Real-time Audio-Visual Media Transport over QUIC

Colin Perkins
University of Glasgow
csp@csperkins.org

Jörg Ott
Technische Universität München | callstats.io
ott@in.tum.de

ABSTRACT

We consider the problem of how to transport low-latency, interactive, real-time traffic over QUIC. This is needed to support applications like WebRTC, but difficult to support due to the reliable, unframed, nature of QUIC streams. We review the needs of low-latency real-time applications and how they have been supported in previous protocols, then propose a minimal set of extensions to QUIC to provide such support. Compared to a raw datagram service, our extensions provide meaningful support for partially reliable and real-time flows, in a backwards compatible manner.

CCS CONCEPTS

• **Information systems → Multimedia streaming**; • **Networks → Transport protocols**;

KEYWORDS

QUIC, RTP, Video Streaming, Interactive Conferencing

ACM Reference Format:
Colin Perkins and Jörg Ott. 2018. Real-time Audio-Visual Media Transport over QUIC. In *Workshop on the Evolution, Performance, and Interoperability of QUIC (EPIQ'18), December 4, 2018, Heraklion, Greece.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3284850.3284856

1 INTRODUCTION

Real-time multimedia, in its various forms, comprises the majority (>70%) of traffic on the Internet [7]. The transport protocol used to deliver this traffic varies depending on the application and its use cases. Interactive applications will typically use RTP [33] running over UDP, since they have strict latency bounds and require timeliness rather than fully reliable delivery. Streaming applications, on the other hand, have more relaxed latency bounds and so generally make use of HTTP adaptive streaming, e.g., MPEG DASH [36], running over TCP, trading-off latency for ease of deployment via commodity CDNs.

As QUIC matures, interest is growing in whether it can effectively deliver some, or all, of this multimedia traffic, to replace or augment the existing transport protocols. Since QUIC is designed as a transport for HTTP/2, there are obvious ways for it to carry HTTP adaptive streaming traffic. The mapping of interactive traffic, and RTP, onto QUIC is less clear, however, since there are some significant semantic mismatches between the two protocols.

We explore the issue of how best to transport real-time audio-visual media over QUIC. We first review the semantics of RTP and DASH, the expectations they have on behaviour of the underlying transport, and the implications of their different use cases for the transport (§2). Building on this, and our experiences with the standardisation of RTP, we discuss how best to deliver interactive media flows over QUIC (§3), and then, based upon a brief analysis of media transport requirements (§4), discuss how QUIC and RTP might co-evolve to better suit the needs of future real-time media applications, both interactive and non-interactive (§5).

2 REAL-TIME MEDIA IN THE INTERNET

The IP layer provides a best effort packet delivery service. The transport layers on this best effort network, and provides services like sequencing, timing, reliability, and congestion control. Explicit congestion notification (ECN) and enhanced QoS for multimedia traffic are deployed in some parts of the network, but their presence cannot be relied upon.

Real-time audio-visual applications have historically used transport protocols that favour timeliness over reliability. The IETF standard in this category is RTP [33]. This is widely deployed in voice-over-IP and video conferencing systems, using H.323 [39], SIP [32], or WebRTC [16] signalling. RTP typically layers on UDP, accepting its unreliable but timely packet delivery, and adding an additional header that provides a sequence number and timestamp, payload type, and source identifier. Extensions provide partial reliability via FEC [21, 29] or retransmission [25, 31], congestion control [17, 42], codec control, and reception quality reports. RTP payload formats specify how codec output is packed into datagrams [11], with special consideration made for application-level framing [8] so each packet can be processed regardless of whether previous packets were received. This intelligent framing comes at a cost: senders and receivers must be updated to understand the payload format for each new codec. In addition, since transport is unreliable, receivers must be robust to packet loss and able to decode partial streams.

RTP was designed at a time, the mid-1990s, when real-time media transport over the Internet was a marginal proposition, and robustness to packet loss and other network vagaries was a real concern. It is well suited to interactive applications and to environments, such as wireless networks, where network performance may be suboptimal. It provides the features to allow smart endpoints to make effective use of such networks, at the cost of implementation complexity.

Early deployments of on-demand streaming also used RTP, often with an RTSP [34] control channel. With the growth of CDNs, however, most transitioned to HTTP adaptive streaming protocols, such as MPEG DASH [36]. Unlike RTP, HTTP-based transports are

EPIQ'18, December 4, 2018, Heraklion, Greece

ACM ISBN 978-1-4503-6082-1/18/12…$15.00
https://doi.org/10.1145/3284850.3284856

not optimised for media delivery. The server prepares the media data for delivery as a sequence of independently decodable *chunks*, each of a few seconds in duration and made available pre-encoded at a range of bit rates. Clients retrieve a *manifest* that indexes the available content, then pull chunks in sequence using HTTP GET requests over TCP. The client chooses the bit rate to fetch for each chunk based on the download rate it observed for previous chunks. Since transport is over TCP, media delivery is reliable, ordered, and congestion controlled. Receiver-side buffers must be large enough to hide the resulting variation in download times and allow continuous, stall-free, play-out. This makes HTTP adaptive streaming poorly suited to low-latency play-out. It is, however, simpler to implement than RTP-based solutions, and can re-use existing CDN infrastructure since the server is a standard HTTP server.

The evolution of real-time media transport has mirrored the evolution, and commodification, of the underlying network infrastructure. RTP includes numerous features intended to help implementations compensate for network pathologies and performance bottlenecks that were common at the time it was designed, but are now less so. HTTP adaptive streaming applications, downloading on-demand media that is real-time but has comparatively relaxed latency bounds, are generally used on high-quality networks where such problems are rare, so the protocol doesn't need to handle them. The result is that RTP addresses use cases that DASH-like protocols do not, but suffers from the complexity needed to do so. Media delivery over QUIC should carefully define its applicability, rather than blindly copy either approach.

3 RUNNING DASH AND RTP ON QUIC

Mapping HTTP adaptive streaming protocols onto QUIC is straightforward, since QUIC was designed to directly support HTTP/2 [4]. Adaptive streaming applications can therefore run unchanged, aside from being updated to use the current version of HTTP. By running over QUIC, such applications gain the benefit of modern loss recovery and congestion control algorithms that may include features that have yet to be incorporated into the operating system's TCP stack. In addition, QUIC natively supports sending multiple streams within a single transport layer flow, with HTTP/2 streams mapping directly onto QUIC streams. Traffic sent on each stream is delivered reliably and in-order, but QUIC does not maintain ordering between streams, removing one source of head-of-line (HoL) blocking. Applications that send each chunk of media on a separate HTTP/2 stream therefore have a natural breakpoint at which they will recover from stalls, potentially improving the user experience.

Effectively mapping RTP onto QUIC is more difficult. In the following, we discuss how RTP can be mapped onto the current version of QUIC. Then, in §4, we begin a more principled feature comparison, to consider if this is the right approach, or whether QUIC and RTP should jointly evolve to provide more a effective transport for interactive multimedia.

3.1 Mapping RTP onto QUIC -14

QUIC [14, 15] provides a reliable, connection-oriented, byte stream service. There is an existing mapping of RTP onto such a service [19], designed for use with TCP, and it's possible to reuse it to run RTP on QUIC. This mapping has the advantage of simplicity: both RTP data and control (RTCP) packets are sent on a single QUIC stream, with each packet prefixed by a 16-bit unsigned integer length field.[1] Packets are demultiplexed based on their RTP payload type. The equivalent framing over TCP is widely implemented, and performs acceptably well if packet loss is bounded [6].

A key problem with this approach is head-of-line blocking. Since all packets are sent on a single reliable stream, loss of any packet will block delivery of later packets until the lost packet has been retransmitted. This is unacceptable for interactive RTP applications that can tolerate some packet loss, but have strict latency bounds.

Various approaches can be taken to reducing this problem, based around use of QUIC streams to distinguish RTP flows that can be independently delivered. A simple approach might map traffic for each RTP synchronisation source (SSRC) to a different QUIC stream, preventing loss of a video packet from blocking delivery of later audio, for example.

Media generated by an SSRC might be further sub-divided across QUIC streams. The sender could encode the media in chunks, following the example of HTTP adaptive streaming applications, switching to a new QUIC stream and sending an I-frame every n video frames, where n is chosen based on the frequency of packet loss events to limit stall duration. In the extreme, $n = 1$ and a new stream is used for every frame of video, although this is likely unnecessary. A more optimal approach might be for the receiver to report loss events and use this feedback to trigger a switch to a new QUIC stream.

3.2 Limitations and Discussion

Such approaches address, to varying degrees, the issues of HoL blocking and framing. They do not, however, eliminate the problem of media data being retransmitted once it has passed its play-out deadline and is no longer needed, which is inherent due to the reliable nature of QUIC and its lack of awareness of timing information.

Unnecessary retransmissions may be addressed by adding unreliable, or partially reliable, transmission to QUIC. The former was suggested by Tiesel *et al.* [37], who discuss marking some streams as unreliable, subject to the usual QUIC congestion control rules, and propose allowing data from unreliable streams to be given priority over reliable streams upon transmission. Lubashev [22] discusses the latter, and takes the approach of indicating, per stream, that parts of the stream (to a certain byte offset) no longer need to be retransmitted. Neither proposal offers media-aware retransmissions, and both need relatively sophisticated APIs to offer fine-grained control of the (re-)transmission and scheduling.

Framed delivery of RTP packets over a stream transport also doesn't account for datagram boundaries. Unless media data units are delineated at the QUIC layer, it is possible that loss of a single datagram will impact multiple encapsulated RTP packets, as those packets can be arbitrarily split across datagrams. Designers of RTP payload formats spend considerable effort making packets independently decodable [8, 11] to avoid this loss multiplier effect, and it

[1] Such a format was suggested recently (2018-08-27) on the QUIC WG mailing list as an extension to QUIC to carry messages and subsequently documented [26].

#	RTP Field and Function	QUIC Mapping	Support?
1	Seq# for loss detection	Seq# plus ACK mechanism	Yes
2	ECN marking	ECN marking	Yes
3	Media-specific timestamps	Use metadata on top of QUIC	No
4	(One-sided) wall-clock sync	Extension or metadata on top of QUIC	No
5	Data retransmission	Adapt retransmission for partial reliability	(Yes)
6	Generic FEC	Could be added as a generic function (was discussed)	(No)
7	Media-specific redundancy	Could be realized as payload	N/A
8	General RX stats from RTCP RR	ACK blocks for losses (abs. and rel.) to be augmented	Partly
9	Congestion control	Done by QUIC, may need an API	(Yes)
10	Selective encryption of payloads	Almost full encryption largely including headers	(Yes)
11	SSRC and CNAME for media bundling	Bundling implicit within a QUIC connection	Implied
12	Payload type + M bit marking	Use payload framing on top of QUIC	N/A
13	Source identification (SSRC/CSRC)	Use metadata on top of QUIC	N/A
14	SDES session metadata	Use metadata on top of QUIC	N/A
15	External signalling channel	Use an in-band QUIC stream if feasible	(Yes)
16	IP Multicast support	Not required	No
17	Mixers and translators	Implement in the application above QUIC	No
18	Transmission scheduling control	Done by QUIC, needs an API	(Yes)
19	Header extensions and metadata	Use metadata on top of QUIC	No
20	Application level framing	Partial, via QUIC streams	Yes
21	Avoidance of HoL blocking	Partial, via QUIC streams	Yes

Table 1: Mapping RTP functions to QUIC

should be considered in the QUIC mapping. API changes to expose the MTU and frame boundaries can solve this in part, but affect packet scheduling, congestion control, and reliability.

QUIC flows are encrypted and authenticated to ensure privacy and integrity of the headers and payload. The Secure RTP extension [2] can be used to provide privacy and integrity protection for RTP traffic, when combined with a keying protocol such as DTLS [9, 23]. QUIC integrates security more tightly, and can benefit from features such as 0-RTT session resumption that have not yet been incorporated into DTLS-SRTP. Another key difference is that QUIC encrypts the entire packet, including the overwhelming majority of the transport headers, whereas Secure RTP encrypts only the payload and leaves the RTP and transport headers in the clear. As we discuss in Section 4, this allows Secure RTP to support header compression and semi-trusted intermediaries, but does expose more information to the network.

It is clear that framing interactive RTP media traffic for transport over QUIC exposes a number of issues. As noted above, solutions exist to some of these within the scope of QUIC as currently specified, but not all. In the following, we consider how QUIC and RTP might co-evolve, to better address these concerns, and provide an effective solution for latency bounded, interactive, media.

4 MEDIA TRANSPORT REQUIREMENTS

The discussion in Section 3 highlighted three significant limitations with running interactive media over the current version of QUIC: HoL blocking, unnecessary retransmissions, and application-level framing. In the following, we begin a more systematic exploration of the issues.

We outline a reasonably complete list of the features of RTP in Table 1, with an indication of how the concepts might map onto a future version of QUIC, and whether they need to be supported. The table indicates the RTP functions, how those could be mapped to an (enhanced) QUIC, and if those functions should be supported as part of a generic, real-time enabled QUIC stack (or rather be left to the application running on top). These features can be broken down into the following categories:

- **Transport-related features** that need support from the transport protocol for efficient and/or effective implementation, such as congestion control, avoiding HoL blocking, loss detection, and ECN support;
- **Session-related features** relating to orchestration of multiple flows, participant identification, and quality of experience reporting; and
- **Media-related features** such as codecs and payload formats, mixing and translation, lip-synchronisation, and media rate adaptation.

There is necessarily close coupling across categories, and the design does not follow a traditional layered model. This may complicate integration of real-time media support into QUIC. For example, effective implementation of congestion control requires close coupling between the transport layer as it reports on lost and ECN marked packets and packet arrival times, the transmission scheduler, and the media codec that must adapt its output to match the required rate.

Of the features in Table 1, transport-level detection and reporting of packet loss and ECN marks (#1, 2) are directly supported by both RTP and QUIC. For other features, such as packet scheduling (#18), congestion control (#9), some aspects of reliability (#5, 6, 7, 8), application level framing (#20), and avoidance of HoL blocking (#21) support is present in QUIC, but is not always well aligned with the needs of real-time media.

Since RTP is inherently a group communication protocol, it naturally supports multicast (#16). Current practice, however, is to implement multiparty support using relays, so multicast support is not required in QUIC (but could readily be used by real-time media applications, if provided [18]).

Many session-level features can be implemented above the QUIC layer. Rather than SSRCs, CNAMEs, and other signalled identifiers for media bundling and demultiplexing (#11), existing QUIC stream mechanism can be used. In a similar way, source identification and

description information (#13, 14) can be carried as session-level metadata in a signalling channel (#15) sent as an additional QUIC stream. The multi-streaming nature of QUIC should greatly simplify multiplexing of media flows and signalling.

As expected, many media-related features have no clear analogue in QUIC. Extensions are needed to support media timing (#3) and inter-flow synchronisation (#4). Per-packet metadata (#12, 19) can be sent in the payload of QUIC frames.

Finally, QUIC and RTP have different approaches to encryption (#10). Secure RTP was designed to leave headers in the clear to enable efficient header compression on low bandwidth and wireless links [5, 27] and to allow for easy processing in intermediaries (#17). Bandwidth increases have made header compression less important, although it may still matter in some wireless environments, but there is certainly still a need for semi-trusted conferencing middleboxes that have access to media header information, to allow source selection, but not to the media payload. These are not well supported by QUIC with its focus on end-to-end encryption and some form of multi-layered security is needed.

5 A STRAW MAN PROPOSAL: QUIC-R

In the following, we outline an initial proposal for a set of extensions to QUIC that better support real-time applications, dubbed QUIC-R. We focus on the shortcomings identified above, and on addressing them in a manner that we believe is consistent with both QUIC and RTP, and adds only reusable and general purpose functionality to QUIC.

5.1 Real-time Media Transport

The main points for adding real-time capabilities to QUIC transport are four-fold:

(1) Controlling transmission scheduling and sender-side buffering, to ensure data is sent in a timely manner, and that outdated or otherwise stale content does not delay new transmissions;
(2) Obtaining fine-grained statistics about timing, losses, etc., at sender and receiver sides, that can be used to tune media transmission and ensure timely play-out;
(3) Providing adjustable repair mechanisms – reactive or proactive – to control retransmission and/or redundant transmissions or discarding of packets; and
(4) Enabling content-aware congestion control for streams, embedded within a larger congestion control framework for the QUIC connection, shared across multiple streams (possibly coupling congestion control across streams, possibly decoupled from others, while staying within an overall congestion window).

A key question to be answered before looking at protocol details is whether real-time communication should fit within the stream concept, or should it be separate? We opt for an integration, to ensure a more uniform API is exposed to programmers. We believe that the core properties of QUIC will benefit from real-time support, and that the versatility of a mechanism to support communication with control over reliability and timing will bring value to a range of applications, not just to interactive audio-visual traffic.

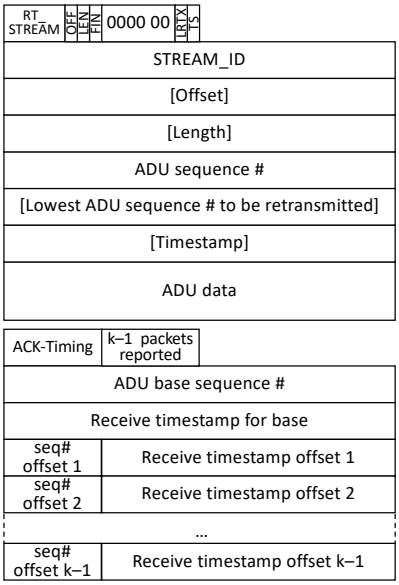

Figure 1: Strawman packet format

To this end, we propose to add **RT-streams** to QUIC. RT-streams share the stream ID space with regular streams, but send `RT_STREAM` frames rather than standard `STREAM` frames. The new frame provides record marking and, within bounds, allows controlled mapping of data to QUIC packets; it also provides a clean abstraction for implementing transmission scheduling, partial reliability, and reception statistics. `STREAM` and `RT_STREAM` frames cannot be mixed on the same stream (but one could, e.g., designate one bit in the stream id space to indicate unreliable streams), and `RT_STREAM` frames can only be sent after 1-RTT keys are established.[2] Support for RT-streams would be negotiated during QUIC connection establishment and be cached along with other peer properties.

An RT-stream comprises a sequence of records, rather than an octet stream, and preserves record boundaries to carry content formatted into Application Data Units (ADUs). Each ADU constitutes an independently decodable and processable piece of encoded media content, similar to RTP payloads, such an audio sample or frame, a video slice or frame, etc. Each ADU is crafted to fit into a single `RT_STREAM` frame and, if ADUs are small, multiple ADUs may go into the same QUIC frame (as is usually the case of audio and other low bitrate media). Sometimes (as, e.g., in the case of video I-frames) a media frame may need to be split across multiple packets and then be formatted by the application into multiple ADUs. We push this responsibility to the application because we do not want to burden an RT-stream in QUIC with a reassembly mechanism for large ADUs and avoid questions of atomic vs. partial delivery of ADUs, maximum ADU size considerations, etc. Instead, we leave this up to the application layer, as readily exercised using the concept of payload formats in RTP.

[2] Sending real-time frames as 0-RTT data should, in principle, also be possible but we leave this for further study.

The header of each RT_STREAM frame (see Figure 1 for a sample, not yet optimized, encoding) includes a record (ADU) sequence number and timestamp, as well as optional offset and length fields that have the same meaning as in STREAM frames. This is the minimal additional header information needed for real-time operation. To support partial reliability, an additional sequence number indicating the retransmit limit. Two bits indicate the presence of the respective fields.

At the sender, data is written to an RT-stream along with its timestamp, an expiry time, and the sequence numbers of any previous records on which it depends. The sequence number assigned to the record is returned to the application. The timestamp, expiry time, and dependency information is used by the sender to schedule packets for transmission, and possible retransmission; the expiry time and dependency information is not included in transmitted frames. It is an error to send more data in a record than fits the path MTU, including all QUIC headers.

At the receiver, records (= ADUs) are released to the application as they arrive, as a sequence of independent records, along with their offset in the stream, sequence number, timestamp, and arrival time. The timestamp allows the application to reconstruct the timing, while the sequence number and offset can be used for loss detection and to reconstruct order.[3] Frames are acknowledged as normal, but are not necessarily retransmitted; the retransmit limit header indicates the point beyond which the sender will no longer retransmit lost packets. An additional ACK-Timing frame is sent, indicating the arrival time of frames, e.g., using a full record sequence number and timestamp for the first (base) packet and then relative offsets for following ones (see figure 1). Any further processing, such as playout buffering, intra- and inter-stream synchronization, are up to the application.

Senders and receivers can send WALLCLOCK frames (not shown in the figure) to carry a 64-bit NTP-format timestamp that is associated with an RT_STREAM frame, similar to how RTCP sender report (SR) packets [33]. The WALLCLOCK frames are sent occasionally, perhaps every few seconds, and *must* be sent in the same QUIC packet as an RT_STREAM frame. This allows the receiver to match the sender's wallclock time to the media-specific timestamp in the RT_STREAM frame, hence allowing it to synchronise playout of multiple flows (e.g., to lip-sync audio and video flows).

Knowledge of real-time behaviour is limited to the sender, which must schedule packets for (re)transmission based on their timestamp, expiry time, dependencies, and the dictates of congestion control. The receiver merely passes frames and their metadata to the application. Retransmissions follow the routine of QUIC, until a packet's expiry time is reached.[4] Further (media-specific) error control, such as redundancy encoding, unequal error protection, FEC, or reference picture selection, is implemented at the application layer.

Packet scheduling and congestion control must coordinate real-time and non-real-time streams. QUIC foresees (at the API) indicating stream data priorities for scheduling purposes, which an application can use to knowingly arbitrate path capacity between different streams. Congestion control needs to cope with the potentially different rate adjustment needs for real-time media sources and elastic data. We expect the changes to congestion control and packet scheduling will be the primary sources of extra complexity in our proposal, compared to baseline QUIC.

One approach could be to define, per QUIC connection, whether the real-time or the regular congestion control regime takes precedence. For real-time precedence, a "pluggable" congestion controller at the application layer could take over and also rule packet pacing of real-time streams, leaving (residual) capacity to elastic streams as defined by their priority. In this case, all congestion cues would also be provided via the API to the pluggable congestion controller [24, 38]. The built-in QUIC congestion control could run in parallel and observe that the mid-term behavior is in line with that of a native QUIC stack, possibly provide hints of excess data transmission and, if needed, act as a *circuit breaker* [28, 35].

5.2 Session Management

Application- and media-specific metadata is not exposed to QUIC. If the equivalents of the RTP SSRC and CSRC list, payload type, marker bit, and RTCP information are needed, they must be sent in the QUIC payload. Packets are acknowledged as normal, and ACK-Timing frames also feedback the arrival time, but if more sophisticated reception quality and/or quality of experience metrics are needed (cf. RTCP XR [10] and its extensions), they must be sent in the QUIC payload.

Each media stream is sent as separate QUIC RT-stream. It's expected that the first stream(s) opened will provide a signalling channel, while later streams will carry media data. The signalling channel is used to convey the meaning of the later streams, in an application-specific manner. Naturally, not all (media) streams need to be RT-streams.

Multiparty communication is commonly implemented in RTP using middleboxes that terminate the RTP flows carrying the media data, mix or select flows, then forward the modified media over new RTP flows to the other participants in the communication. This relies on such middleboxes participating in the signalling exchange and being trusted with access to the unencrypted media traffic. When forwarding data, such nodes preserve a one-to-one mapping between incoming and outgoing streams, so applications can use session-level signalling (e.g., SSRCs and CSRCs) to indicate end-to-end semantics.

The same approach, using explicitly configured trusted middleboxes, can be used when sending media over QUIC. Since QUIC does not support intermediaries or multiparty communication, the timestamps, loss reports, etc., are only valid between the endpoints of the QUIC connection. Providing reception statistics for controlling and optimizing end-to-end media delivery for groups, across intermediary nodes, therefore requires additional mechanisms above the QUIC layer. We defer discussing design options and implications for congestion control, scheduling, and reporting to future work.

The IETF Privacy Enhanced RTP Conferencing (PERC) working group is developing an alternative approach where the middleboxes are semi-trusted. In this approach, packets are encrypted twice: one level of encryption runs end-to-end and protects the media data, while the other runs hop-by-hop. This allows the middleboxes to see enough header information to determine what packets to forward to

[3] Frames are not necessarily all the same size, so while the offset can be used to infer packet loss, a sequence number is needed to know how many frames were lost.
[4] The exception is an RT_STREAM frame with the FIN bit set, that is delivered reliably.

what receivers, but prevents then from accessing or modifying the media data. Similar approaches could be taken with media running over QUIC, where the payload was encrypted separately to the QUIC packets and metadata. Details are for future work.

5.3 API Considerations

The API is deliberately minimal. A QUIC receiver must be aware of RT-streams, so it can pass frames to the application when they arrive, but other than provided framed data and its associated metadata, all the complexity is pushed up above the QUIC layer.

The sender API is more extensive. When an RT-stream is opened, the clock rate used for the timestamps must be specified, so the sender can schedule transmissions (this is passed to the receiver in an application-specific manner). For each frame, a timestamp, expiration time, and any dependencies must be provided, in addition to priority information for the stream data. The sender can optionally provide the application with information about what frames were acknowledged or their receive timing. As noted above, a pluggable congestion control interface is needed to allow the application to take over sending rate adjustments and packet pacing, bypassing built-in QUIC mechanisms.

This API provides a sender, if it so desires, a degree of fine-grained control over (re)transmission scheduling, while allowing less eager real-time application to stay clear of this complexity and let the built-in QUIC-R scheduling and congestion control do the work. All receivers need to care about is playout buffering and rendering, with all other mechanisms needed for operation readily provided by QUIC-R. This makes them easy to implement, reusing existing real-time application code. It also preserves the option to provide additional feedback as part of the QUIC-R (session) payloads for more sophisticated repair and adaptation strategies.

5.4 Applications for QUIC-R

Both today's DASH and RTP applications would benefit from QUIC-R. Adaptive streaming applications could 1) make use of multiple streams to avoid head-of-line blocking; and 2) use unreliable streams for (partial) segments (e.g., those parts not containing an I-frame) to trade off lack of stall events for video frame quality. Such decisions may even be taken, and adjusted, dynamically as a function of the observed path QoS. This would allow changes to the design of adaptive streaming applications. For example, they could avoid the problems caused by nested control loops that present difficulties for proper rate adaptation in DASH [13], and that complicate straightforward integration with Google QUIC [3], while allowing for some unreliability and eliminating HoL blocking. Experiments with an adapted variant of TCP that provides similar features, appear to yield some benefits [1], so we are confident of improvements. With QUIC-R, future designs could even reconsider using a QUIC stream to run a control protocol, perhaps a modern flavour of RTSP, in parallel to provide more effective and lower-latency media control.

Conversational applications using RTP could benefit from readily available security and stream multiplexing mechanisms, greatly simplifying media demultiplexing, signalling, and NAT traversal; as well as the integration of a signalling and media channels. Many of them may be able to just use the built-in QUIC-R congestion control as a starting point, as opposed to implementing their own congestion control. Stream demultiplexing for RTP traffic [20, 30, 41] is inflexible and fragile, and introduces considerable signalling complexity [12, 40]. QUIC provides a native stream abstraction that can be used for this purpose, greatly simplifying applications and providing a more general solution (at the cost of some modest packet expansion compared to RTP-based solutions). This more general demultiplexing solution will reduce the need to send RTP flows, and other traffic, on additional UDP ports to work around limitations of the demultiplex, simplifying NAT traversal. Finally, while signalling for NAT traversal will clearly still need to be involve a binding discovery server, many aspects of media negotiation could move in-band. This addresses frequent complaints about RTP that the flows are not self-describing, that it's difficult to associate signalling with media, and that reliable data cannot be sent on the same path as the media.

6 CONCLUSIONS

We have presented a minimal set of extensions to QUIC to support real-time media. The essential extension is to support partially-reliable framed data with deadlines. Application level framing is needed to make effective use of all packets that arrive, and maintaining real-time behaviour on a best effort network means accepting some potential unreliability. The session- and media-specific details of a protocol, such as RTP, can be implemented above QUIC if it offers these extensions. If they are not offered, the best that can be implemented requires the application work-around unnecessary retransmissions that disrupt timing while suffering from the loss multiplier effect inherent when frame boundaries and packet boundaries are mismatched.

We believe the extensions we propose can be added to QUIC in a backward compatible manner, easing deployment. Nothing must be added to the first release of IETF QUIC. This is possible, in part, because QUIC flows are protected from ossification by end-to-end encryption and greasing.

Our proposal is optimised for *real-time* traffic, providing framing, timing, and loss detection. We believe this service is, and should be, distinct from a simple unreliable datagram service. Applications that have strict timing requirements require different packet scheduling and congestion control decisions to those, e.g., VPNs, that merely desire unreliable transport. We expect that, over time, QUIC will evolve to support reliable streams, unreliable datagrams, *and* real-time framed streams.

Numerous future issues are to be discussed: as QUIC evolves to support multipath communication, scheduling complexity will increase, and so will the demands on a pluggable congestion control module. Awareness of multiple paths will also potentially add to the API. New API designs [38] will be needed, and applications and transport will likely become more closely coupled. The typical user-space implementation of QUIC makes this feasible. Along with QUIC-R, these will allow QUIC to evolve into an effective, general purpose, transport, addressing many of the concerns around transport ossification.

ACKNOWLEDGEMENTS

This work was funded in part by the UK Engineering and Physical Sciences Research Council (grant EP/R04144X/1).

REFERENCES

[1] S. Ahsan, S. McQuistin, C. S. Perkins, and J. Ott. 2018. DASHing Towards Hollywood. In *Proceedings of the Conference on Multimedia Systems (MMSys)*. ACM, Amsterdam, The Netherlands, 1–12. https://doi.org/10.1145/3204949.3204959

[2] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman. 2004. The Secure Real-time Transport Protocol (SRTP). Internet Engineering Task Force. (March 2004).

[3] D. Bhat, A. Rizk, and M. Zink. 2017. Not so QUIC: A performance study of DASH over QUIC. In *Proceedings of the International Symposium on Network and Operating Systems Support for Digital Audio and Video*. ACM, Taipei, Taiwan, 13–18.

[4] M. (Ed.) Bishop. 2018. Hypertext Transfer Protocol (HTTP) over QUIC. Internet Engineering Task Force. (June 2018). draft-ietf-quic-http-14.

[5] C. Bormann, C. Burmeister, M. Degermark, H. Fukushima, H. Hannu, L.-E. Jonsson, R. Hakenberg, T. Koren, K. Le, Z. Liu, A. Martensson, A. Miyazaki, K. Syanbro, T. Wiebke, T. Yoshimura, and H. Zheng. 2001. RObust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed. Internet Engineering Task Force. (July 2001). RFC 3095.

[6] E. Brosh, S. A. Baset, D. Rubenstein, and H. Schulzrinne. 2008. The Delay-Friendliness of TCP. In *Proceedings of the SIGMETRICS conference*. ACM, Annapolis, MD, USA, 49–60. https://doi.org/10.1145/1375457.1375464

[7] Cisco. 2017. Cisco Visual Networking Index: Forecast and Methodology, 2016–2021. Document available online. (June 2017). https://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/.

[8] D. D. Clark and D. L. Tennenhouse. 1990. Architectural Considerations for a New Generation of Protocols. In *Proceedings of the SIGCOMM conference*. ACM, Philadelphia, PA, USA, 200–208. https://doi.org/10.1145/99508.99553

[9] J. Fischl, H. Tschofenig, and E. Rescorla. 2010. Framework for Establishing a Secure Real-time Transport Protocol (SRTP) Security Context Using Datagram Transport Layer Security (DTLS). Internet Engineering Task Force. (May 2010). RFC 5763.

[10] T. Friedman, R. Caceres, and A. Clark. 2003. RTP Control Protocol Extended Reports (RTCP XR). Internet Engineering Task Force. (November 2003). RFC 3611.

[11] M. Handley and C. S. Perkins. 1999. Guidelines for Writers of RTP Payload Format Specifications. Internet Engineering Task Force. (December 1999). RFC 2736.

[12] C. Holmberg, A. Alvestrand, and C. Jennings. 2018. Negotiating Media Multiplexing Using the Session Description Protocol (SDP). Internet Engineering Task Force. (September 2018). draft-ietf-mmusic-sdp-bundle-negotiation-53.

[13] T.-Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari. 2012. Confused, Timid, and Unstable: Picking a Video Streaming Rate is Hard. In *Proceedings of the Internet Measurement Conference*. ACM, Boston, MA, USA, 225–238.

[14] J. Iyengar and I. (Eds.) Swett. 2018. QUIC Loss Detection and Congestion Control. Internet Engineering Task Force. (August 2018). draft-ietf-quic-recovery-14.

[15] J. Iyengar and M. (Eds.) Thomson. 2018. QUIC: A UDP-Based Multiplexed and Secure Transport. Internet Engineering Task Force. (August 2018). draft-ietf-quic-transport-14.

[16] C. Jennings, T. Hardie, and M. Westerlund. 2013. Real-Time Communications for the Web. *IEEE Communications Magazine* 51, 4 (April 2013), 20–26. https://doi.org/10.1109/MCOM.2013.6495756

[17] I. Johansson and Z. Sarker. 2017. Self-Clocked Rate Adaptation for Multimedia. Internet Engineering Task Force. (December 2017). RFC 8298.

[18] L. Lardue and R. Bradbury. 2018. Hypertext Transfer Protocol (HTTP) over multicast QUIC. Internet Engineering Task Force. (August 2018). draft-pardue-quic-http-mcast-03.

[19] J. Lazzaro. 2006. Framing Real-time Transport Protocol (RTP) and RTP Control Protocol (RTCP) Packets over Connection-Oriented Transport. Internet Engineering Task Force. (July 2006). RFC 4571.

[20] J. Lennox, M. Westerlund, Q. Wu, and C. S. Perkins. 2017. Sending Multiple RTP Streams in a Single RTP Session. Internet Engineering Task Force. (March 2017). https://doi.org/10.17487/RFC8108 RFC 8108.

[21] A. Li. 2007. RTP Payload Format for Generic Forward Error Correction. Internet Engineering Task Force. (December 2007). RFC 5109.

[22] I. Lubashev. 2018. Partially Reliable Message Streams for QUIC. Internet Engineering Task Force. (May 2018). draft-lubashev-quic-partial-reliability-03.

[23] D. McGrew and E. Rescorla. 2010. Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP). Internet Engineering Task Force. (May 2010). RFC 5764.

[24] S. McQuistin, C. S. Perkins, and M. Fayed. 2016. Implementing Real-Time Transport Services over an Ossified Network. In *Proceedings of the Applied Networking Research Workshop*. ACM/IRTF/ISOC, Berlin, Germany, 81–87. https://doi.org/10.1145/2959424.2959443

[25] J. Ott, S. Wenger, N. Sato, C. Burmeister, and J. Rey. 2006. Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF). Internet Engineering Task Force. (July 2006). RFC 4585.

[26] T. Pauly, E. Kinnear, and D. Schinazi. 2018. An Unreliable Datagram Extension to QUIC. Internet Engineering Task Force. (September 2018). draft-pauly-quic-datagram-00.

[27] G. Pelletier and K. Sandlund. 2008. RObust Header Compression Version 2 (ROHCv2): Profiles for RTP, UDP, IP, ESP and UDP-Lite. Internet Engineering Task Force. (April 2008). RFC 5225.

[28] C. Perkins and V. Singh. 2017. Multimedia Congestion Control: Circuit Breakers for Unicast RTP Sessions. Internet Engineering Task Force. (March 2017). RFC 8083.

[29] C. S. Perkins, I. Kouvelas, O. Hodson, V. Hardman, M. Handley, J.-C. Bolot, A. Vega-García, and S. Fosse-Parisis. 1997. RTP Payload for Redundant Audio Data. Internet Engineering Task Force. (September 1997). RFC 2198.

[30] M. Petit-Huguenin and G. Salgueiro. 2016. Multiplexing Scheme Updates for Secure Real-time Transport Protocol (SRTP) Extension for Datagram Transport Layer Security (DTLS). Internet Engineering Task Force. (September 2016). RFC 7983.

[31] J. Rey, D. Leon, A. Miyazaki, V. Varsa, and R. Hakenberg. 2006. RTP Retransmission Payload Format. Internet Engineering Task Force. (July 2006). RFC 4588.

[32] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. 2002. SIP: Session Initiation Protocol. Internet Engineering Task Force. (June 2002). RFC 3261.

[33] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. 2003. RTP: A Transport Protocol for Real-Time Applications. Internet Engineering Task Force. (July 2003). RFC 3550.

[34] H. Schulzrinne, A. Rao, and R. Lanphier. 1998. Real Time Streaming Protocol (RTSP). Internet Engineering Task Force. (April 1998). RFC 2326.

[35] V. Singh, S. McQuistin, M. Ellis, and C. S. Perkins. 2013. Circuit Breakers for Multimedia Congestion Control. In *Proceedings of the International Packet Video Workshop*. IEEE, San Jose, CA, USA. https://doi.org/10.1109/PV.2013.6691439

[36] T. Stockhammer. 2011. Dynamic Adaptive Streaming Over HTTP – Standards and Design Principles. In *Proceedings of the Conference on Multimedia Systems (MMSys)*. ACM, San Jose, CA, USA, 133–143. https://doi.org/10.1145/1943552.1943572

[37] P. Tiesel, M. Palmer, B. Chandrasekaran, A. Feldman, and J. Ott. 2017. Considerations for Unreliable Streams in QUIC. Internet Engineering Task Force. (October 2017). draft-tiesel-quic-unreliable-streams-01.

[38] B. Trammell, C. S. Perkins, and M. Kühlewind. 2017. Post Sockets: Towards an Evolvable Network Transport Interface. In *Proceedings of the International Workshop on Future of Internet Transport*. IFIP, Stockholm, Sweden. https://doi.org/10.23919/IFIPNetworking.2017.8264874

[39] International Telecommunication Union. 2009. Packet-based multimedia communications systems. (December 2009). ITU-T Rec. H.323.

[40] M. Westerlund, B. Burman, C. S. Perkins, A. Alvestrand, and R. Even. 2018. Guidelines for using the Multiplexing Features of RTP to Support Multiple Media Streams. Internet Engineering Task Force. (July 2018). draft-ietf-avtcore-multiplex-guidelines-06.

[41] M. Westerlund, C. S. Perkins, and J. Lennox. 2015. Sending Multiple Types of Media in a Single RTP Session. Internet Engineering Task Force. (December 2015). draft-ietf-avtcore-multi-media-rtp-session-13.

[42] X. Zhu, P. Pan, S. Mena, P. Jones, J. Fu, and S. D'Aronco. 2018. NADA: A Unified Congestion Control Scheme for Real-Time Media. Internet Engineering Task Force. (August 2018). draft-ietf-rmcat-nada-09.