Workgroup: QUIC Working Group
Internet-Draft:
draft-ietf-quic-version-negotiation-03
Published: 5 February 2021
Intended Status: Informational
Expires: 9 August 2021
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 offered is unacceptable. This document describes a version negotiation mechanism that allows a client and server to select a mutually supported version. Optionally, if the original and negotiated version share a compatible first flight format, the negotiation can take place without incurring an extra round trip.

Discussion of this work is encouraged to happen on the QUIC IETF mailing list quic@ietf.org or on the GitHub repository which contains the draft: https://github.com/quicwg/version-negotiation/.

Discussion Venues

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

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The version-invariant properties of QUIC [INV] define a version negotiation (VN) packet but do not specify how an endpoint reacts when it receives one. QUIC version 1 [QUIC] allows the server to use a VN packet to indicate that the version the client offered 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. With proper safety mechanisms in place, the VN packet can be part of a mechanism to allow two QUIC implementations to negotiate between two totally disjoint versions of QUIC, at the cost of an extra round trip. However, it is beneficial to avoid that cost whenever possible, especially given that most incremental versions are broadly similar to the the previous version.

This specification describes a simple version negotiation mechanism which optionally leverages similarities between versions and can negotiate between the set of "compatible" versions in a single round trip.

Discussion of this work is encouraged to happen on the QUIC IETF mailing list quic@ietf.org or on the GitHub repository which contains the draft: https://github.com/quicwg/version-negotiation/.

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.

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.

Version compatibility is not bijective: 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.

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.

3. 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.

The client initiates a QUIC connection by sending a first flight of QUIC packets with a long header to the server [INV]. We'll refer to the version of those packets as the "original version". The client's first flight includes handshake version information (see Section 4) which will be used to optionally enable compatible version negotation (see Section 3.3), and to prevent version downgrade attacks (see Section 5).

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 3.2. If the server can parse the first flight, it can either establish the connection using the original version, or it MAY attempt compatible version negotiation, see Section 3.3.

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.

3.1. Connections and Version Negotiation

QUIC connections are shared state between a client and a server [INV]. The compatible version negotiation mechanism defined in this document (see Section 3.3) operates inside of a QUIC connection; i.e., the packets with the original version are part of the same connection as the packets with the negotiated version. On the other hand, the incompatible version negotiation mechanism, which leverages QUIC Version Negotiation packets (see Section 3.2) conceptually operates across two QUIC connections: one before the Version Negotiation packet, and a distinct connection after.

3.2. Incompatible Version Negotiation

The server starts incompatible version negotiation by sending a VN packet, listing all the versions that it does support.

Upon receiving the VN packet, the client will search for a version it supports in the list provided by the server. If it doesn't find one, it aborts the connection attempt. Otherwise, it selects a mutually supported version and sends a new first flight with that version - we refer to this version as 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 handshake version information (see Section 4) required to ensure that VN was genuine, i.e. that no attacker injected packets in order to influence the VN process, see Section 5.

3.3. Compatible Version Negotiation

When the server can parse the client's first flight using the original version, it can extract the client's handshake version information (see Section 4). This contains the list of versions that the client thinks its first flight is compatible with.

If the server supports one of the client's compatible versions, and the server also believes that the original version is compatible with this version, then the server converts the client's first flight to that version and replies to the client as if it had received the converted first flight. The version used by the server in its reply is refered to as the "negotiated version". The server MUST NOT reply with a version that is not present in the client's compatible versions, unless it is the original version.

If the server does not find a compatible version, it will use the original version if it supports it, and if it doesn't then the server will perform incompatible version negotiation instead, see Section 3.2.

For the duration of the compatible version negotiation process, clients MUST use the same 5-tuple (source and destination IP addresses and UDP port numbers). During that time, clients MUST also use the same Destination Connection ID, except if the server explicitly instructs the client to use a different Destination Connection ID (for example, a QUIC version 1 server can accomplish this by sending an INITIAL packet with a Source Connection ID that differed from the client's Destination Connection ID). This allows load balancers to ensure that packets for a given connection are routed to the same server.

4. Handshake Version Information

During the handshake, endpoints will exchange handshake version information, which is a blob of data that is defined below. In QUIC version 1, the handshake version information is transmitted using a new transport parameter, version_negotiation. The contents of handshake version information depend on whether the client or server is sending it, and are shown below (using the notation from the "Notational Conventions" section of [QUIC]):

```
Client Handshake Version Information {
  Currently Attempted Version (32),
  Previously Attempted Version (32),
  Received Negotiation Version Count (i),
  Received Negotiation Version (32) ...,
  Compatible Version Count (i),
  Compatible Version (32) ...,
}
```

Figure 1: Client Handshake Version Information

The content of each field is described below:

Currently Attempted Version: The version that the client is attempting to use. This field MUST be equal to the value of the Version field in the long header that carries this data.

Previously Attempted Version: If the client is sending this first flight in response to a Version Negotiation packet, this field contains the version that the client used in the previous first flight that triggered the version negotiation packet. If the client did not receive a Version Negotiation packet, this field SHALL be all-zeroes.

Received Negotiation Version Count: A variable-length integer specifying the number of Received Negotiation Version fields following it. If the client is sending this first flight in response to a Version Negotiation packet, the subsequent versions SHALL include all the versions from that Version Negotiation packet in order, even if they are not supported by the client (even if the versions are reserved). If the client has not received a Version Negotiation packet on this connection, this field SHALL be 0.

Compatible Version Count: A variable-length integer specifying the number of Compatible Version fields following it. The client lists all versions that this first flight is compatible with in the subsequent Compatible Version fields, ordered by descending preference. Note that the version in the Currently Attempted Version field MUST be included in the Compatible Version list to allow the client to communicate the currently attempted version's preference. Note that this preference is only advisory, servers MAY choose to use their own preference instead.

```
Server Handshake Version Information {
  Negotiated Version (32),
  Supported Version Count (i),
  Supported Version (32) ...,
}
```

Figure 2: Server Handshake Version Information

The content of each field is described below:

Negotiated Version: The version that the server chose to use for the connection. This field MUST be equal to the value of the Version field in the long header that carries this data.

Supported Version Count: A variable-length integer specifying the number of Supported Version fields following it. The server encodes all versions it supports in the subsequent list, ordered by descending preference. Note that the version in the Negotiated Version field MUST be included in the Supported Version list to allow the server to communicate the negotiated version's preference. Note that this preference is only advisory, clients MAY choose to use their own preference instead.

Clients MAY include versions following the pattern 0x?a?a?a?a in their Compatible Version list, and the server in their Supported Version list. Those versions are reserved to exercise version negotiation (see the Versions section of [QUIC]), and MUST be ignored when parsing these fields. On the other hand, the Received Negotiation Version list MUST be identical to the received Version Negotiation packet, so clients MUST NOT add or remove reserved version from that list.

5. Version Downgrade Prevention

Clients MUST ignore any received Version Negotiation packets that contain the version that they initially attempted. Once a client has reacted to a Version Negotiation packet, it MUST drop all subsequent Version Negotiation packets on that connection.

Servers MUST validate that the client's Currently Attempted Version matches the version in the long header that carried the handshake version information. Similarly, clients MUST validate that the server's Negotiated Version matches the long header version. If an endpoint's validation fails, it MUST close the connection. If the connection was using QUIC version 1, it MUST be closed with a transport error of type VERSION_NEGOTIATION_ERROR.

When a server parses the client's handshake version information, if the Received Negotiation Version Count is not zero, the server MUST validate that it could have sent the Version Negotation packet described by the client in response to a first flight of version Previously Attempted Version. In particular, the server MUST ensure that there are no versions that it supports that are absent from the Received Negotiation Versions, and that the ordering matches the server's preference. If this validation fails, the server MUST close the connection. If the connection was using QUIC version 1, it MUST be closed with a transport error of type VERSION_NEGOTIATION_ERROR. This mitigates an attacker's ability to forge Version Negotiation packets to force a version downgrade.

If a server operator is progressively deploying a new QUIC version throughout its fleet, it MAY perform a two-step process where it first progressively adds support for the new version, but without enforcing its presence in Received Negotiation Versions. Once all servers have been upgraded, the second step is to start enforcing that the new version is present in Received Negotiation Versions. This opens connections to version downgrades during the upgrade window, since those could be due to clients communicating with both upgraded and non-upgraded servers.

If an endpoint receives its peer's Handshake Version Information and fails to parse it (for example, if it is too short), then the endpoint MUST close the connection. If the connection was using QUIC version 1, it MUST be closed with a transport error of type TRANSPORT_PARAMETER_ERROR.

6. Supported Versions

The server's Supported Version list allows it to communicate the full list of versions it supports to the client. In the case where clients initially attempt connections with the oldest version they support, this allows them to be notified of more recent versions the server supports. If the client notices that the server supports a version that is more preferable that the one initially attempted by default, the client SHOULD cache that information and attempt the preferred version in subsequent connections.

7. Client Choice of Original Version

The client's first flight SHOULD be sent using the version that the server is most likely to support (in the absence of other information, this will often be the oldest version the client supports).

8. 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. This impacts compatible version negotiation because a server who wishes to send a retry packet before parsing the client's first flight won't have parsed the client's handshake version information yet. 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 send a retry packet first and validating the client's IP address before starting version negotiation and deciding whether to use compatible version negotiation on that connection (in that scenario the retry packet would be sent using the original version).

9. 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. Alternatively, that document could specify that compatible version negotiation causes 0-RTT data to be rejected by the server.

10. 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. For example, a successor to QUIC version 1 may wish to design its transport parameters in a way that does not preclude compatibility. Additionally, frames in QUIC version 1 do not use a self-describing encoding, so unrecognized frame types cannot be parsed or ignored (see the Extension Frames section of [QUIC]); this means that new versions that wish to be very similar to QUIC version 1 and compatible with it should avoid introducing new frames in initial packets.

11. Security Considerations

The security of this version negotiation mechanism relies on the authenticity of the handshake 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.

The presence of the Attempted Version and Negotiated Version fields mitigates an attacker's ability to forge packets by altering the version.

12. IANA Considerations

12.1. QUIC Transport Parameter

If this document is approved, IANA shall assign the following entry in the QUIC Transport Parameter Registry:

Value	Parameter Name	Reference
0x73DB	version_negotiation	This document

12.2. QUIC Transport Error Code

If this document is approved, IANA shall assign the following entry in the QUIC Transport Error Codes Registry:

Value	Parameter Name	Reference
0x53F8	VERSION_NEGOTIATION_ERROR	This document

13. Normative References

[INV] Thomson, M., "Version-Independent Properties of QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-invariants-13, 14 January 2021, <http://www.ietf.org/internet-drafts/draft-ietf-quic-invariants-13.txt>.

[QUIC] Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", Work in Progress, Internet-Draft, draft-ietf-quic-transport-34, 14 January 2021, <http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-34.txt>.

[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>.

[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/info/rfc8174>.

Acknowledgments

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

Authors' Addresses

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

Email: dschinazi.ietf@gmail.com

Eric Rescorla
Mozilla

Email: ekr@rtfm.com