QUIC Working Group
Internet-Draft
Intended status: Standards Track
Expires: 7 March 2026

H. Zheng
Y. Liu
Alibaba Inc.

3 September 2025

FEC Extension for QUIC
draft-zheng-quic-fec-extension-01

Abstract

This document specifies a Forward Error Correction (FEC) extension for the QUIC protocol, which provide protection against packet loss.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 7 March 2026.

Copyright Notice

Table of Contents

1. Introduction

This document specifies an extension to QUIC version 1 [QUIC-TRANSPORT] to supports Forward Error Correction (FEC). FEC is a method of erasure control in data transmission by simultaneously sending redundant data (see [RFC6363]). This redundancy allows the receiver to detect and recover limited lost data without waiting for retransmission, which can be beneficial in latency-sensitive scenarios.

For instance, real-time communication frameworks like WebRTC, which is widely applied in scenarios such as video conferenceing and voice call. FEC plays an important role by recovering lost packets under unreliable network, thereby optimizing user experience.

QUIC-FEC might become a crucial foundation for MoQ (Media over QUIC), which is primarily used for video stream transmission, which can migrate the impact of packet loss on media transmission.

This document defines how FEC framework take effects in QUIC transport process. Since FEC extension introduce new QUIC frames and parameters, it requires negotiation between two endpoints.

This document does not define how the flows to be protected are determined, nor does it defines how the redundant data are calculated via FEC schemes.

For further discussion, the FEC mechanism might effect data retransmissions by notifying the remote of successfully recovered packets, thereby effectively avoiding bandwidth waste.

1.1. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

We assume that the reader is familiar with the terminology used in [QUIC-TRANSPORT]. In addition, we introduce the following FEC-related terms:

* Source symbol: piece of information exchanged by two endpoints. This document considers QUIC packets payloads as source symbols.

* Repair symbol: redundant information constructed from the combination of several source symbols.

* Erasure: loss of one or more symbols

* Erasure correction code: algorithm generating repair symbols and reconstructing missing source symbols from a set of source and repair symbols.

* Forward Erasure Correction: process of recovering erased symbols before the detection of their erasure.

* FEC scheme: the conjunction of an erasure correction code and the specific protocol elements required to use it with the design described in this document.

* Encoder: entity producing repair symbols using an erasure correction code. The encoder can be a library used by the protocol implementation or a program running in a separate process or machine.

* Decoder: entity reconstructing missing source symbols using an erasure correction code. The decoder can be a library used by the protocol implementation or a program running in a separate process or machine.

* Coding window: window of source symbols that are required to decode a repair symbol.

2. Architecture Overview

FEC Framework is designed as a module within QUIC-Transport layer. The primary target of current draft is to establish common mechanism for integrating FEC logic into the QUIC protocol, thus ensuring interoperability across different QUIC instances.

Endpoints use transport parameters in [QUIC-TRANSPORT] to negotiate whether to enable the FEC mechanism. The FEC Framework specifies necessary configuration information that needs to be exchanged between QUIC endpoints using transport parameters during the QUIC handshake process.

The FEC Framework operates within the QUIC-Transport layer. It facilitates the transmission of FEC information and redundant data using QUIC frames.

The FEC framework calls upon the FEC scheme to manage the detailed processes of FEC encoding and decoding. The FEC scheme defines detailed algorithm process, and the procedures used to identidy packet payload data in the context of FEC scheme. This draft will describe the interface specification between FEC Framework and FEC scheme.

FEC Framework MUST ensure compatibility with existing modules within the QUIC protocol. Special attention is given to ensuring interoperability with the congestion control and the ACK handling and retransmission decisions (see [QUIC-RECOVERY] and [RFC9265]).

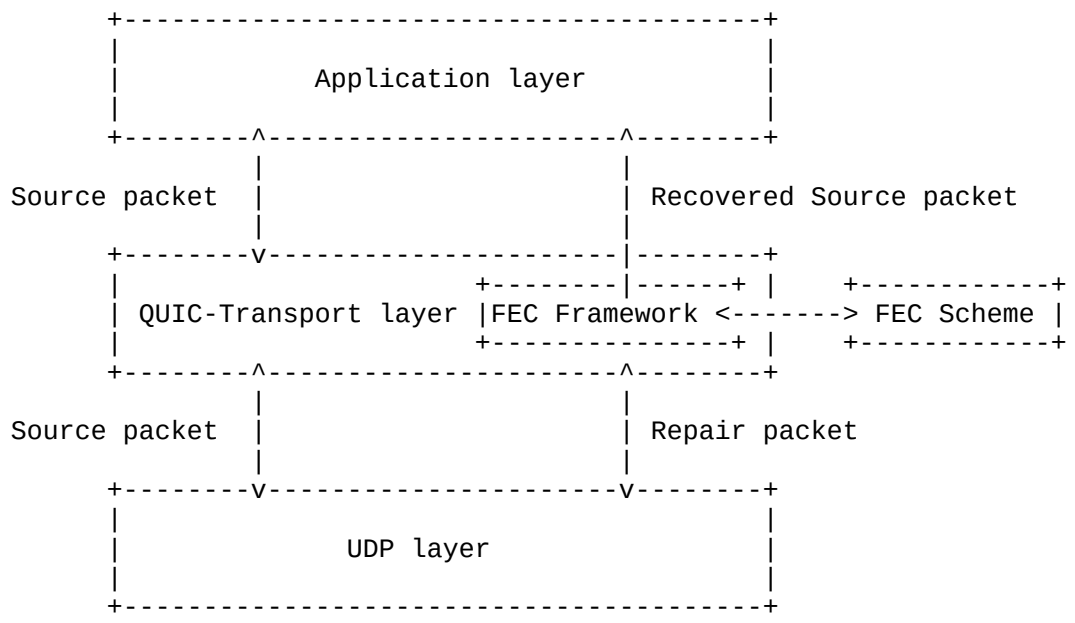

Figure 1: FEC Framework

3. Handshake Negotiation and Transport Parameter

3.1. Transport Parameter

This extension defines several new transport parameters used to negotiate the use of the FEC extension and specific FEC schemes during the connection handshake (see [QUIC-TRANSPORT]). The new transport parameters are defined as follows:

* fec_encode_schemes (current version uses 0xfece01): A list of variable-length integers specifying the FEC schemes supported by endpoints for encoding. The presence of this transport parameter indicates whether FEC functionality is enabled for encoding.

* fec_decode_schemes (current version uses 0xfecd02): A list of variable-length integers defining the FEC schemes supported by endpoints for decoding. The content of this transport parameter determines which FEC scheme is used for decoding.

Figure 2 introduce the transport parameter format of fec_encode_schemes and fec_decode_schemes:

```
FEC_ENCODER_SCHEMES TRANSPORT_PARAM{
  0xfece01 (i),
  Transport Parameter Length (i),
  Transport Parameter Value {
    FEC Encoder Schemes Length (i),
    FEC Scheme1 ID(i),
    FEC Scheme2 ID(i),
    ...
  }
}

FEC_DECODER_SCHEMES TRANSPORT_PARAM{
  0xfecd01 (i),
  Transport Parameter Length (i),
  Transport Parameter Value {
    FEC Encoder Schemes Length (i),
    FEC Scheme1 ID(i),
    FEC Scheme2 ID(i),
    ...
  }
}
```

Figure 2: Transport Parameter for FEC

* fec_max_symbol_num (current version uses 0xfecb02): A variable-length integer indicating the block size (i.e., the number of symbols in each encoding group) used in block encoding. This is an optional transport parameter and SHOULD be sent only when an endpoint uses block code schemes for encoding. For coding methods such as convolutional codes that do not require a fixed block size, this parameter SHOULD NOT be included.

For a single endpoint, FEC encoding and decoding are independent processes. By negotiating these separately, the framework can support distinct FEC schemes for encoding and decoding, or enable unidirectional encoding/decoding only. This flexibility allows FEC operations to adapt to diverse requirements and configurations.

To ensure interoperability of FEC negotiation across QUIC protocol stacks, FEC schemes MUST use standardized identifiers registered with IANA (Internet Assigned Numbers Authority). This standardization guarantees seamless communication and compatibility between implementations.

During negotiation, if a receiver’s encoder scheme matches any of the sender’s encoder schemes, the negotiation for that direction is considered successful.

3.2. Handshake Negotiation

Endpoints negotiate the transport parameters for FEC during the connection handshake, as outlined in Section 3.1.

Figure 3 illustrates an example of the negotiation process:

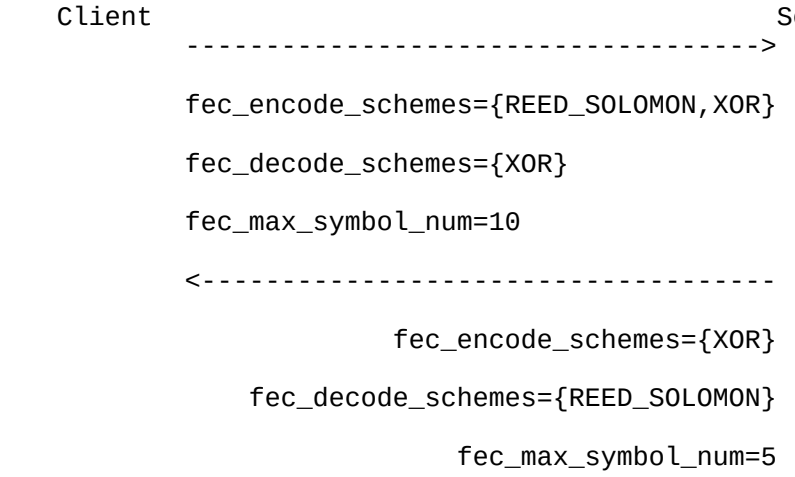

Figure 3: Handshake negotiation for FEC

Please note that in the actual process, the Client first declare the list of encoder/decoder algorithms it can handle. When the server receive the fec_encode_schemes and fec_decode_schemes, it chooses one proper scheme for each direction, and set fec_decode_schemes and fec_encode_schemes params as corresponding scheme for decoding and encoding.

Note that if the client wants to specify the FEC algorithm, it can provide only one FEC algorithm for fec negotiation process, the server could choose to support or not; otherwise, the algorithm selection SHOULD be based on the server’s priority.

For example, if the client’s FEC schemes parameter is set as follows:

* fec_encode_schemes {Algorithm1, Algorithm2, Algorithm3, ...}

* fec_decode_schemes {Algorithm1, Algorithm2, Algorithm3, ...}

The encoder/decoder algorithm used in practice is chosen and returned by the Server:

* supported_fec_encode_schemes {Algorithm1}

* supported_fec_decode_schemes {Algorithm3}

Note that the encoding/decoding option and the selected algorithm may differ after negotiation.

4. Transport mechanism

The FEC module directly participates in and influences the sending and receiving stages of packet transmission:

As the sender, the FEC Framework takes the original data frame as source symbols and provides the required source symbols to the FEC scheme module. The FEC scheme can encode the data grouped by streams, or mixed data from different streams.

FEC Framework retrieves the following information from the FEC scheme:

1. Source Symbol ID (SID) frame: SID frame is viewed as an unique identification for source symbol, which is composed of block ID and symbol index typically.

2. Repair packets: When the incoming source symbols meets the encoding conditions, the FEC scheme could generate original source symbol with repair packets. Repair packet contains FEC Payload ID, repair symbol, and optional repair key.

At the sender, FEC Framework is required to acquire and insert the SID frame into the packet that includes the source symbols. Subsequently, FEC Framework SHOULD generate and transmit repair packets to the receiver after processing a sufficient number of source symbols. Generally, the repair packets SHOULD be sent immediately following the source packets from the same block.

As the receiver, the source packets can be normally parsed by ignoring SID frame, but it might be necessary to store source symbols and identifications to perform FEC decoding after receiving repair packets from the same block.

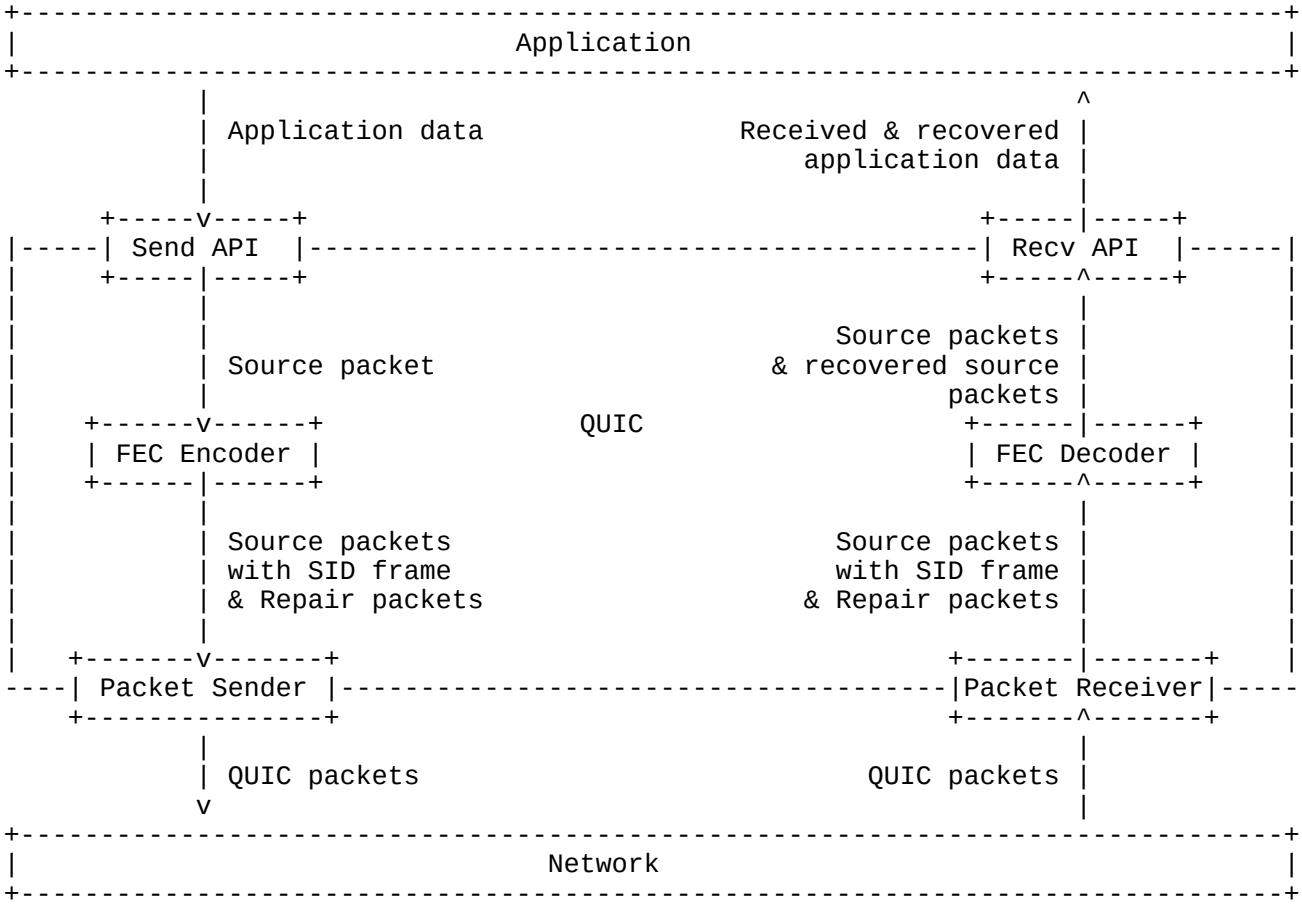

Figure 4: Work Flow for FEC

4.1. Sender Operation

Define FEC-protected QUIC payload

Endpoints can decide which types of data require FEC protection. As applying equal protection to all data types may result in inefficient bandwidth utilization, it is RECOMMENDED that protocol implementations selectively prioritize protection for critical data, such as restricting FEC to stream frames and datagram frames. For retransmitted frames or non-time-sensitive data, FEC SHOULD NOT be applied.

4.1.1. Identify source symbols using SID frame

In the context of packets that require protection through FEC (Forward Error Correction), it is necessary to define a SID (Source Identifier) frame that conveys the identity information of the data. This is intended to help the receiver clarify the following information during the decoding process:

* Which part of data insides the packet will be taken part in the FEC decode process.

* Which block and the index inside the block the source symbol belongs to.

Therefore, it’s recommended to append the SID frame after the fec-protected QUIC packet data to indicate that the data preceding this frame is identified by the current SID frame and is involved in the decoding process at the receiver.

Based on the above reason, generating and inserting the SID frame SHOULD occur before the encryption process of packets.

As the FEC module requires an additional data frame to be appended after the protected packet, developers MUST consider reserving space for the FEC data frames within the packet. This ensures that the length of the packet does not exceed the MTU (Maximum Transmission Unit) limit. Additionally, the SID MUST include Explicit Source Payload ID information to facilitate the resolution of the current data’s identity, as shown in Figure 6.

4.1.2. Generate and send repair symbols

The encoding content of the Repair package originates from the original data identified by the SID frame. It requires encoding the identity information of the FEC for the receiver to recognize the protection object of the current Repair package, as shown in Figure 6.

The specific encoding process is carried out by the FEC scheme module, which typically requires the sender to pre-set the following parameters:

* Block size: How many original packets are involved in the calculation for a computing unit, which directly affects the frequency of Repair symbol generation.

* Redundancy level of FEC: How many redundant packets can be generated from the block.

* Repair key (optional): The encoding key required by the FEC scheme module, which usually needs to be included in the Repair package for the receiver to use for decoding.

4.2. Receiver Operation

4.2.1. Process recovered symbols

In the FEC module, the block triggering the decoding module generally needs to satisfy the following conditions:

1. The block receives a sufficient number of source symbols;

2. The block receive at least one repair symbol.

However, the specific required amounts of source symbols and repair symbols SHOULD be determined by the FEC schemes module.

5. FEC Frame Types

FEC will introduce two new frames into QUIC protocol: one is the Source Symbol ID frame, which SHOULD be attached to the original packet, and the other is the Repair Frame, which contains repair data, and SHOULD be encapsulated and transmitted independently.

5.1. Source Symbol ID Frame

FEC SID FRAME contains the identification of the current source packet, which declares the encoding flow and the block current symbol is involved.

SID Frame SHOULD be inserted after the fec-protected QUIC frame to indicate the range of FEC protected data. Since the maximum length of the repair symbol is equal to that of the source symbol, it is necessary to restrict the content length of the source packet to prevent the content length of the repair packet from exceeding the MTU limit.

SRC_SYMBOL_ID Frame Structure:

```
SRC_SYMBOL_ID Frame {
  Type (i) = 0xfec5,
  Flow ID (i),
  Explicit Source Payload ID(i),
}
```

Figure 5: SRC_SYMBOL_ID Frame

* Flow ID (i): A variable-length integer that identifies which data stream the symbol belongs to.

* Explicit Source Payload ID (SID): A variable-length integer that serves as a unique identifier for the source symbol. It MUST includes the block id and the symbol index of current source symbol. This can be formatted as follows.

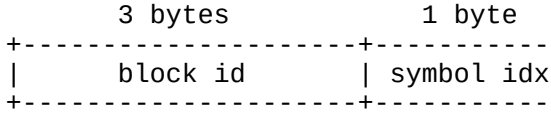

Figure 6: Explicit Source Payload ID

5.2. Repair Symbol Frame

In contrast to the SID Frame mentioned above, the Repair Frame SHOULD include not only the identification but also the encoded content of the source symbols.

Repair symbol frame SHOULD incorporate the following informations:

```
REPAIR_SYMBOL Frame {
  Type (i) = 0xfec6,
  Repair ID {
    Flow ID(i),
    Expilicit Repair payload id (4 bytes),
  }
  Repair Key (4 bytes),
  Repair Symbol Payload (E),
}
```

Figure 7: REPAIR_SYMBOL Frame

* Repair ID(i): - Flow ID(i): A variable-length integer that identifies which data stream the symbol belongs to; - Explicit Repair Payload ID (4 bytes): A variable-length integer that serves as a unique identifier for the repair symbol. It MUST includes the block id and the symbol index of current repair symbol.

* Repair Key (4 bytes): An optional key generated by fec scheme, and is used for FEC decoding.

* Repair Symbol Payload: The content of repair symbol, whose length is typically aligns with the block’s longest symbol. This can be formatted as Figure above.

6. Operation & Management Consideration

The QUIC-FEC protocol are expected to be a universal solution against packet loss.

When implemented, the QUIC packets will be finally divided into different FEC blocks according to the stream size or fixed size. Since that, the FEC Framework only needs to focus on whether the symbols within the block satisfy the encoding/decoding conditions, regardless of the upper layer protocols.

It’s important for FEC Framework to determine how to group QUIC packets into blocks, which needs to consider the optimisation effects at the application layer and the FEC scheme support.

For example, dividing data into fixed-size blocks is a more convenient approach and allows better control over the packet repair rate.

However, this may result in some repair packets arriving too late for the source packets to be protected, just as for source packet 4 belows, repair packet of packet 4 will not be send until stream 2 finish the sending:

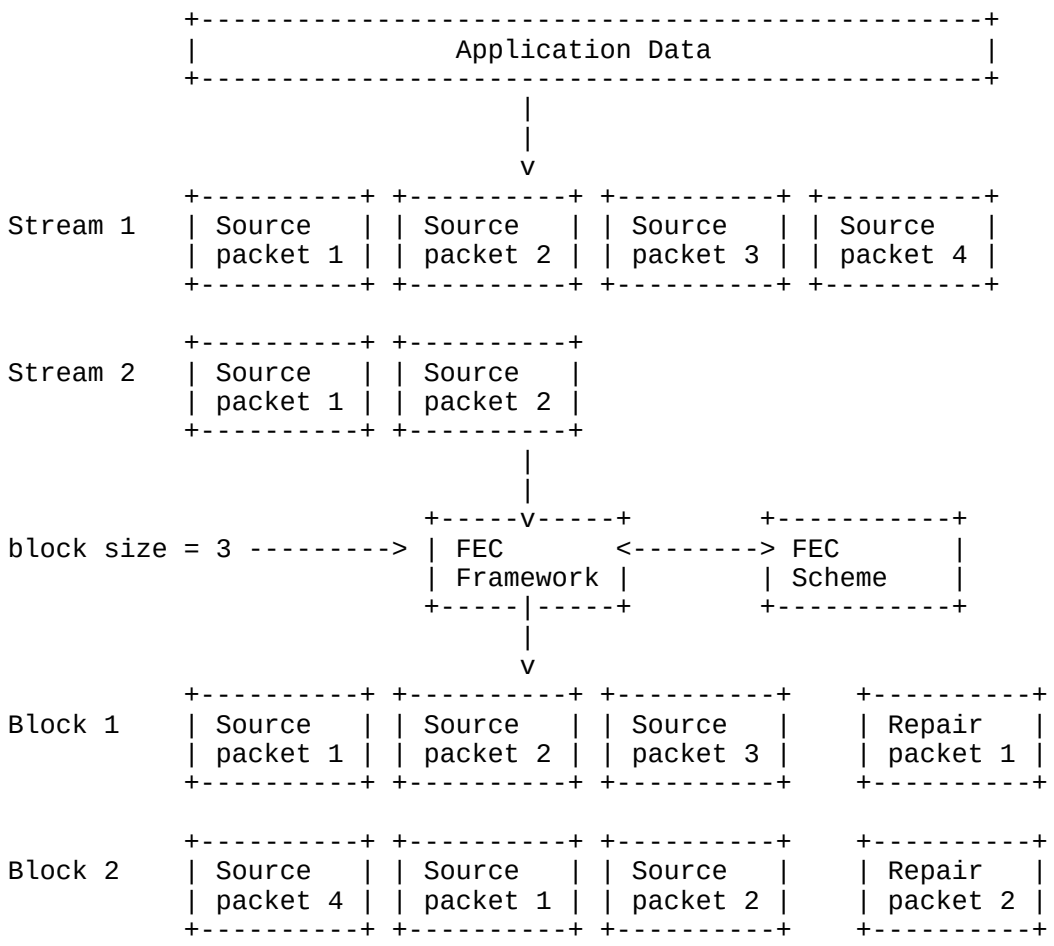

Figure 8: FIXED BLOCK Implementation Steps

Dividing the block according to the QUIC Stream can avoid the problem of long time difference between the generation of repair packets and the arrival of protected packets.

But not all the fec schemes supports variable-length block codecs. It requires the fec scheme to carry the block size information with the FEC frame. For instance, certain algorithms employ a repair key field to specify the number of packets and their indices within a block, enabling the receiver to determine when decoding is possible. Without this information, the receiver cannot support encoding schemes with flexible block sizes.

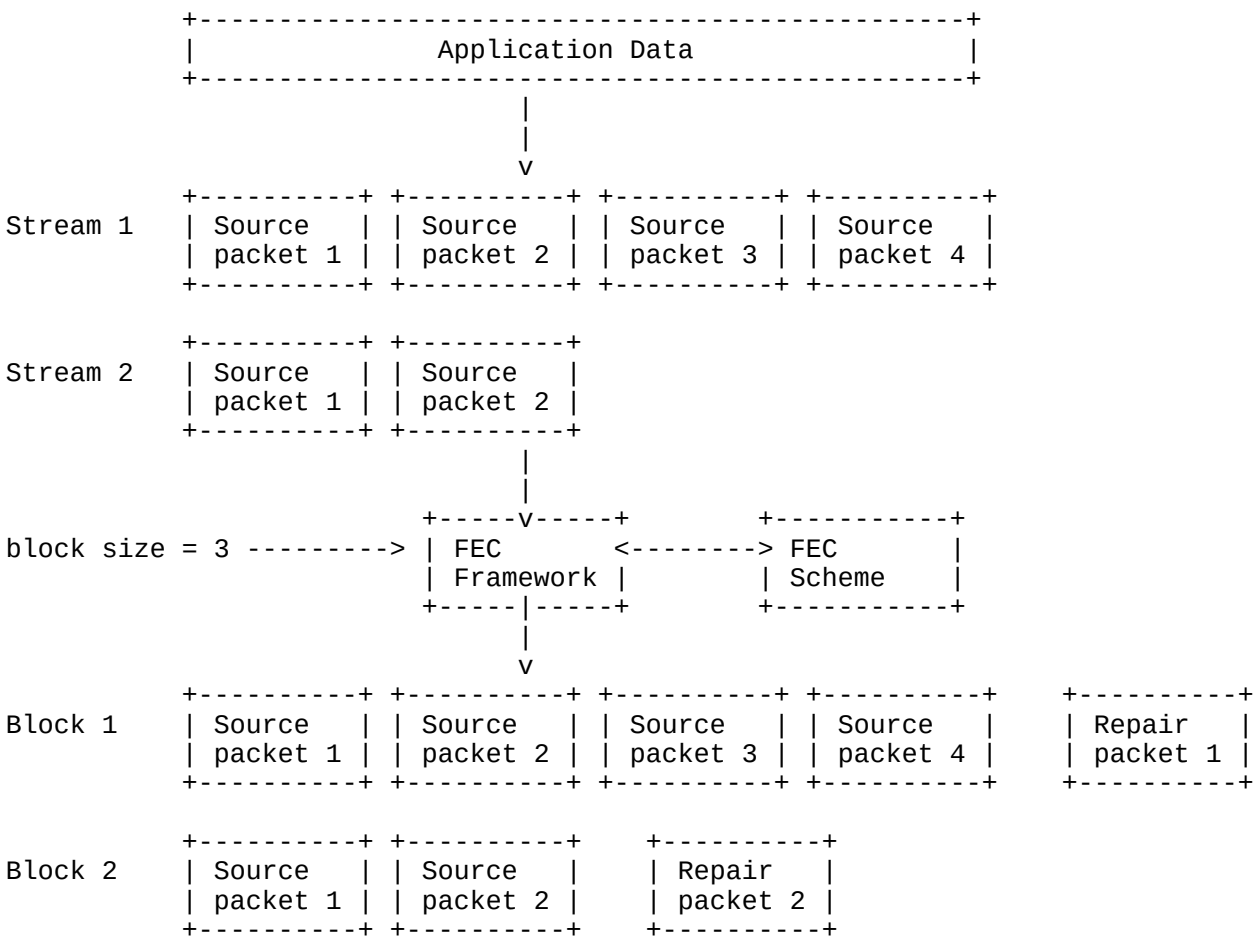

Figure 9: FLEXIBLE BLOCK Implementation Steps

In short, the FEC scheme determines which block partitioning method can be chosen, and the application layer optimization effect determines which partitioning method should be chosen. And the specific codec flow and repair key design should be decided by the fec scheme.

The both steps can be applied to most application layer requests based on the QUIC protocol, and it’s been proven to work with the HTTP/3 and MoQ protocols as the above steps.

However, FEC should not protect all application layer data without discrimination in the QUIC connection. In order to reduce the bandwidth overhead, the FEC module should provide the upper layers with an interface to switch off the FEC module at the Stream level.

It did make differences on when and how to switch off FEC among different application layer. We offers an abstract steps as an example here:

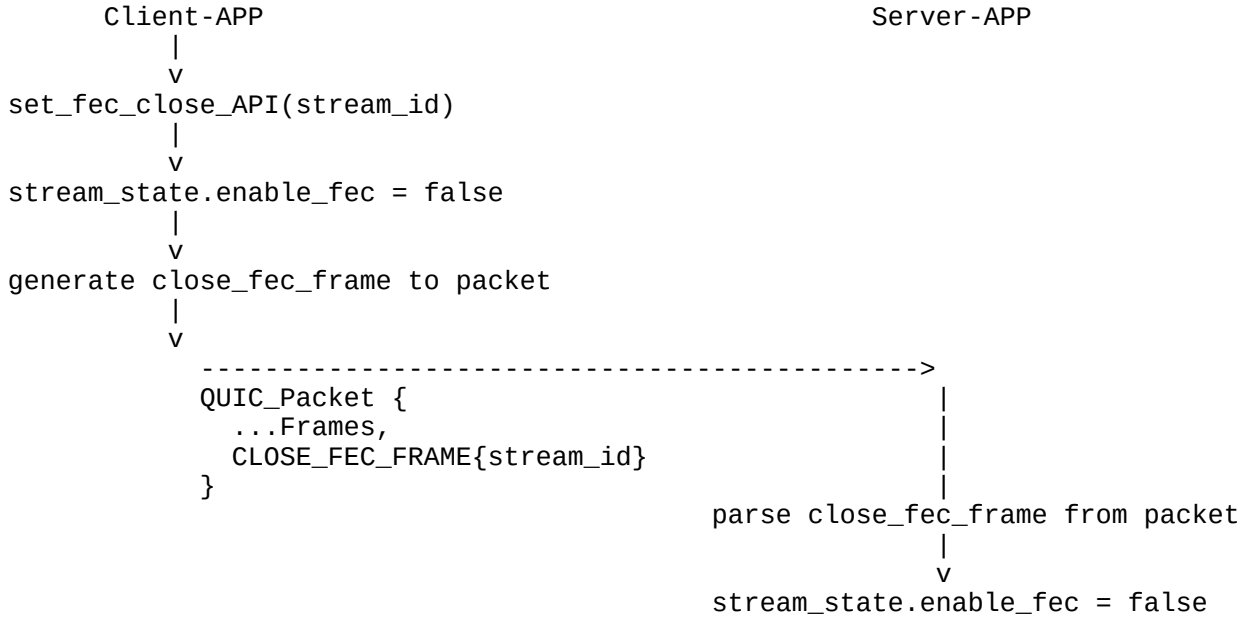

Figure 10: CLOSE FEC Implementation Steps

7. Security Consideration

TBD

8. IANA Considerations

This document defines three new transport parameters for the negotiation of enable fec for QUIC, and two new frame types.

8.1. New transport parameters

The following entry in Table 1 should be added to the "QUIC Transport Parameters" registry under the "QUIC Protocol" heading.

Parameter ID	Parameter name	Specification
0xfece01	fec_encode_schemes	Section 3.1
0xfecd02	fec_decode_schemes	Section 3.1
0xfecb02	fec_max_symbol_num	Section 3.1

Table 1: Addition to QUIC Transport Parameters Entries

8.2. New frames

The following frame types defined in Table 2 should be added to the "QUIC Frame Types" registry under the "QUIC Protocol" heading.

Frame ID	Frame name	Specification
0xfec5	SID Frame	Section 5.1
0xfec6	Repair Frame	Section 5.2

Table 2: Addition to QUIC Frame Types Entries

9. Acknowledgments

The specification has been materially improved through technical discussions with Ian Swett. Thanks to his recommendations with the draft and FEC mechanism.

10. References

10.1. Normative References

[QUIC-TLS] Thomson, M., Ed. and S. Turner, Ed., "Using TLS to Secure QUIC", RFC 9001, DOI 10.17487/RFC9001, May 2021, <https://www.rfc-editor.org/rfc/rfc9001>.

[QUIC-TRANSPORT] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

[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/rfc/rfc2119>.

[RFC2199] Ramos, A., "Request for Comments Summary RFC Numbers 2100-2199", RFC 2199, DOI 10.17487/RFC2199, January 1998, <https://www.rfc-editor.org/rfc/rfc2199>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, DOI 10.17487/RFC5226, May 2008, <https://www.rfc-editor.org/rfc/rfc5226>.

[RFC6363] Watson, M., Begen, A., and V. Roca, "Forward Error Correction (FEC) Framework", RFC 6363, DOI 10.17487/RFC6363, October 2011, <https://www.rfc-editor.org/rfc/rfc6363>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC9174] Sipos, B., Demmer, M., Ott, J., and S. Perreault, "Delay-Tolerant Networking TCP Convergence-Layer Protocol Version 4", RFC 9174, DOI 10.17487/RFC9174, January 2022, <https://www.rfc-editor.org/rfc/rfc9174>.

[RFC9265] Kuhn, N., Lochin, E., Michel, F., and M. Welzl, "Forward Erasure Correction (FEC) Coding and Congestion Control in Transport", RFC 9265, DOI 10.17487/RFC9265, July 2022, <https://www.rfc-editor.org/rfc/rfc9265>.

10.2. Informative References

[QUIC-RECOVERY] Iyengar, J., Ed. and I. Swett, Ed., "QUIC Loss Detection and Congestion Control", RFC 9002, DOI 10.17487/RFC9002, May 2021, <https://www.rfc-editor.org/rfc/rfc9002>.

[RFC5052] Watson, M., Luby, M., and L. Vicisano, "Forward Error Correction (FEC) Building Block", RFC 5052, DOI 10.17487/RFC5052, August 2007, <https://www.rfc-editor.org/rfc/rfc5052>.

[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/rfc/rfc5510>.

Authors' Addresses

Zheng Huanhuan
Alibaba Inc.
Email: zhenghuanhuan.zhh@taobao.com

Yanmei Liu
Alibaba Inc.
Email: miaoji.lym@alibaba-inc.com