Workgroup: QUIC
Internet-Draft:
draft-ietf-quic-version-negotiation-13
Updates: 8999 (if approved)
Published: 6 November 2022
Intended Status: Standards Track
Expires: 10 May 2023
Authors: D. Schinazi E. Rescorla
Google LLC Mozilla

Compatible Version Negotiation for QUIC

Abstract

QUIC does not provide a complete version negotiation mechanism but instead only provides a way for the server to indicate that the version the client chose is unacceptable. This document describes a version negotiation mechanism that allows a client and server to select a mutually supported version. Optionally, if the client's chosen version and the negotiated version share a compatible first flight format, the negotiation can take place without incurring an extra round trip. This document updates RFC 8999.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://quicwg.github.io/version-negotiation/draft-ietf-quic-version-negotiation.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-quic-version-negotiation/.

Discussion of this document takes place on the QUIC Working Group mailing list (mailto:quic@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/quic/. Subscribe at https://www.ietf.org/mailman/listinfo/quic/.

Source for this draft and an issue tracker can be found at https://github.com/quicwg/version-negotiation.

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 10 May 2023.

Copyright Notice

Table of Contents

1. Introduction

The version-invariant properties of QUIC [QUIC-INVARIANTS] define a Version Negotiation packet but do not specify how an endpoint reacts when it receives one. QUIC version 1 [QUIC] allows the server to use a Version Negotiation packet to indicate that the version the client chose is unacceptable, but doesn't allow the client to safely make use of that information to create a new connection with a mutually supported version. This document updates [QUIC-INVARIANTS] by defining version negotiation mechanisms that leverage the Version Negotiation packet.

With proper safety mechanisms in place, the Version Negotiation packet can be part of a mechanism to allow two QUIC implementations to negotiate between two totally disjoint versions of QUIC. This document specifies version negotiation using Version Negotiation packets, which adds an extra round trip to connection establishment if needed.

It is beneficial to avoid additional round trips whenever possible, especially given that most incremental versions are broadly similar to the previous version. This specification also defines a simple version negotiation mechanism which leverages similarities between versions and can negotiate between "compatible" versions without additional round trips.

1.1. Conventions

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.

1.2. Definitions

The document uses the following terms:

* In the context of a given QUIC connection, the "first flight" of packets refers to the set of packets the client creates and sends to initiate the connection before it has heard back from the server.

* In the context of a given QUIC connection, the "client's chosen version" is the QUIC version of the connection's first flight.

*The "original version" is the QUIC version of the very first packet the client sends to the server. If version negotiation spans multiple connections (see Section 2.4), the original version is equal to the client's chosen version of the first QUIC connection.

*The "negotiated version" is the QUIC version in use on the connection once the version negotiation process completes.

*The "Maximum Segment Lifetime" (MSL) represents the time a QUIC packet can exist in the network. Implementations can make this configurable, and a **RECOMMENDED** value is one minute. Note that the term "segment" here originated in Section 3.4.1 of [TCP].

2. Version Negotiation Mechanism

This document specifies two means of performing version negotiation: one "incompatible" which requires a round trip and is applicable to all versions, and one "compatible" that allows saving the round trip but only applies when the versions are compatible (see Section 2.2).

The client initiates a QUIC connection by choosing an original version and sending a first flight of QUIC packets with a long header to the server [QUIC-INVARIANTS]. The client's first flight includes Version Information (see Section 3) which will be used to optionally enable compatible version negotiation (see Section 2.3), and to prevent version downgrade attacks (see Section 4).

Upon receiving this first flight, the server verifies whether it knows how to parse first flights from the original version. If it does not, then it starts incompatible version negotiation, see Section 2.1, which causes the client to initiate a new connection with a different version. For instance, if the client initiates a connection with version A and the server starts incompatible version negotiation and the client then initiates a new connection with version B, we say that the first connection's client chosen version is A, the second connection's client chosen version is B, and the original version for the entire sequence is A.

If the server can parse the first flight, it can either establish the connection using the client's chosen version, or it **MAY** select any other compatible version, as described in Section 2.3.

Note that it is possible for a server to have the ability to parse the first flight of a given version without fully supporting it, in the sense that it implements enough of the version's specification to parse first flight packets but not enough to fully establish a connection using that version.

2.1. Incompatible Version Negotiation

The server starts incompatible version negotiation by sending a Version Negotiation packet. This packet **SHALL** include each entry from the server's set of Offered Versions (see Section 5) in a Supported Version field. The server **MAY** add reserved versions (as defined in Section 6.3 of [QUIC]) in Supported Version fields.

Clients will ignore a Version Negotiation packet if it contains the original version attempted by the client; see Section 4. The client also ignores a Version Negotiation packet that contains incorrect connection ID fields; see Section 6 of [QUIC-INVARIANTS].

Upon receiving the Version Negotiation packet, the client **SHALL** search for a version it supports in the list provided by the server. If it doesn't find one, it **SHALL** abort the connection attempt. Otherwise, it **SHALL** select a mutually supported version and send a new first flight with that version - this version is now the negotiated version.

The new first flight will allow the endpoints to establish a connection using the negotiated version. The handshake of the negotiated version will exchange version information (see Section 3) required to ensure that version negotiation was genuine, i.e. that no attacker injected packets in order to influence the version negotiation process, see Section 4.

Only servers can start incompatible version negotiation: clients **MUST NOT** send Version Negotiation packets and servers **MUST** ignore all received Version Negotiation packets.

2.2. Compatible Versions

If A and B are two distinct versions of QUIC, A is said to be "compatible" with B if it is possible to take a first flight of packets from version A and convert it into a first flight of packets from version B. As an example, if versions A and B are absolutely equal in their wire image and behavior during the handshake but differ after the handshake, then A is compatible with B and B is compatible with A. Note that the conversion of the first flight can be lossy: some data such as QUIC version 1 0-RTT packets could be ignored during conversion and retransmitted later.

Version compatibility is not symmetric: it is possible for version A to be compatible with version B and for B not to be compatible with A. This could happen for example if version B is a strict superset of version A: if version A includes the concept of streams and STREAM frames, and version B includes the concept of streams and the hypothetical concept of tubes along with STREAM and TUBE frames,

then A would be compatible with B but B would not be compatible with A.

Note that version compatibility does not mean that every single possible instance of a first flight will succeed in conversion to the other version. A first flight using version A is said to be "compatible" with version B if two conditions are met: first that version A is compatible with version B, and second that the conversion of this first flight to version B is well-defined. For example, if version B is equal to A in all aspects except it introduced a new frame in its first flight that version A cannot parse or even ignore, then B could still be compatible with A as conversions would succeed for connections where that frame is not used. In this example, first flights using version B that carry this new frame would not be compatible with version A.

When a new version of QUIC is defined, it is assumed to not be compatible with any other version unless otherwise specified. Similarly, no other version is compatible with the new version unless otherwise specified. Implementations **MUST NOT** assume compatibility between versions unless explicitly specified.

Note that both endpoints might disagree on whether two versions are compatible or not. For example, two versions could have been defined concurrently and then specified as compatible in a third document much later - in that scenario one endpoint might be aware of the compatibility document while the other may not.

2.3. Compatible Version Negotiation

When the server can parse the client's first flight using the client's chosen version, it can extract the client's Version Information structure (see Section 3). This contains the list of versions that the client knows its first flight is compatible with.

In order to perform compatible version negotiation, the server **MUST** select one of these versions that (1) it supports and (2) it knows the client's chosen version to be compatible with. This selected version is now the negotiated version. After selecting it, the server attempts to convert the client's first flight into that version, and replies to the client as if it had received the converted first flight.

If those formats are identical, as in cases where the negotiated version is the same as the client's chosen version, then this will be the identity transform. If the first flight is correctly formatted, then this conversion process cannot fail by definition of the first flight being compatible; if the server is unable to convert the first flight, it **MUST** abort the handshake.

If a document specifies that a QUIC version is compatible with another, that document **MUST** specify the mechanism by which clients are made aware of the negotiated version. An example of such a mechanism is to have the client determine the server's negotiated version by examining the QUIC long header Version field. Note that, in this example mechanism, it is possible for the server to initially send packets with the client's chosen version before switching to the negotiated version (this can happen when the client's Version Information structure spans multiple packets; in that case the server might acknowledge the first packet in the client's chosen version and later switch to a different negotiated version). Mutually compatible versions **SHOULD** use the same mechanism.

Note that, after the first flight is converted to the negotiated version, the handshake completes in the negotiated version. If the negotiated version has requirements that apply during the handshake, those requirements apply to the entire handshake, including the converted first flight. In particular, if the negotiated version mandates that endpoints perform validations on handshake packets, endpoints **MUST** also perform such validations on the converted first flight. For instance, if the negotiated version requires that the 5-tuple remain stable for the entire handshake (as QUIC version 1 does), then both endpoints need to validate the 5-tuple of all handshake packets, including the converted first flight.

Note also that the client can disable compatible version negotiation by only including the Chosen Version in the Available Versions field of the Version Information; see Section 3.

If the server does not find a compatible version (including the client's chosen version), it will perform incompatible version negotiation instead, see Section 2.1.

Note that it is possible to have incompatible version negotiation followed by compatible version negotiation. For instance, if version A is compatible with B and C is compatible with D, the following scenario could occur:

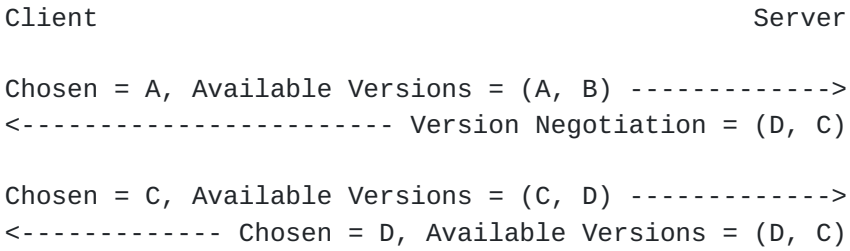

Figure 1: Combined Negotiation Example

In this example, the client selected C from the server's Version Negotiation packet, but the server preferred D and then selected it from the client's offer.

2.4. Connections and Version Negotiation

QUIC connections are shared state between a client and a server [QUIC-INVARIANTS]. The compatible version negotiation mechanism defined in this document (see Section 2.3) is performed as part of a single QUIC connection; that is, the packets with the client's chosen version are part of the same connection as the packets with the negotiated version.

In comparison, the incompatible version negotiation mechanism, which leverages QUIC Version Negotiation packets (see Section 2.1) conceptually operates across two QUIC connections: the connection attempt prior to receiving the Version Negotiation packet is distinct from the connection with the incompatible version that follows.

Note that this separation across two connections is conceptual: it applies to normative requirements on QUIC connections, but does not require implementations to internally use two distinct connection objects.

2.5. Client Choice of Original Version

When the client picks its original version, it will try to avoid incompatible version negotiation to save a round trip. Therefore, the client **SHOULD** pick an original version to maximize the combined probability that both:

* The server knows how to parse first flights from the original version.

* The original version is compatible with the client's preferred version.

Without additional information, this could mean selecting the oldest version that the client supports, while advertising newer compatible versions in the client's first flight.

3. Version Information

During the handshake, endpoints will exchange Version Information, which consists of a chosen version and a list of available versions. Any version of QUIC that supports this mechanism **MUST** provide a mechanism to exchange Version Information in both directions during the handshake, such that this data is authenticated.

In QUIC version 1, the Version Information is transmitted using a new "version_information" transport parameter; see Section 7.4 of [QUIC]. The contents of Version Information are shown below (using the notation from the "Notational Conventions" section of [QUIC]):

```
Version Information {
  Chosen Version (32),
  Available Versions (32) ...,
}
```

Figure 2: Version Information Format

The content of each field is described below:

Chosen Version: The version that the sender has chosen to use for this connection. In most cases, this field will be equal to the value of the Version field in the long header that carries this data; however future versions or extensions can choose to set different values in the long header Version field.

The contents of the Available Versions field depends on whether it is sent by the client or by the server.

Client-Sent Available Versions: When sent by a client, the Available Versions field lists all the versions that this first flight is compatible with, ordered by descending preference. Note that the version in the Chosen Version field **MUST** be included in this list to allow the client to communicate the chosen version's preference. Note that this preference is only advisory, servers **MAY** choose to use their own preference instead.

Server-Sent Available Versions: When sent by a server, the Available Versions field lists all the Fully-Deployed Versions of this server deployment, see Section 5. The ordering of the versions in this field does not carry any semantics. Note that the version in the Chosen Version field is not necessarily included in this list, because the server operator could be in the process of removing support for this version. For the same reason, the Available Versions field **MAY** be empty.

Clients and servers **MAY** both include versions following the pattern 0x?a?a?a?a in their Available Versions list. Those versions are reserved to exercise version negotiation (see the Versions section of [QUIC]), and will never be selected when choosing a version to use.

4. Version Downgrade Prevention

A version downgrade is an attack where a malicious entity manages to make the QUIC endpoints negotiate a QUIC version different from the

one they would have negotiated in the absence of the attack. The mechanism described in this document is designed to prevent downgrade attacks.

Clients **MUST** ignore any received Version Negotiation packets that contain the original version. A client that makes a connection attempt based on information received from a Version Negotiation packet **MUST** ignore any Version Negotiation packets it receives in response to that connection attempt.

Both endpoints **MUST** parse their peer's Version Information during the handshake. If that leads to a parsing failure (for example, if it is too short or if its length is not divisible by four), then the endpoint **MUST** close the connection; if the connection was using QUIC version 1, that connection closure **MUST** use a transport error of type TRANSPORT_PARAMETER_ERROR. If an endpoint receives a Chosen Version equal to zero, or any Available Version equal to zero, it **MUST** treat it as a parsing failure. If a server receives a Version Information where the Chosen Version is not included in Available Versions, it **MUST** treat it as a parsing failure.

Every QUIC version that supports version negotiation **MUST** define a method for closing the connection with a version negotiation error. For QUIC version 1, version negotiation errors are signaled using a transport error of type VERSION_NEGOTIATION_ERROR; see Section 10.2.

When a server receives a client's first flight, the server will first establish which QUIC version is in use for this connection in order to properly parse the first flight. For example, the server determines that QUIC version 1 is in use by observing that the Version field of the first Long Header packet it receives is set to 0x00000001. When the server then processes the client's Version Information, the server **MUST** validate that the client's Chosen Version matches the version in use for the connection. If the two differ, the server **MUST** close the connection with a version negotiation error. For example, if a server receives the client's Version Information over QUIC version 1 (as indicated by the Version field of the Long Header packets that carried the transport parameters) and the client's Chosen Version is not set to 0x00000001, the server will close the connection with a version negotiation error.

If a client receives a Version Information where the server's Chosen Version was not sent by the client as part of its Available Versions, the client **MUST** close the connection with a version negotiation error.

If the Version Information was missing, the endpoints **MAY** complete the handshake. However, if a client has reacted to a Version

Negotiation packet and the Version Information was missing, the client **MUST** close the connection with a version negotiation error.

If the client received and acted on a Version Negotiation packet, the client **MUST** validate the server's Available Versions field. The Available Versions field is validated by confirming that the client would have attempted the same version with knowledge of the versions the server supports. That is, the client would have selected the same version if it received a Version Negotiation packet that listed the versions in the server's Available Versions field, plus the negotiated version. If the client would have selected a different version, the client **MUST** close the connection with a version negotiation error. In particular, if the client reacted to a Version Negotiation packet and the server's Available Versions field is empty, the client **MUST** close the connection with a version negotiation error. These connection closures prevent an attacker from being able to use forged Version Negotiation packets to force a version downgrade.

As an example, let's assume a client supports hypothetical QUIC versions 10, 12, and 14 with a preference for higher versions. The client initiates a connection attempt with version 12. Let's explore two independent example scenarios:

* In the first scenario, the server supports versions 10, 13, and 14 but only 13 and 14 are Fully-Deployed (see Section 5). The server sends a Version Negotiation packet with versions 10, 13, and 14. This triggers an incompatible version negotiation and the client initiates a new connection with version 14. Then the server's Available Versions field contains 13 and 14. In that scenario, the client would have also picked 14 if it had received a Version Negotiation packet with versions 13 and 14, therefore the handshake succeeds using negotiated version 14.

* In the second scenario, the server supports versions 10, 13, and 14 and they are all Fully-Deployed. However, the attacker forges a Version Negotiation packet with versions 10 and 13. This triggers an incompatible version negotiation and the client initiates a new connection with version 10. Then the server's Available Versions field contains 10, 13 and 14. In that scenario, the client would have picked 14 instead of 10 if it had received a Version Negotiation packet with versions 10, 13 and 14, therefore the client aborts the handshake with a version negotiation error.

This validation of Available Versions is not sufficient to prevent downgrade. Downgrade prevention also depends on the client ignoring Version Negotiation packets that contain the original version; see Section 2.1.

After the process of version negotiation in this document completes, the version in use for the connection is the version that the server sent in the Chosen Version field of its Version Information. That remains true even if other versions were used in the Version field of long headers at any point in the lifetime of the connection. In particular, since during compatible version negotiation the client is made aware of the negotiated version by the QUIC long header version (see Section 2.3), clients **MUST** validate that the server's Chosen Version is equal to the negotiated version; if they do not match, the client **MUST** close the connection with a version negotiation error. This prevents an attacker's ability to influence version negotiation by forging the Version long header field.

5. Server Deployments of QUIC

While this document mainly discusses a single QUIC server, it is common for deployments of QUIC servers to include a fleet of multiple server instances. We therefore define the following terms:

Acceptable Versions: This is the set of versions supported by a given server instance. More specifically, these are the versions that a given server instance will use if a client sends a first flight using them.

Offered Versions: This is the set of versions that a given server instance will send in a Version Negotiation packet if it receives a first flight from an unknown version. This set will most often be equal to the Acceptable Versions set, except during short transitions while versions are added or removed (see below).

Fully-Deployed Versions: This is the set of QUIC versions that is supported and negotiated by every single QUIC server instance in this deployment. If a deployment only contains a single server instance, then this set is equal to the Offered Versions set, except during short transitions while versions are added or removed (see below).

If a deployment contains multiple server instances, software updates may not happen at exactly the same time on all server instances. Because of this, a client might receive a Version Negotiation packet from a server instance that has already been updated and the client's resulting connection attempt might reach a different server instance which hasn't been updated yet.

However, even when there is only a single server instance, it is still possible to receive a stale Version Negotiation packet if the server performs its software update while the Version Negotiation packet is in flight.

This could cause the version downgrade prevention mechanism described in Section 4 to falsely detect a downgrade attack. To

avoid that, server operators **SHOULD** perform a three-step process when they wish to add or remove support for a version:

When adding support for a new version:

- The first step is to progressively add support for the new version to all server instances. This step updates the Acceptable Versions but not the Offered Versions nor the Fully-Deployed Versions. Once all server instances have been updated, operators wait for at least one MSL to allow any in-flight Version Negotiation packets to arrive.

- Then, the second step is to progressively add the new version to Offered Versions on all server instances. Once complete, operators wait for at least another MSL.

- Finally, the third step is to progressively add the new version to Fully-Deployed Versions on all server instances.

When removing support for a version:

- The first step is to progressively remove the version from Fully-Deployed Versions on all server instances. Once it has been removed on all server instances, operators wait for at least one MSL to allow any in-flight Version Negotiation packets to arrive.

- Then, the second step is to progressively remove the version from Offered Versions on all server instances. Once complete, operators wait for at least another MSL.

- Finally, the third step is to progressively remove support for the version from all server instances. That step updates the Acceptable Versions.

Note that, during the update window, connections are vulnerable to downgrade attacks for partially-deployed versions. This is because a client cannot distinguish such a downgrade attack from legitimate exchanges with both updated and non-updated server instances.

6. Application Layer Protocol Considerations

When a client creates a QUIC connection, its goal is to use an application layer protocol. Therefore, when considering which versions are compatible, clients will only consider versions that support one of the intended application layer protocols. If the client's first flight advertises multiple Application Layer Protocol Negotiation (ALPN) [ALPN] tokens and multiple compatible versions, it is possible for some application layer protocols to not be able to run over some of the offered compatible versions. It is the

server's responsibility to only select an ALPN token that can run over the compatible QUIC version that it selects.

A given ALPN token **MUST NOT** be used with a new QUIC version different from the version for which the ALPN token was originally defined, unless all the following requirements are met:

- The new QUIC version supports the transport features required by the application protocol.
- The new QUIC version supports ALPN.
- The version of QUIC for which the ALPN token was originally defined is compatible with the new QUIC version.

When incompatible version negotiation is in use, the second connection which is created in response to the received version negotiation packet **MUST** restart its application layer protocol negotiation process without taking into account the original version.

7. Considerations for Future Versions

In order to facilitate the deployment of future versions of QUIC, designers of future versions **SHOULD** attempt to design their new version such that commonly deployed versions are compatible with it.

QUIC version 1 defines multiple features which are not documented in the QUIC invariants. Since, at the time of writing, QUIC version 1 is widely deployed, this section discusses considerations for future versions to help with compatibility with QUIC version 1.

7.1. Interaction with Retry

QUIC version 1 features Retry packets, which the server can send to validate the client's IP address before parsing the client's first flight. A server that sends a Retry packet can do so before parsing the client's first flight. A server that sends a Retry packet therefore might not have processed the client's Version Information before doing so.

If a future document wishes to define compatibility between two versions that support retry, that document **MUST** specify how version negotiation (both compatible and incompatible) interacts with retry during a handshake that requires both. For example, that could be accomplished by having the server first send a Retry packet in the original version thereby validating the client's IP address before attempting compatible version negotiation. If both versions support authenticating Retry packets, the compatibility definition needs to define how to authenticate the Retry in the negotiated version

handshake even though the Retry itself was sent using the client's chosen version.

7.2. Interaction with TLS resumption

QUIC version 1 uses TLS 1.3, which supports session resumption by sending session tickets in one connection that can be used in a later connection; see Section 2.2 of [TLS]. New versions that also use TLS 1.3 **SHOULD** mandate that their session tickets are tightly scoped to one version of QUIC; i.e., require that clients not use them across multiple version and that servers validate this client requirement. This helps mitigate cross-protocol attacks.

7.3. Interaction with 0-RTT

QUIC version 1 allows sending data from the client to the server during the handshake, by using 0-RTT packets. If a future document wishes to define compatibility between two versions that support 0-RTT, that document **MUST** address the scenario where there are 0-RTT packets in the client's first flight. For example, this could be accomplished by defining which transformations are applied to 0-RTT packets. That document could specify that compatible version negotiation causes 0-RTT data to be rejected by the server.

8. Special Handling for QUIC Version 1

Because QUIC version 1 was the only IETF Standards Track version of QUIC published before this document, it is handled specially as follows: if a client is starting a QUIC version 1 connection in response to a received Version Negotiation packet, and the version_information transport parameter is missing from the server's transport parameters, then the client **SHALL** proceed as if the server's transport parameters contained a version_information transport parameter with a Chosen Version set to 0x00000001 and an Available Version list containing exactly one version set to 0x00000001. This allows version negotiation to work with servers that only support QUIC version 1. Note that implementations which wish to use version negotiation to negotiate versions other than QUIC version 1 will need to implement the version negotiation mechanism defined in this document.

9. Security Considerations

The security of this version negotiation mechanism relies on the authenticity of the Version Information exchanged during the handshake. In QUIC version 1, transport parameters are authenticated ensuring the security of this mechanism. Negotiation between compatible versions will have the security of the weakest common version.

The requirement that versions not be assumed compatible mitigates the possibility of cross-protocol attacks, but more analysis is still needed here. That analysis is out of scope for this document.

10. IANA Considerations

10.1. QUIC Transport Parameter

This document registers a new value in the "QUIC Transport Parameters" registry maintained at <https://www.iana.org/assignments/quic>.

Value: 0xFF73DB
Parameter Name: version_information
Status: provisional
Specification: This document

When this document is approved, it will request permanent allocation of a codepoint in the 0-63 range to replace the provisional codepoint described above.

10.2. QUIC Transport Error Code

This document registers a new value in the "QUIC Transport Error Codes" registry maintained at <https://www.iana.org/assignments/quic>.

Value: 0x53F8
Code: VERSION_NEGOTIATION_ERROR
Description: Error negotiating version
Status: provisional
Specification: This document

When this document is approved, it will request permanent allocation of a codepoint in the 0-63 range to replace the provisional codepoint described above.

11. References

11.1. Normative References

[ALPN] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <https://www.rfc-editor.org/rfc/rfc7301>.

[QUIC] 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>.

[QUIC-INVARIANTS] Thomson, M., "Version-Independent Properties of QUIC", RFC 8999, DOI 10.17487/RFC8999, May 2021, <https://www.rfc-editor.org/rfc/rfc8999>.

[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>.

[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>.

[TLS] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/rfc/rfc8446>.

11.2. Informative References

[TCP] Eddy, W., Ed., "Transmission Control Protocol (TCP)", STD 7, RFC 9293, DOI 10.17487/RFC9293, August 2022, <https://www.rfc-editor.org/rfc/rfc9293>.

Acknowledgments

The authors would like to thank Nick Banks, Mike Bishop, Martin Duke, Ryan Hamilton, Roberto Peon, Anthony Rossi, and Martin Thomson for their input and contributions.

Authors' Addresses

David Schinazi
Google LLC
1600 Amphitheatre Parkway
Mountain View, CA 94043
United States of America

Email: dschinazi.ietf@gmail.com

Eric Rescorla
Mozilla

Email: ekr@rtfm.com