QUIC
Internet-Draft
Intended status: Informational
Expires: November 30, 2018

I. Lubashev
Akamai Technologies

May 29, 2018

Partially Reliable Message Streams for QUIC
draft-lubashev-quic-partial-reliability-03

Abstract

This memo introduces a new EXPIRED_STREAM_DATA frame to enable partial reliability for QUIC streams. The EXPIRED_STREAM_DATA frame allows a sender to give up on retransmitting older parts of a stream and to notify the receiver about this decision. The content of this draft is intended for merging into QUIC transport, recovery, and applicability drafts as a negotiable extension and/or QUIC Version 2 transport feature.

Table of Contents

1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Introduction

Some applications, especially applications with near real-time requirements, need transport that supports partially reliable streams - streams that deliver bytes in order but allow for applicaiton-controlled gaps. These applications communicate using application-specific messages that are serialized over QUIC streams. Applications desire partially reliable streams when their messages expire and lose their usefulness due to later events (time passing, newer messages, etc).

Examples of applications that can benefit from partially reliable streams are real time video (all prior data is to be expired when a new key frame is available) and data replication (expire previous updates, when a new update overwrites the data).

The content of this draft is intended for [I-D.ietf-quic-transport], [I-D.ietf-quic-recovery] and, [I-D.ietf-quic-applicability] as a QUIC extension and/or QUIC Version 2.

2.1. Stream-per-Message Alternative

It is possible to avoid the need for partially reliable streams by encoding one message per QUIC stream. When a message expires, the sender can reset the stream, causing RST_STREAM frame to be transmitted, unless all data in the stream has already been fully acknowledged. Likewise, the receiver can send STOP_SENDING frame to indicate its disinterest in the message. The problem with this approach is that messages transmitted by the application typically belong to a message stream, and applications may need to support multiple concurrent message streams. Hence, a message-per-stream approach requires each message to contain an extra header portion to associate the message with a logical application stream. In case of short messages, this approach introduces a significant overhead due to STREAM frames and message headers. It also places the burden on the application to reorder data arriving on multiple QUIC streams. Furthermore, splitting each application stream into multiple QUIC streams renders QUIC's per-stream flow control ineffective and requires an application to build its own.

2.2. Partially Reliable Message Streams

The proposed single-stream mechanism keeps aplication messages arriving in order on a single stream, while allowing the application to control message expiration.

The key to partially reliabile message streams is notifying the receiver about data that will not be retransmitted and ensuring that the receiver can identify the beginning of each new message.

It is important to note that the proposed protocol does not guarantee that data is read by the receiver application at the stream offsets written to by the sender application.

2.3. Minimum retransmittable offset and smallest receive offset

For fully reliable streams, the smallest unacknowledged data offset is treated by the sender to be the minimum retransmittable offset. Likewise, the smallest receive offset for a stream is the smallest

data offset that has not been received by the receiver. Due to loss and reordering, the smallest receive offset may be smaller than the largest received offset.

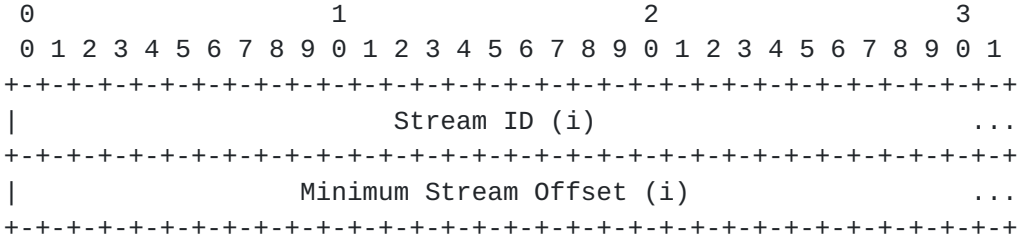
Partially reliable streams allow the sender to advance its minimum retransmittable offset and notify the receiver to advance its smallest receive offset.

3. EXPIRED_STREAM_DATA Frame

The EXPIRED_STREAM_DATA frame (type=0x??) is used by a sender to inform a receiver of the minimum retransmittable offset (Section 2.3) for a stream.

An endpoint that receives an EXPIRED_STREAM_DATA frame for a send-only stream MUST terminate the connection with error PROTOCOL_VIOLATION.

The frame is as follows:



The fields in the EXPIRED_STREAM_DATA frame are as follows:

Stream ID: The stream ID of the stream that is affected encoded as a variable-length integer.

Minimum Stream Offset: A variable-length integer indicating the minimum offset of the stream data that will sent (or re-transmitted) on the identified stream, in units of octets.

Since Stream 0 MUST be reliable, Stream ID MUST NOT be 0.

Upon receipt of an EXPIRED_STREAM_DATA frame, the receiver advances the smallest receive offset for the stream (Section 2.3) to be the Minimum Stream Offset value.

The sender MUST NOT reduce the minimum retransmittable offset for a stream, but loss and reordering can cause EXPIRED_STREAM_DATA frames to be received out of order. EXPIRED_STREAM_DATA frames that do not advance the smallest receive offset for the stream MUST be ignored.

It is possible for the smallest receive offset to become larger than the largest received offset a the stream. Receipt of an EXPIRED_STREAM_DATA does not advance the largest received offset for the stream.

4. Sender Interface and Behavior

QUIC library interface needs provide a way for a sender to expire data previously written to the transport by updating the minimum retransmittable offset (Section 2.3) for a stream. A typical sender would call this API function whenever data previously enqueued for transmission expires, per application semantics. The sender would keep track of the message boundaries and request expiration of data on a message boundary.

4.1. Communicating Message Boundary

To allow a sender application to expire stream data written to the transport but never sent to the receiver, the sender transport needs to create a gap between data previously sent on the stream and data to be sent after the expiration point. The gap ensures that the receiver does not deliver subsequent octets to the application until the receipt of the EXPIRED_STREAM_DATA frame, in case packets containing the EXPIRED_STREAM_DATA frame and subsequent STREAM frame are reordered.

To avoid complicated connection flow control accounting (see version 02 of this draft [1]), a single octet gap is used for communicating the message boundary. Sender's EXPIRED_STREAM_DATA frame extends the minimum stream offset past that gap. Upon receipt of the EXPIRED_STREAM_DATA frame, the receiver is able to notify the application of a gap, which allows the application to identify the beginning of a new message.

4.2. Translating Application Offsets to QUIC Offsets

Since the QUIC library and the application need to communicate data offsets (for example, for the purpose of updating the minimum retransmittable stream offset), the QUIC library needs to translate appliction offsets to QUIC offsets. Depending on the richness of the APIs exposed to the application, keeping a single difference between the current application and QUIC offsets is likely to be sufficient.

4.3. Sender Behavior

This section discusses sender behavior in terms of QUIC offsets, and the translation from applicatoin offsets (see Section 4.2) is implicit.

When an application instructs its QUIC transport to advance the minimum retransmittable offset for a stream, and there is any unacknowledged data (including unsent data) at an offset smaller than the new minimum retransmittable offset, the sender SHOULD transmit an EXPIRED_STREAM_DATA frame (Section 3), except as provided for in Section 4.3.1.

- When the new minimum retransmittable offset is less than or equal to the current send offset, the Minimum Stream Offset field in the EXPIRED_STREAM_DATA frame is set to the new minimum retransmittable offset.

- When the new minimum retransmittable offset is larger the current send offset, the Minimum Stream Offset field in the EXPIRED_STREAM_DATA frame is set to the current send offset plus 1, and stream data starting at the new minimum retransmittable offset is henceforth sent starting at the current send offset plus 1 (which becomes the new minimum retransmittable offset). Hence, it may be possible for a minimum retransmittable offset to become larger than the current send offset for a stream.

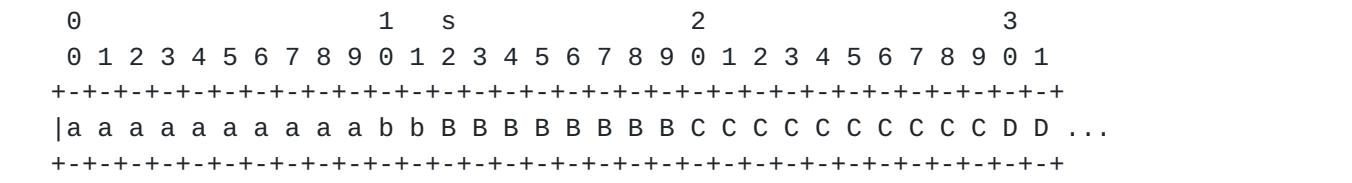
4.3.1. Coalescing Minimum Retransmittable Offset Updates

When an application instructs its QUIC transport to advance the minimum retransmittable offset for a stream, but the current send offset is not larger than the minimum retransmittable offset specified in the _previous_ call to this API function, the current stream offset is not advanced and an EXPIRED_STREAM_DATA frame is not sent. Stream data starting at the requested minimum retransmittable offset is henceforth sent starting at the previous minimum retransmittable offset (which remains the minimum retransmittable offset for the stream).

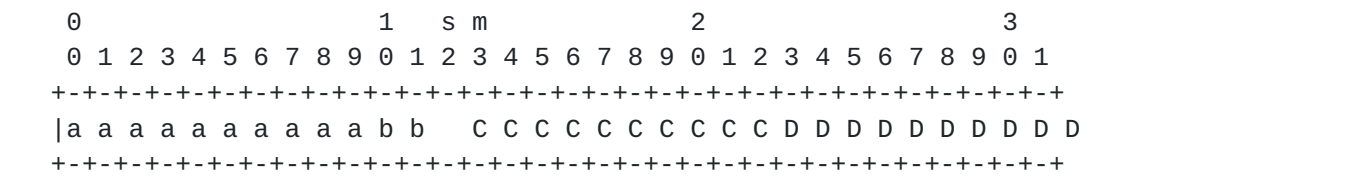
Note that the coalescing rule does not apply (the EXPIRED_STREAM_DATA frame _is_ sent) if the very first message has expired before any of its octets have been transmitted. This allows the receiver to always ascertain the location of any gaps in messages it is receiving.

4.3.2. Example

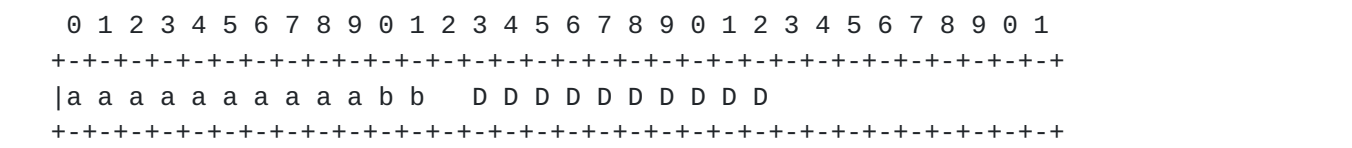
For example, an application wrote four 10-octet messages (A, B, C, D) to the transport, and the current send offset (the next offset to be sent) is 12. In this example, the upper-case indicates bytes to be sent, while the lower-case indicates bytes already sent.



When the application desires to expire messages A and B, it requests the minimum retransmittable offset to be 20. The transport then sends an EXPIRED_STREAM_DATA frame with Minimum Stream Offset field set to 13, and the subsequent STREAM frame would send message C starting at stream offset 13.



However, if the application requestes to expire octets corresponding to message C before any subsequent STREAM frames could be sent, no new EXPIRED_STREAM_DATA frame is sent, and the subsequent STREAM frame would send message D starting at stream offset 13.



5. Receiver Interface and Behavior

Upon receipt of an EXPIRED_STREAM_DATA frame (Section 3), the receiver SHOULD assume that none of the data before the new smallest receive offset (Section 2.3) will be retransmitted. A receiver SHOULD discard any stream data received for an offset smaller than the new smallest receive offset, possibly advancing the largest received offset for the stream. Discarding such data ensures that when the application observes a gap in the data stream, what follows the gap is a beginning of a new message.

It is recommended that a QUIC library API provides a way for the receiver application to learn of the presence of a gap in the data stream, indicating that the data that follows the gap is a beginning of a new message.

6. Retransmission of EXPIRED_STREAM_DATA

The most recent EXPIRED_STREAM_DATA frame (Section 3) for a stream MUST be retransmitted if it is declared lost, until the sender is certain that the receiver is not expecting retransmission of any expired data. I.e. the frame MUST be retransmitted until the stream enters "half-closed (local)" state, or all data between the largest Minimum Stream Offset field in an acknowledged EXPIRED_STREAM_DATA frame and the current minimum retransmittable offset (Section 2.3) has been acknowledged.

7. IANA Considerations

This document has no actions for IANA.

8. Security Considerations

This document has no new security considerations.

9. Change Log

9.1. Since version 00

- Fixed flow control to disallow other streams to use connection credits designated for skipping expired bytes.

9.2. Since version 01

- Added an ability by the receiver as well as the sender to control partial reliability of QUIC streams.

- Added Exempt Stream Bytes value and updated connection flow control calculation to use Exempt Stream Bytes value.

- Replaced the Min Stream Offset value with the existing values: "min retransmittable offset" (for sender) and "smallest receive offset" (for receiver). (Section 2.3)

- Changed MIN_STREAM_DATA frame to be a receiver-transmitted frame.

- Added sender-transmitted EXPIRED_STREAM_DATA frame. (Section 3)

9.3. Since version 02

- Significantly simplifed the proposal by treating the stream as a message stream, allowing for data offsets not to be preserved between the sender and the receiver.

- Reverted to sender-only transport-level control of message expiration.

- Removed the need for Exempt Stream Bytes and changes to connection flow control accounting.

- Removed MIN_STREAM_DATA frame.

10. Acknowledgments

Many thanks to Mike Bishop, Ian Swett, and Subodh Iyengar for their reviews, feedback, and ideas. Thus draft would not happen without their input. Kudos to the QUIC working group for a mountain of feedback on this draft and for diligently plowing through hard problems, making thousands of big and small decisions, to make the Internet better for everyone.

11. References

11.1. Normative References

[I-D.ietf-quic-applicability]
Kuehlewind, M. and B. Trammell, "Applicability of the QUIC Transport Protocol", draft-ietf-quic-applicability-01 (work in progress), October 2017.

[I-D.ietf-quic-recovery]
Iyengar, J. and I. Swett, "QUIC Loss Detection and Congestion Control", draft-ietf-quic-recovery-12 (work in progress), May 2018.

[I-D.ietf-quic-transport]
Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-12 (work in progress), May 2018.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

11.2. URIs

[1] https://tools.ietf.org/html/draft-lubashev-quic-partial-reliability-02

Author's Address

Igor Lubashev
Akamai Technologies

EMail: igorlord@alum.mit.edu