University of Glasgow | School of Computing Science

QUICsilver: 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

According to *The Zettabyte Era* white paper by Cisco [2], the percentage of Internet Protocol (IP)[1] traffic carrying multimedia data will increase from 73% of all IP traffic in 2016 to 83% by 2021. Live video and online gaming are commonly used applications on the modern Internet and are predicted to form 13% and 4% of all IP traffic by 2021 respectively; these applications have strict latency bounds in order to appear to respond in real-time to user input.

QUIC is a modern, fully-reliable userspace transport protocol which is being adopted as an alternative to the Transmission Control Protocol (TCP)[3] for delay-tolerant client-server applications. QUIC accounted for a over 30% of Google's outbound traffic in 2016 [**?**], and traffic using QUIC as a transport protocol represented up to 9.1% of global Internet traffic in 2018 [4]. However, standard QUIC not suitable for real-time applications due to high latencies caused by head-of-line blocking as a result of its reliability guarantees and in-order delivery of data to the application. QUIC is a userspace protocol, and can be modified more easily than kernel-level implementations of transport protocols such as TCP. This flexibility allows QUIC to be significantly modified, meaning that QUIC can be adapted for use with real-time multimedia applications.

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.

Real-time multimedia applications typically use the Real-Time Transport Protocol [5],

QUICsilver is a partially reliable implementation of QUIC which achieves 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 can be further improved by taking advantage of existing retransmission behaviours in QUIC, allowing the retransmission of video 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).

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 the `ngtcp2` QUIC stack with standard retransmission behaviours, while tests for partial reliability used the QUICsilver implementation with partial reliability behaviours described above [TODO: write design section].

4.1 Differences in Stack Latency

Stack latencies remain consistently within 15ms of the path latency in QUICsilver 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, QUICsilver 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.

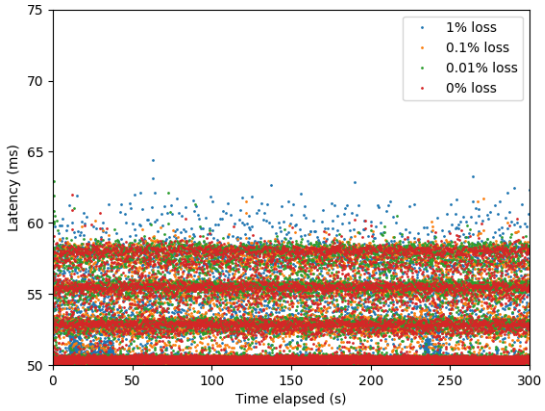
4.2 Differences in Application Latency

QUICsilver 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.

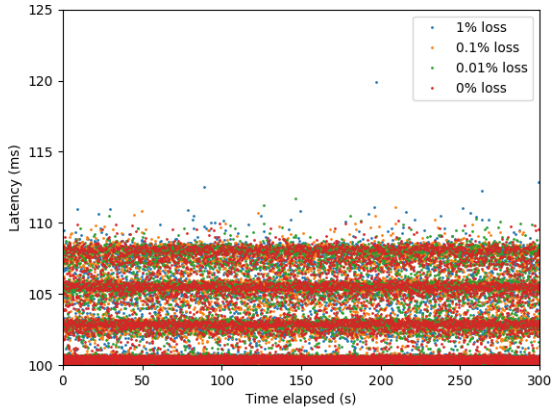
The distinctive four-band pattern present in all instances of QUICsilver tests is due to I-frames being sent as 4 QUIC packets without a 16.7ms delay between them; four packets must be delivered to the application within the same call, but iterating through the reorder buffer and removing relevant items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

The bands in the standard QUIC 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.

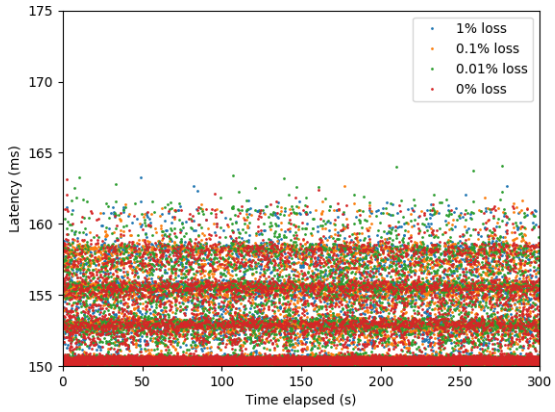
This head-of-line blocking behaviour associated with guaranteed reliability does not affect 0% loss links, where standard QUIC achieves consistently lower application latencies than QUICsilver, and has minimal impact on average application latency in 0.01% loss links. However, for 0.1% and 1% loss scenarios, QUICsilver 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) QUICsilver, 50ms

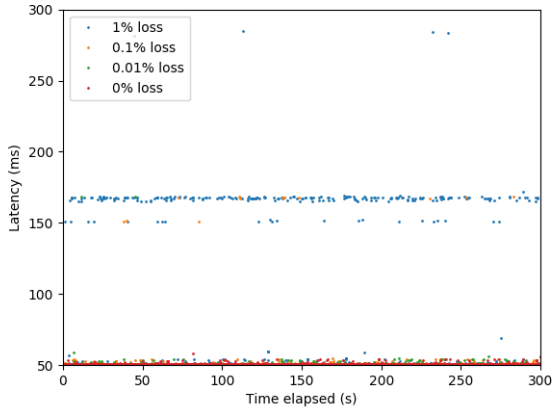


(b) QUICsilver, 100ms

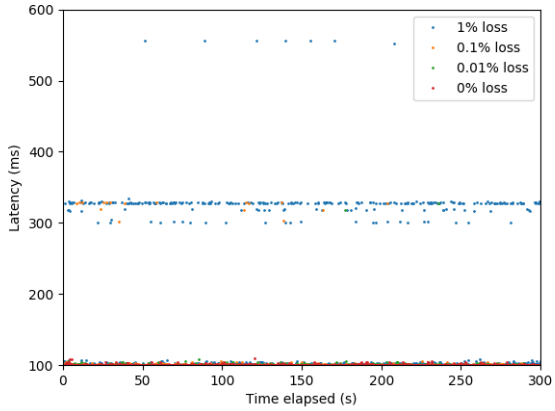


(c) QUICsilver, 150ms

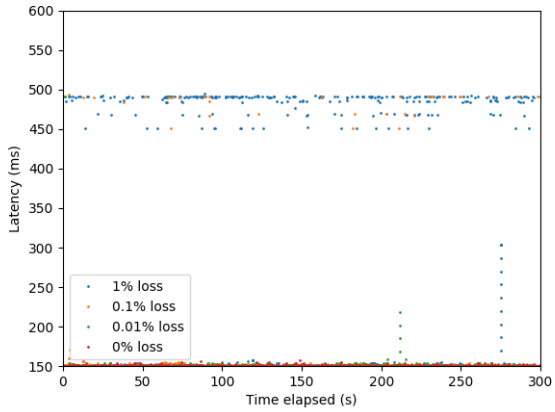
Figure 1: Latency between server and client stack (QUICsilver, partial reliability)



(a) Guaranteed reliability, 50ms

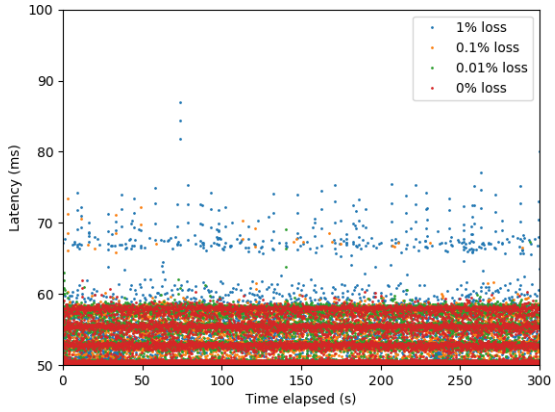


(b) Guaranteed reliability, 100ms

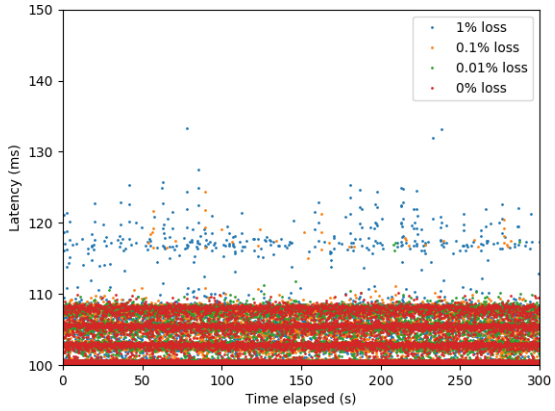


(c) Guaranteed reliability, 150ms

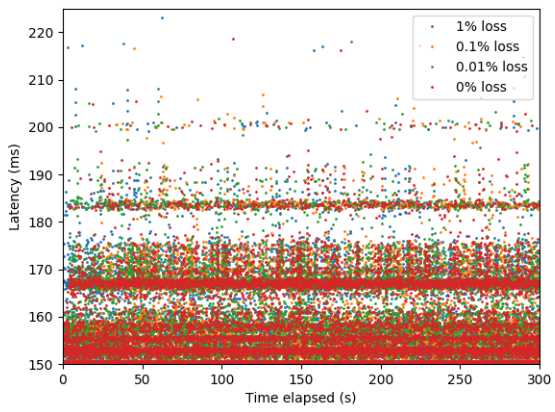
Figure 2: Latency between server and client stack (standard QUIC, guaranteed reliability)



(a) QUICsilver, 50ms

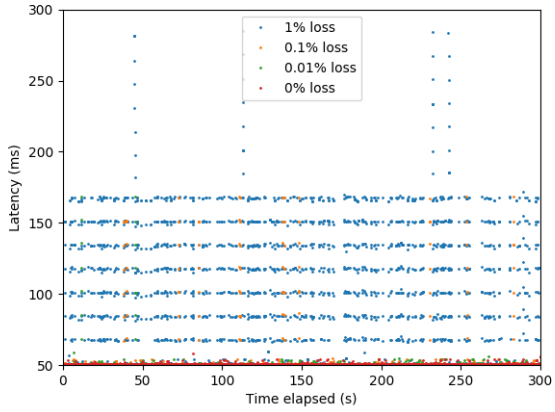


(b) QUICsilver, 100ms

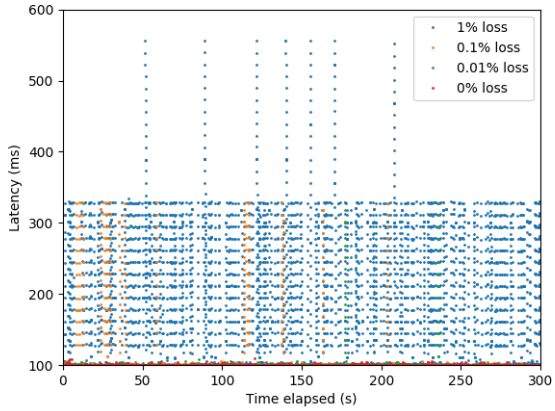


(c) QUICsilver, 150ms

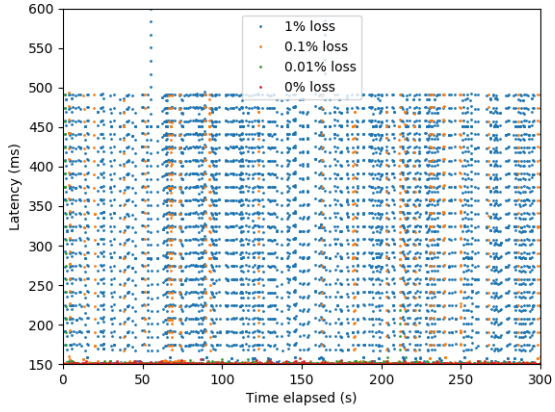
Figure 3: Latency between server and client application (QUICsilver, partial reliability)



(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%

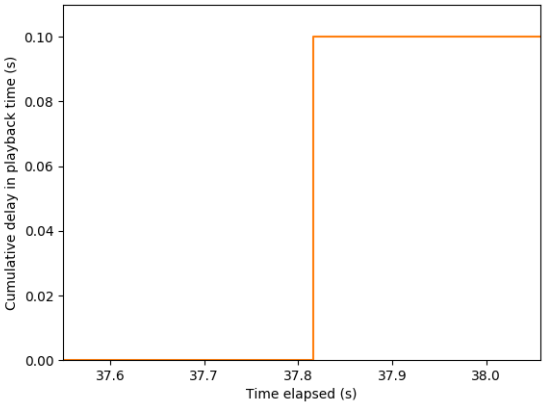
Table 1: Number of stalls and percentage of frames played for guaranteed reliability, 100ms

Loss Rate	Latency (ms)	Discrepancy (s)	Total latency (s)
0.01%	50	0.2	50.2
0.1%	50	1.2	51.2
1%	50	19.8	69.8
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	154.0
1%	150	66.0	216.0

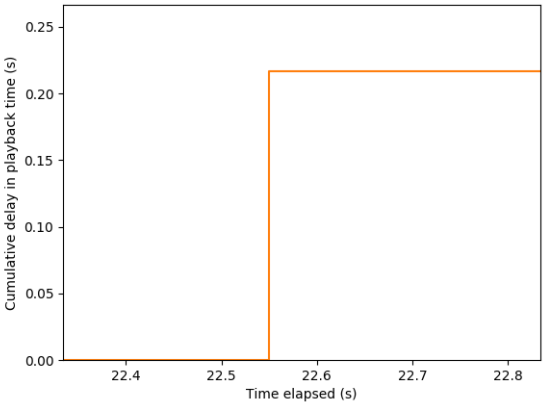
Table 2: Playback discrepancy and total latency for a live-recorded frame to be played by the client application at the end of each test run for each loss rate and latency for guaranteed reliability

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.

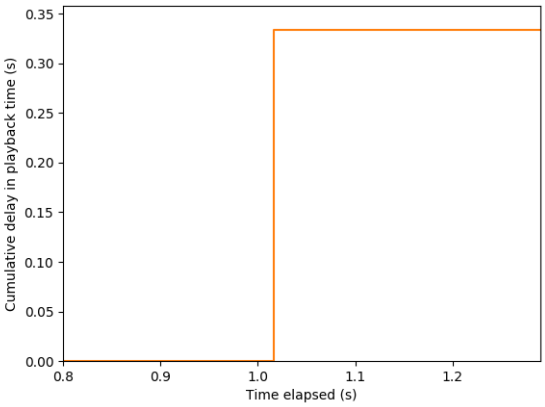
Although 100% of the video frames in the media content being sent are delivered to the client application with guar-



(a) Single instance playback delay, 50ms

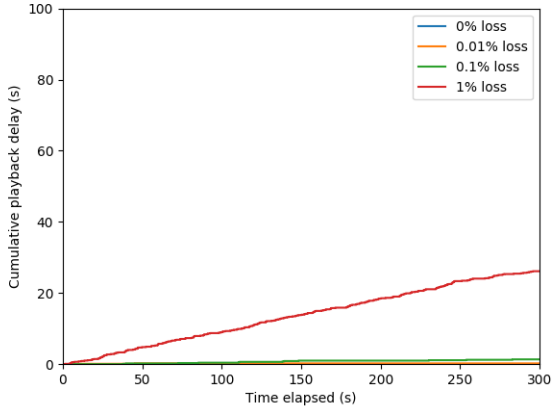


(b) Single instance playback delay, 100ms

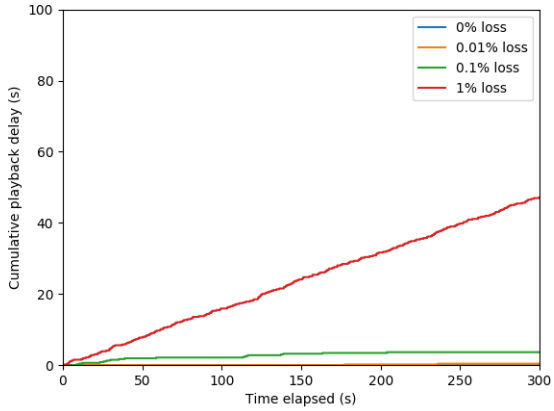


(c) Single instance playback delay, 150ms

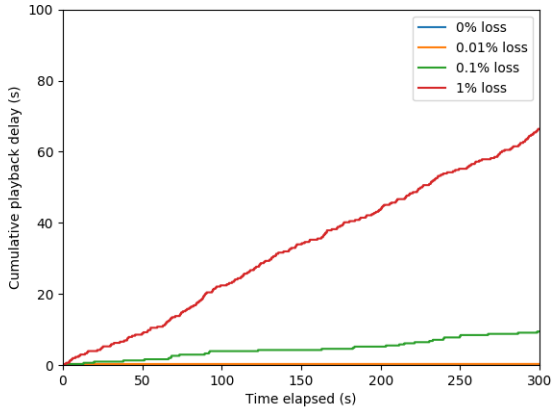
Figure 5: Increase in cumulative playback delay as a result of a single lost packet on various links (standard QUIC, guaranteed reliability)



(a) Playback offsets, 50ms

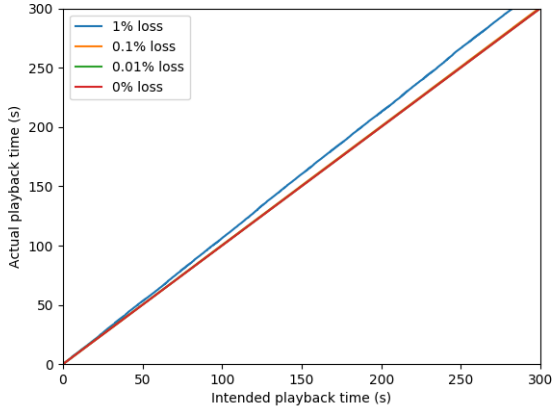


(b) Playback offsets, 100ms

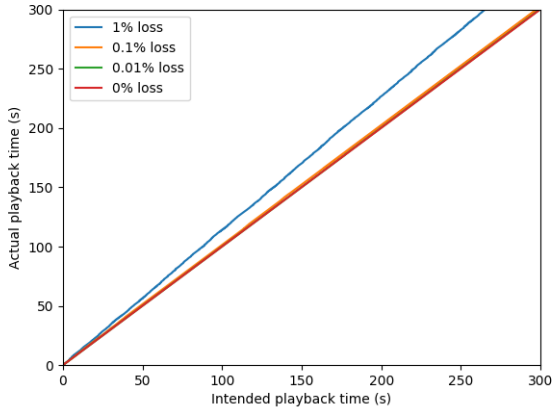


(c) Playback offsets, 150ms

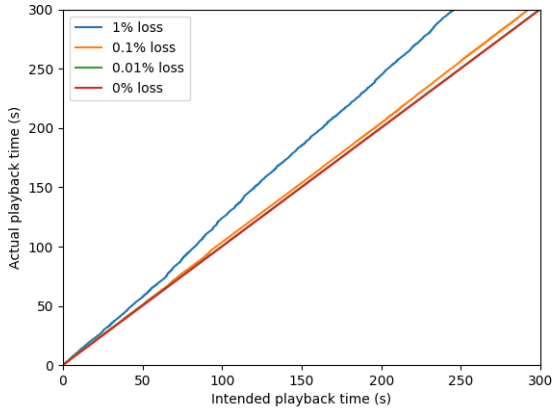
Figure 6: Differences between when a frame is played out compared to the intended playback time (guaranteed reliability)



(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 path latencies in these tests were constant, but path latencies 'in the wild' are variable, and occasionally subject to spikes if a link is congested. The total playback discrepancy suffered by an application would sharply increase as a result of these spikes; a return to lower latencies afterwards would not reduce it.

It should also be noted that this discrepancy is applied to packets which have not been lost in transit: lost packets are subject to even higher latencies, and are increased by multiples of the round-trip time depending on the number of times they are lost. As shown in figure 4, packets which have been received before a missing packet but have a higher packet number also suffer from increased application latency. International Telecommunication Recommendation G.114 recommends no more than 150ms latency "mouth-to-ear" for audio communications [6]; the path latency must be even smaller than this to allow for time to capture live audio and video and to perform codec processing. Figure 4 shows that fully reliable QUIC stays within this limit at 0% and 0.01% loss for 50ms and 100ms links, but it often exceeds even the "mouth-to-ear" limit at 0.1% and 1% loss for all links; 0% and 0.01% loss flows on a 150ms link cannot satisfy the recommended latency requirement due to the link latency. Gaming has even stricter latency bounds [TODO: I know gaming latencies should be around 80ms or below anecdotally, but need a citation with exact bounds for this].

In the case of video streaming, content buffering can be used to disguise stalls. For example, if a user wanted to stream a 1 hour, pre-recorded video, the client might buffer a given amount of content before starting playback. Given that an I-frame is sent every 10th frame as 4 QUIC packets, 216,000 video frames will be sent using 280,800 QUIC packets in total: 28 stalls can be expected to occur at 0.01% loss, 281 stalls at 0.1% loss, and 2808 stalls at 1% loss. At a cumulative latency increase of 100ms per lost packet on a 50ms link, a buffer of 200 seconds should completely conceal stalls for 0.01% and 0.1% loss rates, and will conceal stalls at 1% until 2564 seconds (around 43 minutes) into playback. For a 100ms link with a 216ms latency increase, a buffer of 200 seconds should completely conceal stalls for 0.01% and 0.1% loss rates, and will conceal stalls until 1183 seconds (around 20 minutes) into playback. A 400 second buffer should conceal all stalls. However, buffering is not a feasible strategy for reducing the number of stalls in real-time media delivery: live-generated content cannot be buffered in advance, and buffering does not reduce the cumulative playback delay created as a result of stalls in fully reliable QUIC.

QUICsilver 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 QUICsilver

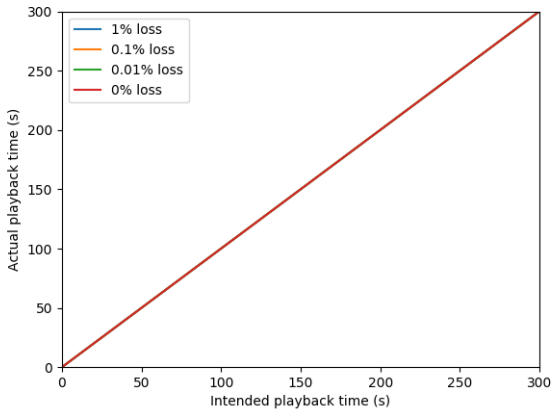


Figure 8: Differences between when a frame is played out compared to when the frame was sent (QUICsilver, partial reliability, all loss rates)

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 the appearance of glitched content for 167ms due to P-frames acting in reference to an incorrect I-frame.

As described in section 3, QUICsilver is capable of performing retransmissions of live I-frames and P-frames, and stale I-frames which are still required for live P-frames. A 4-frame delay adds a maximum of 66.6ms additional latency between the server and the client application, which would create a total maximum of 116.6ms on a 50ms link; this is already relatively high for a real-time application. Removing this delay entirely results in the implementation achieving the lowest possible latency while effectively transmitting P-frames unreliably and retransmitting I-frames once at most.

Whether a stale I-frame can be retransmitted to enable the playback of future P-frames depends on the path latency: if an I-frame is sent once every 10 frames, there is a a space of 150ms between I-frames being played back. Given that a retransmission takes one round-trip time in total, an I-frame fragment would enable the 9th subsequent P-frame if it was sent over a 66.6ms latency link; lower latencies would allow a larger number of associated P-frames to be played successfully, while any latency above this would not perform any retransmissions and would function as an unreliable transport. Sending I-frames less frequently would

allow a larger window for I-frame fragments to be retransmitted, but the amount of glitched playback would increase in proportion with the link latency; the longer an application has to wait for a complete I-frame, the more P-frames are played in reference to the incorrect I-frame.

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 standard QUIC (table 2), but the crucial difference between these two outcomes is that the content received after a loss in QUICsilver 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 correctly 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 QUICsilver results in improved throughput of useful data compared to attempting to optimise applications on top of standard QUIC for real-time behaviour.

5. FUTURE WORK

5.1 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.

5.2 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 QUICsilver 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.

5.3 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.

6. CONCLUSIONS

6.1 Placement of Complexity

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 and only removed from the retransmission buffer with a client-sent ACK, while movement would use partial reliability and deadline-based removal 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.

6.2 Partial Reliability for Other Real-Time Applcations

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 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.

6.3 Uses for QUICsilver

QUICsilver provides better performance for applications which require timely responses to user input than standard QUIC through the use of partial reliablity. It achieves this by adjusting its stream read offsets and removing items from its retransmit buffers in response to playback deadlines, yielding consistently lower application latencies at all link latencies and loss levels. This results in intermittent

glitches in video playback, but over 95% of unique frames sent by the server are played back without glitches even in 1% loss environments, and QUICsilver experiences no cumulative playback latency which prevents extended sessions of real-time media applications from suffering an increasing delay between frames being sent and frames being played. Adjustments to the rate at which I-frames are sent would allow I-frame retransmissions to allow subsequent P-frames to be played correctly; at the current rate of 1 in 10 frames, QUICsilver effectively performs as an unreliable transport at path latencies above 66.6ms.

Standard QUIC could potentially be used for real-time applications over low latency paths with less than 0.01% loss, but it is vulnerable to increased application latency and playback discrepancies lasting for the rest of the sessions as a result of latency spikes on the path. However, standard QUIC is more suited for high-quality video streaming than QUICsilver: pre-existing content can be buffered in advance to mask stalls, and 100% of frames are played without glitches. Buffering pre-existing content could also be performed with QUICsilver, but the stack and application latencies would increase due to performing liveliness checks on many items which would be stored in the retransmit and reorder buffers at the client and server respectively; this is wasted time given that these checks to prevent stalls are not necessary when stalls are already being masked by buffered content.

7. REFERENCES

[1] RFC 791: Internet Protocol: DARPA Internet Program Protocol Specification. Sep 1981.

[2] CISCO. The Zettabyte Era: Trends and Analysis. *Cisco*, (May 2015):1–29, 2015.

[3] J. Rey, D. Leon, A. Miyazaki, V. Varsa, and R. Hakenberg. RFC 793: Transmission Control Protocol: DARPA Internet Program Protocol Specification. Sep 1981.

[4] J. Rüth, I. Poese, C. Dietzel, and O. Hohlfeld. A First Look at QUIC in the Wild. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10771 LNCS:255–268, 2018.

[5] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RFC 3550: RTP: A Transport Protocol for Real-Time Applications. Jul 2003.

[6] I. T. Union. ITU-T Recommendation G.114. 2003 May.