Workgroup: QUIC
Internet-Draft:
draft-ietf-quic-version-negotiation-04
Published: 27 May 2021
Intended Status: Standards Track
Expires: 28 November 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 Venues

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

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

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.

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.

In this document, 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.

2. 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 Acceptaple 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 6 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 this opens connections to version downgrades (but only for partially-deployed versions) during the update window, since those could be due to clients communicating with both updated and non-updated server instances.

3. 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 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.

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.

4. 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 Version Information (see Section 5) which will be used to optionally enable compatible version negotation (see Section 4.3), and to prevent version downgrade attacks (see Section 6).

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 4.2. 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 4.3.

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.

4.1. 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 4.3) is performed as part of a single QUIC connection; that is, the packets with the original 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 4.2) 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.

4.2. 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 2) in a Supported Version field. The server MAY add reserved versions (as defined in the Versions section of [QUIC]) in Supported Version fields.

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 version information (see Section 5) required to ensure that VN was genuine, i.e. that no attacker injected packets in order to influence the VN process, see Section 6.

4.3. Compatible Version Negotiation

When the server can parse the client's first flight using the original version, it can extract the client's Version Information structure (see Section 5). 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 4.2.

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.

5. Version Information

During the handshake, endpoints will exchange Version Information, which is a blob of data that is defined below. In QUIC version 1, the Version Information is transmitted using a new transport parameter, version_information. The contents of Version Information are shown below (using the notation from the "Notational Conventions" section of [QUIC]):

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

Figure 1: 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.

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

Client-Sent Other Versions: When sent by a client, the Other 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 Other Versions: When sent by a server, the Other Versions field lists all the Fully-Deployed Versions of this server deployment, see Section 2.

Clients and servers MAY both include versions following the pattern 0x?a?a?a?a in their Other 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.

6. 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.

Both endpoints MUST parse their peer's Version Information during the handshake. If parsing the Version Information failed (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 the Version Information was missing, the endpoints MAY complete the handshake if they have reason to believe the peer might not

support this extension. However, if a client has reacted to a Version Negotiation packet and the Version Information was missing, the client MUST close the connection; if the connection was using QUIC version 1, that connection closure MUST use a transport error of type VERSION_NEGOTIATION_ERROR.

If a client has reacted to a Version Negotiation packet, it MUST validate that the server's Other Versions field does not contain the client's original version, and that the client would have selected the same negotiated version if it had received a Version Negotiation packet whose Supported Versions field had the same contents as the server's Other Versions field. If any of these checks fail, the client MUST close the connection; if the connection was using QUIC version 1, that connection closure MUST use a transport error of type VERSION_NEGOTIATION_ERROR. This connection closure prevents an attacker from being able to use forged Version Negotiation packets to force a version downgrade.

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; endpoints MUST NOT change the version that they consider to be in use based on the Version field of long headers as that field could be forged by attackers.

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 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 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.

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
0xFF73DB	version_information	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, <https://tools.ietf.org/html/draft-ietf-quic-invariants-13>.

[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, <https://tools.ietf.org/html/draft-ietf-quic-transport-34>.

[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>.

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