Coding for QUIC

Ian Swett, Marie-Jose Montpetit, Vincent Roca

nwcrg
Internet-Draft
Intended status: Informational
Expires: December 23, 2018

I. Swett
Google
M-J. Montpetit
Triangle Video
V. Roca
INRIA

June 21, 2018

Coding for QUIC
draft-swett-nwcrg-coding-for-quic-01

Abstract

This document focusses on the integration of FEC coding in the QUIC transport protocol, in order to recover from packet losses. This document does not specify any FEC code but defines mechanisms to negotiate and integrate FEC Schemes in QUIC. By using proactive loss recovery, it is expected to improve QUIC performance in sessions impacted by packet losses. More precisely it is expected to improve QUIC performance with real-time sessions (since FEC coding makes packet loss recovery insensitive to the round trip time), with multicast sessions (since the same repair packet can recover several different losses at several receivers), and with multipath sessions (since repair packets add diversity).

Table of Contents

1. Introduction

QUIC is a new transport that aims at improving network performance by enabling out of order delivery, partial reliability, and methods of recovery besides retransmission, while also improving security. This document specifies a framework to enable FEC codes to be used to

recover from lost packets within a single QUIC stream or across several QUIC streams.

The ability to add FEC coding in QUIC may be beneficial in several situations:

- for a robust transmission of latency sensitive traffic, for instance real-time flows, since it enables to recover packet losses independently of the round trip time;
- for the transmission of contents to a large set of QUIC reception endpoints, since the same repair frame may help recovering several different packet losses at different receivers;
- for multipath communications, since repair traffic adds diversity.

This framework does not mandate the use of any specific FEC code (i.e., how to encode and decode) nor FEC Scheme (i.e., that specifies both a FEC code and how to use it, in particular in terms of signaling). Instead it allows to negotiate the FEC Scheme to use at session startup, assuming that more than one solution could potentially be offered concurrently. Without loss of generality, we assume that the encoding operations compute a linear combination of QUIC packets, regardless of whether these codes are of block type (as with Reed-Solomon codes [RFC5510]) or sliding window type (as with RLC codes [RLC]).

2. Definitions and Abbreviations

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].

Terms and definitions that apply to coding are available in [nc-taxonomy]. More specifically, this document uses the following definitions:

Packet versus Symbol: a Packet is the unit of data that is exchanged over the network while a Symbol is the unit of data that is manipulated during the encoding and decoding operations

Source Symbol: a unit of data originating from the source that is used as input to encoding operations

Repair Symbol: a unit of data that is the result of a coding operation

This document uses the following abbreviations:

E: size of an encoding symbol (i.e., source or repair symbol), assumed fixed (in bytes)

3. General Design Considerations

This section lists a few general considerations that govern the framework for FEC coding support in QUIC.

3.1. FEC Code versus FEC Scheme, Block Codes versus Sliding Window Codes

A FEC code specifies the details of encoding and decoding operations. In addition to that, a FEC Scheme defines the additional protocol aspects required to use a particular FEC code [nc-taxonomy]. In particular the FEC Scheme defines signaling (e.g., information contained in Source and Repair Packet header or trailers) needed to synchronize encoders and decoders.

Block coding (e.g., Reed-Solomon [RFC5510]) and sliding window coding (e.g., RLC [RLC]) are two broad classes of FEC codes [nc-taxonomy]. In the first case, the input flow must be first segmented into a sequence of blocks, FEC encoding and decoding being performed independently on a per-block basis. In the second case rely, a sliding encoding window continuously slides over the input flow. It is envisioned that the two classes of codes could be used to bring FEC protection to QUIC, usually with an advantage for sliding window codes when it comes to low latency communications.

3.2. FEC Scheme Negotiation

There are multiple FEC Scheme candidates. Therefore a negotiation step is needed to select one or more codes to be used over a QUIC session. This will be implemented using the one step negotiation of the new QUIC negotiation mechanism [QUIC-transport], during the QUIC handshake.

Editor’s notes:

* It is likely that FEC Scheme negotiation requires the use of a new dedicated Extension Frame Type. To Be Clarified and text updated.

* It is not clear whether negotiation is meant to select a **single** FEC Scheme or **multiple** FEC Schemes. In the second case (multiple FEC) it is required to have a complementary mechanism to indicate which FEC Scheme is used in a given REPAIR frame (which could be done through as many

REPAIR frame type values as potential FEC Scheme negotiated). Is it what we want to achieve? Not sure.

* It is not clear whether negotiation is carried out at QUIC level (and therefore for multiple streams) or at a stream level (and therefore multiple streams may use multiple FEC Schemes). The terminology used above should be updated to reflect the choice.

3.3. FEC Protection Within an Encrypted Channel

FEC encoding is applied before any QUIC encryption and authentication processing. Source symbols, that constitute the data units used by the FEC codec, contain cleartext application data.

3.4. About Middleboxes

The coding approach described in this document does not allow on path elements (middleboxes) to take part in FEC protection. The traffic being encrypted end-to-end, the middleboxes are not in position to perform FEC decoding, nor to add any redundant traffic.

3.5. FEC Protection at the Stream Level

Streams in QUIC provide a lightweight, ordered byte-stream abstraction. FEC encoding is applied at the stream level, within a single stream or across two or more streams of the same QUIC session. This is motivated by the fact that FEC protection is not necessarily beneficial to all data streams, but only to a subset of them. For instance FEC protection can be highly beneficial to live video streams to which the proactive erasure correction feature of FEC, independent of the RTT, should be highly beneficial. On the opposite, FEC protection is probably less attractive for latency insensitive bulk unicast flows.

In order to facilitate experiments, and in order to enable backward compatibility, the STREAM frames that carry application data are kept unmodified. On the opposite, frames that carry one or more repair symbols use a dedicated REPAIR frame type, chosen within the type range dedicated to "Extension Frames".

3.6. About Gaps in the Set of Source Symbols Considered During Encoding

A given FEC Scheme MAY support or not the presence of gaps in the set of source symbols that constitute a block (for Block codes) or an encoding window (for Sliding Window codes). A potential cause for non contiguous sets of source symbols is the acknowledgment of one of them. When this happens, the QUIC sending endpoint may want to

remove this source symbol from further FEC encodings. This is particularly true with Sliding Window codes because of their flexibility during FEC encoding (i.e., the encoding window can change between two consecutive FEC encodings).

Supporting gaps can be motivated by the desire to reduce encoding and decoding complexity since there are fewer variables. However this choice has major consequences in terms of signaling. Indeed each repair symbol transmitted MUST be accompanied with enough information for the QUIC decoding endpoint to unambiguously identify the exact composition of the block or encoding window. Without any gap, the identity of the first source symbol plus the number of symbols in the block or encoding window is sufficient. With gaps, a more complex encoding needs to be used, perhaps similar to the encoding used for selective acknowledgments.

Whether or not gaps are supported MUST be clarified in each FEC Scheme.

4. FEC Scheme Negotiation in QUIC

FEC Scheme negotiation has two goals:

- Selecting a FEC Scheme (or FEC Schemes) that can be used by the QUIC transmission and reception endpoints. This process requires an exchange between them;

- Communicating a certain number of parameters, the "Configuration Information", that are not expected to change over the session lifetime. For instance, this is the case of the symbol size parameter, E (in bytes), that needs either to be agreed between the endpoints, or chosen by the sender and communicated to the receiver(s);

Editor’s notes:

  * It is likely that FEC Scheme negotiation requires the use of a new dedicated Extension Frame Type. The details remain TBD.

  * The Negotiation Frame Type format remains TBD.

  * How to communicate the parameters remains TBD.

  * The present document only provides high level principles, the details are of course the responsibility of the FEC Scheme.

* In case negotiation is different when protecting a single versus several streams, this section may be moved to the respective sections.

* How does it work in case of a multicast session?

* Do we negotiate here a FEC Scheme on a per-Stream basis (or group of Streams to be protected jointly)? Or do we negotiate a FEC Scheme on a QUIC session basis, therefore to be used for all the Streams that need FEC protection?

4.1. FEC Scheme Selection Process

Let us consider the FEC Scheme selection process between the QUIC endpoints. Figure 1 illustrates the principle when a QUIC reception endpoint initiates the exchange.

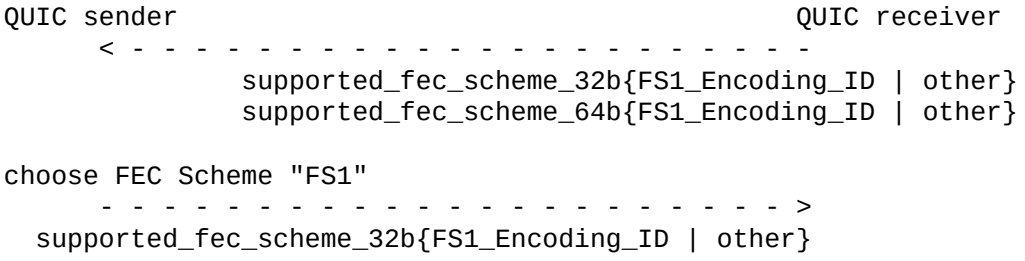

Figure 1: Example FEC Scheme selection process, during the initial negotiation.

The supported_fec_scheme_16b and supported_fec_scheme_32b are two new TransportParameterId to be added to the "Table 7: Initial QUIC Transport Parameters Entries" Section 13.1, of [QUIC-transport]. The supported_fec_scheme_32b contains a 32-bit data field to carry opaque 32-bit value, while the supported_fec_scheme_64b contains a 64-bit data field to carry opaque 64-bit value (see Section 4.2).

4.2. FEC Scheme Configuration Information

Let us now focus on the communication of configuration information specific to the selected FEC Scheme. In Figure 1, the supported_fec_scheme_32b{FS1_Encoding_ID} contains a field meant to carry the FEC Encoding ID of the FEC Scheme selected plus addditional configuration information if any. If a 32 bit opaque field is not sufficient, the supported_fec_scheme_64b can be used instead and proposes a 64 bit opaque field.

5. Procedures when Protecting a Single QUIC Stream

This section focusses on the simple case where FEC protection is applied to a single QUIC stream. We consider a unidirectional data flow between a QUIC sending endpoint and one (or more) QUIC reception endpoints.

5.1. Application data, STREAM Frame data and Source Symbols

Application data is kept in a transmission buffer at a QUIC sending endpoint, and sent within STREAM frames. Each STREAM frame that carries data contains an Offset field that indicates the offset within the stream of the first byte of the Stream Data field, as well as a Length field that indicates the number of bytes contained in the Stream Data field. Upon receiving a STREAM frame, using the Offset and Length fields, a QUIC reception endpoint can easily store data in its reception buffer. But since a QUIC Packet may be lost during transmission, the reception buffer may have gaps.

Figure 2 illustrates how source symbols are mapped to the QUIC transmission or reception buffers (same principle on either side). Since any source (and repair) symbol is of fixed size (E bytes) for a given stream, since QUIC guaranties that the first byte in the stream has an offset of 0, the position of each source symbol is known by both ends.

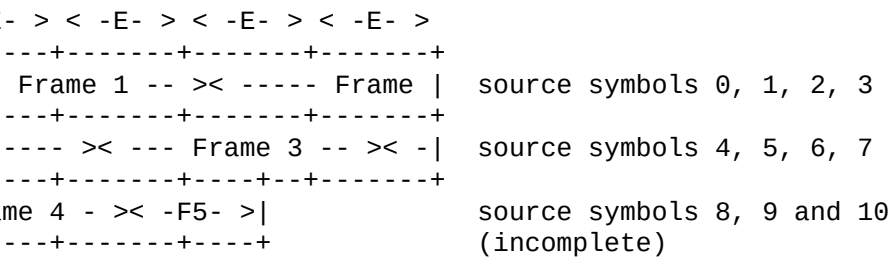

Figure 2: Example of source symbol mapping, when the E value is relatively small.

Any value for E is possible, from a single byte to several hundreds or thousands of bytes. In general, the source symbols are not aligned with data chunks sent in the STREAM frames. A given STREAM frame may carry all the bytes of a given source symbol. But when a source symbol straddles two or more (e.g., if E is large compared to usual frame size) STREAM frames, a proper reception of these two (or more) STREAM frames is needed for a QUIC reception endpoint to consider that the source symbol is available for FEC decoding operations. The choice of an appropriate value for E may depend on the use case (in particular on the nature of application data). A

reasonably small value reduces the probability that a source symbol straddles two or more STREAM frames, a situation that is considered as potentially harmful (the unit of control, the source symbol, and unit of transmission, the frame, are not aligned). However an overly small value also increases processing complexity (FEC encoding and decoding are performed over a larger linear system) so there is an incentive to use a larger value. An appropriate balance should be found, and this choice is considered as out of scope for this document.

5.2. Signaling Considerations within STREAM and REPAIR Frames

Once the initial negotiation succeeded and an appropriate FEC Scheme has been chosen between the QUIC endpoints, data is exchanged as follows. Source data is transmitted within STREAM frames, as would happen without any FEC based loss recovery mechanism (in particular without considering source symbols boundaries). Repair data, computed during FEC encoding, on the opposite, is sent within a dedicated REPAIR frame type, chosen within the type range dedicated to "Extension Frames". In both cases, the same Stream ID is used since both flows relate to the same stream.

The REPAIR frame format is FEC Scheme dependent. The document specifying a FEC Scheme to be used with QUIC MUST define the REPAIR frame format, among other things. The REPAIR frame MUST carry enough information for a QUIC reception endpoint to understand exactly how this repair symbol(s) has(ve) been generated. It implies that each REPAIR symbol MUST communicate the block (with block codes) or encoding window (with Sliding Window codes) composition. This MAY be achieved by communicating in case there is no gap in the source symbol set (see XXX):

- the offset of the first source symbol of the block or encoding window;
- the number of source symbols in the block or encoding window, which can be either a number of symbols or a number of bytes since symbols are of fixed size, E.

Note that unlike FEC Schemes for FLUTE/ALC, NORM, and FECFRAME, here there is no notion of Encoding Symbol Id (ESI), an identifier managed in a sequential manner to identify source and repair symbols. The use of an offset within the stream, with the guaranty that no wrapping to zero can occur, provides an alternative mechanism to identify any source symbol.

As explained above, source data is transmitted without any modification, which provides backward compatibility. This is

advantage in situations where the same QUIC stream is delivered to several QUIC reception endpoints (multicast): it may be appropriate to select a given FEC Scheme even if it is known that a subset of the QUIC reception endpoints do not support it.

Editor's notes:

* This I-D proposes to define a single generic REPAIR frame type, but an alternative could be to have a one-to-one mapping between a REPAIR frame type and a specific FEC Scheme.

* The use of frame type within the Extension Frames range for REPAIR frames is meant to facilitate experimentations. If the use of coding in QUIC is recognized as having benefits, a dedicated (or more, see above) frame type could be selected later on.

5.3. Management of Silent Periods and End of Stream

If an application does not submit fresh data for some time, the last source symbol may not be totally filled. It follows that this last source symbol cannot be considered during FEC encoding and therefore the associated bytes of the application stream are not protected. A similar problem arrives when a stream is finished, the application having no more data to submit to QUIC. Here also, the bytes of the last incomplete source symbol are not protected by FEC encoding.

In order to solve this problem, it is RECOMMENDED that a QUIC sending endpoint:

o Identifies when such a situation is likely to occur, for instance by waiting no more than a certain time during an application silent period;

o Upon time-out, the application falls back to the alternative re-transmission based loss recovery mechanism for the bytes of the last incomplete source symbol;

Editor's notes: Clearly, the above mechanism requires more thoughts as well as experimental work. The "end of stream" situation may be addressed through zero padding perhaps easily. However the use of zero padding for transitory silent periods may add a lot of specification and implementation complexity...

6. Procedures when Protecting Several QUIC Streams

This section focusses on the general case where FEC protection is globally applied across two or more QUIC streams.

Editor's notes: It is not clear whether this use-case is needed. It adds specification and implementation complexity that need to be balanced with the expected benefits.

* Receiver: A first complexity comes from the requirement to identify to which stream a decoded source symbol belongs to. This is also one of the main difficulty for FECFRAME (both with block and sliding window codes) which required to distinguish an ADU (submitted by the application) from an ADUI (the same ADU plus an additional FlowID among other things). Do we want this level of complexity?

* Sender: Another complexity comes from the encoding window management at a sender. With multiple streams, shifting the encoding window to the right needs to be done based on timestamps associated to source symbols of the various streams: the oldest source symbol across all the streams will be removed.

* When two largely different streams are protected togethers (e.g., a high definition 4K video flow plus the associated relatively low-rate audio stream), is this extra complexity balanced by significant performance improvements compared to an independent protection on each stream (intuition is yes, the low bitrate flow is better protected iff the encoding window is large enough)? And when the various streams have a comparable bitrate? More work (incl. experimental work) is needed to answer this question.

6.1. Application data, STREAM Frame data and Source Symbols

Within each stream, the source symbols MUST be defined as in the simple case of a single stream. Figure 2 remains valid.

6.2. Block or Encoding Window Management

The details of how to create the block or encoding window are specific to the FEC Scheme. A possible approach is the following.

When creating the block (block FEC code) or encoding window (sliding window FEC code), the source symbols to consider of each stream are appended. All the relevant source symbols of the first stream are appended, followed by all the source symbols of the second stream,

etc. These sequences do not follow any timing consideration in order to simplify signaling.

Figure 3 illustrates, in case of a Sliding Window FEC Scheme, an encoding window with source symbols belonging to two streams, of Stream ID 120 and 51 respectively.

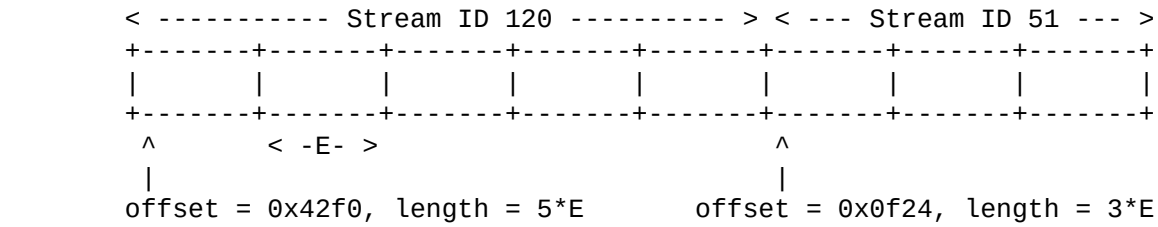

Figure 3: Example of encoding window of a Sliding Window FEC Scheme and FEC protection across two streams.

6.3. Signaling Considerations within STREAM and REPAIR Frames

Source data on each stream is transmitted within STREAM frames, as would happen without any FEC based loss recovery mechanism.

Repair symbols, generated during FEC encoding as a linear combination of source symbols that belong to one or more of the streams, are transmitted within REPAIR frames. Each REPAIR frame can be associated to any of the input streams it protects, and therefore associated to any of the associated Stream IDs.

Editor's notes: Check that indeed, with FEC protection across several streams, assigning a REPAIR frame to any of the streams it protects is meaningful. Should an approach for selecting one stream (and Stream ID) be preferred?

The REPAIR frame format is FEC Scheme dependent and MUST be defined by document specifying a FEC Scheme. One of the key information of this REPAIR frame is the composition of the block (with block codes) or encoding window (with sliding window codes) used to perform FEC encoding. Indeed, this is the only manner to convey this information since an application flow is not predictable (e.g., if an application flow is momentarily suspended, the composition of the block or encoding window will be affected). One possibility is to list, in each REPAIR frame header:

- the actual number of streams considered (the maximum number is known after the negotiation step, but if one of the streams remains silent for some time, it may not contribute during

encoding and therefore be absent from the block or encoding window);

- for each stream concerned, its Stream ID, the offset of the first source symbol considered as well as the length, i.e., the number of bytes considered.

This approach does not enable to keep track of the source symbol ordering across streams, but enables a non ambiguous description of the encoding window.

The FEC Scheme specification MUST also detail how to manage the block or encoding window. For instance, should the oldest source symbol of any stream be removed from the encoding window when this latter is shifted to the right? This would mean that a timestamp is attached to each source symbol in order to identify the oldest one across all streams.

7. Security Considerations

TBD

8. IANA Considerations

TBD

9. Acknowledgments

TBD

10. References

10.1. Normative References

[QUIC-transport] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-12 (work in progress), May 2018, <https://datatracker.ietf.org/doc/draft-ietf-quic-transport/>.

[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>.

10.2. Informative References

[nc-taxonomy]
Roca et al., V., "Taxonomy of Coding Techniques for Efficient Network Communications", draft-irtf-nwcrg-network-coding-taxonomy (Work in Progress) (work in progress), March 2018, <https://datatracker.ietf.org/doc/draft-irtf-nwcrg-network-coding-taxonomy/>.

[RFC5510] Lacan, J., Roca, V., Peltotalo, J., and S. Peltotalo, "Reed-Solomon Forward Error Correction (FEC) Schemes", RFC 5510, DOI 10.17487/RFC5510, April 2009, <https://www.rfc-editor.org/info/rfc5510>.

[RLC] Roca, V., "Sliding Window Random Linear Code (RLC) Forward Erasure Correction (FEC) Scheme for FECFRAME", Work in Progress, Transport Area Working Group (TSVWG) draft-ietf-tsvwg-rlc-fec-scheme (Work in Progress), May 2018, <https://tools.ietf.org/html/draft-ietf-tsvwg-rlc-fec-scheme>.

Authors' Addresses

Ian Swett
Google
Cambridge, MA
US

Email: ianswett@google.com

Marie-Jose Montpetit
Triangle Video
Boston, MA
US

Email: marie@mjmontpetit.com

Vincent Roca
INRIA
Univ. Grenoble Alpes
France

Email: vincent.roca@inria.fr