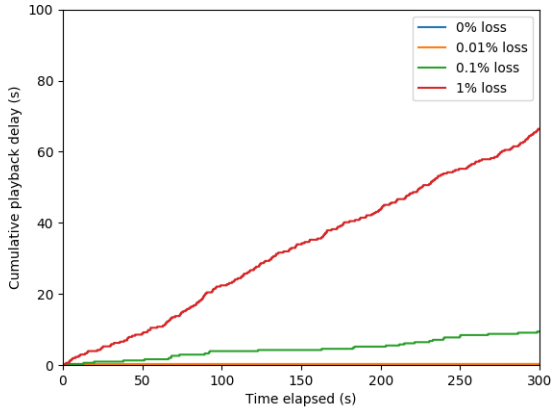
Figure 5: Increase in cumulative playback delay as a result of a single lost packet on various links (guaranteed reliability)



(a) Playback offsets, 50ms

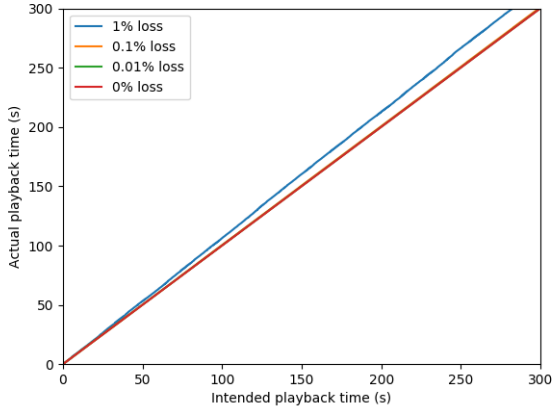


(b) Playback offsets, 100ms

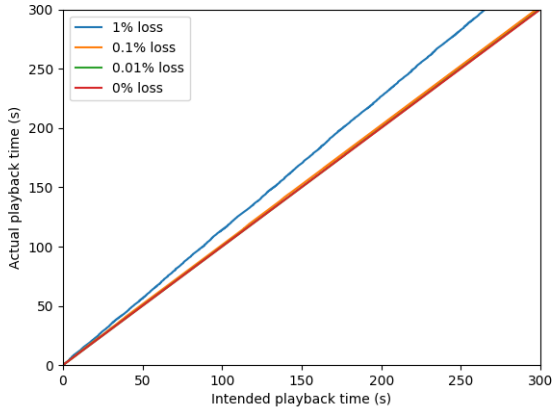


(c) Playback offsets, 150ms

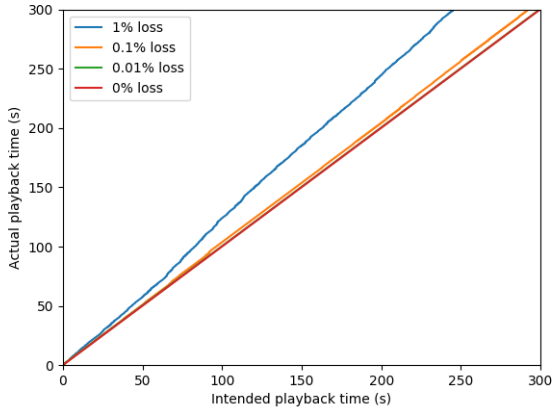
Figure 6: Differences between when a frame is played out compared to the intended playback time (guaranteed reliability)



(a) Playback times, 50ms



(b) Playback times, 100ms



(c) Playback times, 150ms

Figure 7: Differences between when a frame is played out compared to when the frame was sent (guaranteed reliability)

anteed reliability behaviours, there is an increasing latency between sending a frame and playing it at the client; the longer an application is running, the greater this playback discrepancy becomes.

The partially reliable implementation of QUIC created for this project, however, experiences no stalls due to incrementing stream read offsets in response to playback deadlines: if there is a gap in the data received on a given stream and there is data approaching its playback deadline held in the reorder buffer, the modified implementation will skip ahead to the live data and will not wait for the preceding gap to be filled. This allows the client to play received content without additional delays caused by head-of-line blocking. The percentage of 'useful frames' in table 3 is calculated as the number of complete I-frames and P-frames with an associated complete I-frame, compared to the total number of unique frames sent by the server; incomplete I-frames and their subsequent P-frames are not useful for playback.

Loss Rate	Stalls	Useful frames
0%	0	100.0%
0.01%	0	100.0%
0.1%	0	99.644%
1%	0	95.472%

Table 3: Number of stalls and percentage of useful frames played for partial reliability

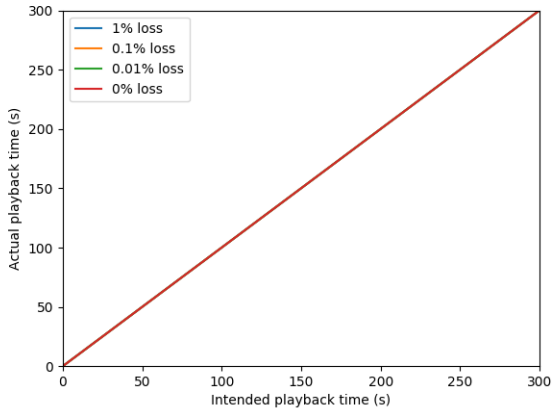


Figure 8: Differences between when a frame is played out compared to when the frame was sent (partial reliability, all loss rates)

Fewer frames are played back at the client, which will result in occasional glitches in playback. Given that 1 in every 10 frames sent is an I-frame consisting of 4 QUIC packets, there is a 70% chance that a dropped packet contains a P-frame; at a playback rate of 60fps, this would result in an absence of updated content for 16.7ms. If a dropped packet contains a section of an I-frame, this will affect the subsequent 9 P-frames which are dependent on it; this would result in no updated content for 167ms.

For a test run of 300 seconds, 18,000 video frames are sent. On average 180 of these are dropped in the worst-case 1% loss scenario; if 30% of these are I-frames and 70% are P-frames, 11.01 seconds of playback in total are subject to glitches across all latencies. This is higher than the total playback offset in a 50ms connection subject to 1% loss for guaranteed reliability (table 2), but the crucial difference between these two outcomes is that the content received after a loss in the partially reliable implementation is played without any cumulative delay (figure 8): some playback quality is sacrificed for maintaining a consistent low latency to allow for the level of responsiveness required by real-time applications.

An application using guaranteed reliability could discard stale frames and continue playback from the frame with the closest RTP timestamp to the current playback deadline to reduce this cumulative delay, but it would play fewer frames than the partially reliable implementation: the minimum and maximum absences of updated content for partially reliable QUIC for all link latencies are 16.7ms and 167ms respectively, while delays in the datasets gathered for fully reliable QUIC range between 100ms and 333ms, depending on the link latency. This shows that partially reliable QUIC results in improved throughput of useful data compared to attempting to optimise applications on top of standard QUIC for real-time behaviour.

5. RELATED WORK

6. FUTURE WORK

6.1 Congestion Control

Removing an item from the retransmission buffer in the `ngtcp2` implementation of QUIC is complex. The function call used to receive an acknowledgement contains numerous updates to congestion control statistics; simply attempting to remove the item from the retransmit buffer by freeing the associated memory causes a range of errors due to failed `assert` checks elsewhere in the stack. As a result, falsified acknowledgements were created at the server to remove stale entries which had not received an acknowledgement from the client. This is problematic in that it causes the client to believe there is no congestion on the network, and this QUIC flow becomes overly aggressive compared to other flows sharing the link as a result. This did not cause problems with testing on the simple one-to-one topology with a known round-trip time and no co-existing flows described in section 4, but the congestion control statistics must be more carefully adjusted in future iterations to allow real-time QUIC flows to be fair to other traffic on shared links.

6.2 Improved I-frame Handling

QUIC packets containing fragments of I-frames which will not arrive in time to be played back themselves, but are required for live P-frames are retransmitted by the server. An implementation of real-time QUIC which immediately passes data to the application upon arrival would deliver these 'stale' fragments to the application, however, the implementation of partial reliability which delivers video frames in order does not; the stream read offsets at the client are incremented upon the delivery of live data and the detection of stale data, so these I-frame fragments are never passed to the application.

Future refinements of this implementation would include mechanisms to allow the client to determine if an incoming packet contains an I-frame based on its associated RTP timestamp, and introducing additional read offsets within a stream to allow required stale I-frames to be read without also reading stale P-frame data in the process. This would allow complete frames to be delivered in-order to the application, while continuing to drop stale P-frames and stale I-frames with no live dependencies to minimise latency. I-frame payloads would also need to be read as a complete block to allow delivery to the application as a complete frame, as opposed to the current method of being delivered as several separate reads.

6.3 Multiple Streams

Real-time QUIC will allow multimedia application developers to use multiplexed streams to deliver data to an application concurrently, but this significantly increases the difficulty in developing these applications: the content received from each stream needs to be co-ordinated in order to be used by the application correctly. The tests in this paper were performed using a single stream; further experiments will focus on how to co-ordinate multiplexed streams to optimise the quality of video playback.

7. CONCLUSIONS

Passing QUIC payloads (i.e. RTP packets) to the application as soon as they arrive avoids adding even more complexity to an already extensive transport protocol, but it requires application developers to implement mechanisms to reorder incoming frames, assemble I-frames, and deal with frames which are corrupted, incomplete, or missing dependencies; the transport protocol effectively becomes RTP with selective retransmissions. This allows developers flexibility in exactly how a given application should behave while improving the amount of useful data received, but it increases both the entry barrier to creating real-time multimedia applications and the difficulty in maintaining them.

Delivering complete video frames in-order while allowing gaps, as the implementation of real-time QUIC in this project aims to do, removes the responsibility of frame reordering and reconstruction from application developers, but increases the complexity of QUIC while also restricting the applications which can be developed: novel applications, such as multiplayer gaming over QUIC, may require many different types of messages other than I-frames and P-frames. Awareness of these messages would need to be added within the QUIC protocol in order for a receiver to be able to parse stream information as a complete message, and for the sender to be able to establish more complex dependencies and retransmission behaviours; for example, an important message such as a character receiving an item should be transmitted with guaranteed reliability, while movement would use partial reliability in a similar manner to P-frames. This is simply not feasible given the wide range of behaviours and transmittable information that real-time applications could have.

The development of video-based real-time applications which rely on a limited number of message types (I-frames and P-frames) can be simplified by making QUIC aware of how to parse these messages and pass them to the application as complete frames in sequential order; this project has been successful in achieving improved real-time video playback performance using this approach. However, the increasing popularity of real-time applications with variable information and behaviour, such as augmented reality, virtual reality, and multiplayer gaming, suggests that passing data to the application as soon as it arrives and embedding message dependency information in payloads is the better approach to take for a generalised real-time implementation of QUIC. This prevents ossification in terms of which applications which can use real-time QUIC and also avoids adding an unrealistic level of complexity to QUIC implementations. Creating a new partially reliable QUIC stream type for use alongside guaranteed reliability streams would ensure that key information reaches the receiver while allowing low-latency for other content.