Workgroup: QUIC Working Group
Internet-Draft:
draft-ietf-quic-version-negotiation-00
Published: 26 February 2020
Intended Status: Informational
Expires: 29 August 2020
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 Initial 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/.

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 29 August 2020.

Copyright Notice

Table of Contents

1. Introduction

QUIC [QUIC] does not provide a complete version negotiation (VN) mechanism; the VN packet only allows the server 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/.

2. 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.

3. Version Negotiation Mechanism

The mechanism defined in this document is straightforward: the client maintains a list of QUIC versions it supports, ordered by preference. Its Initial packet is 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); that Initial packet then lists all compatible versions (Section 7) that the client supports in the Compatible Version fields of its transport parameters (Figure 1). Note that the client's compatible version list always contains its currently attempted version.

- If the server supports one of the client's compatible versions, it selects a version it supports from the client's compatible version list. It then responds with that version in all of its future packets (except for Retry, as below).

- If the server does not support any of the client's compatible versions, it sends a Version Negotiation packet listing all the versions it supports.

If the server leverages compatible versions and responds with a different version from the client's currently attempted version, it MUST NOT select a version not offered by the client. The client MUST validate that the version in the server's packets is one of the compatible versions that it offered and that it matches the negotiated version in the server's transport parameters.

If the server sends a Retry, it MUST use the same version that the client provided in its Initial. Version negotiation takes place after the retry cycle is over.

In order for negotiation to complete successfully, the client's Initial packet (and initial CRYPTO frames) MUST be interpretable by the server. This implies that servers must retain the ability to process the Initial packet from older versions as long as they are

reasonably popular. This is not generally an issue in practice as long as the the overall structure of the protocol remains similar.

4. Version Negotiation Transport Parameter

This document registers a new transport parameter, version_negotiation. The contents of this transport parameter depend on whether the client or server is sending it, and are shown below:

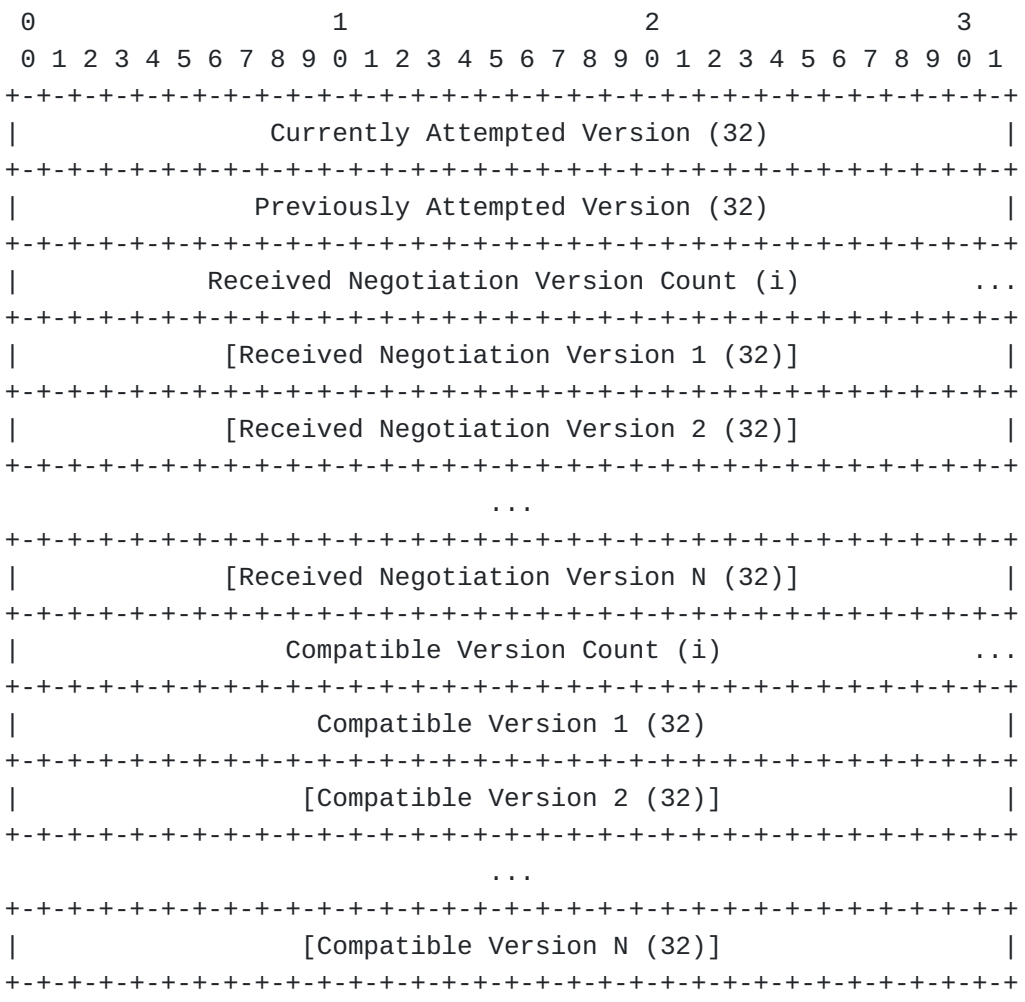

Figure 1: Client Transport Parameter Format

The content of each field is described below:

Currently Attempted Version: The version that the client is using in this Initial. This field MUST be equal to the value of the Version field in the long header that carries this transport parameter.

Previously Attempted Version: If the client is sending this Initial in response to a Version Negotiation packet, this field contains the version that the client used in the previous Initial packet

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 Initial 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 compatible with Currently Attempted Version 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.

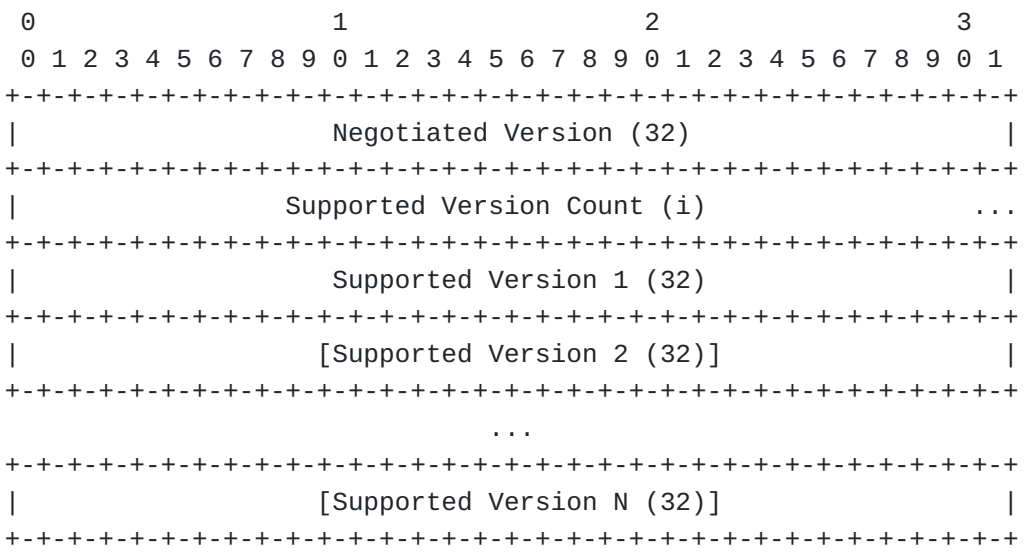

Figure 2: Server Transport Parameter Format

The content of each field is described below:

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

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.

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.

Servers MUST validate that the client's Currently Attempted Version matches the version in the long header that carried the transport parameter. 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 with an error of type VERSION_NEGOTIATION_ERROR.

When a server parses the client's version_negotiation transport parameter, 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 an Initial 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 with an 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.

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. Compatible Versions

Two versions of QUIC A and B are said to be "compatible" if a version A Initial can be used to negotiate version B and vice versa. The most common scenario is a sequence of versions 1, 2, 3, etc. in which all the Initial packets have the same basic structure but might include specific extensions (especially inside the crypto handshake) that are only meaningful in some subset of versions and are ignored in others. Note that it is not possible to add new frame types in Initial packets because QUIC frames do not use a self-describing encoding, so unrecognized frame types cannot be parsed or ignored (see the Extension Frames section of [QUIC]).

When a new version of QUIC is defined, it is assumed to not be compatible with any other version unless otherwise specified. Implementations MUST NOT assume compatibility between version unless explicitly specified.

8. Security Considerations

The crypto handshake is already required to guarantee agreement on the supported parameters, so 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.

9. IANA Considerations

If this document is approved, IANA shall assign the identifier 0x73DB for the version_negotiation transport parameter from the QUIC Transport Parameter Registry and the identifier 0x53F8 for VERSION_NEGOTIATION_ERROR from the QUIC Transport Error Codes registry.

10. Normative References

[QUIC] Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", Work in Progress, Internet-Draft, draft-ietf-quic-transport-27, 21 February 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-27.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>.

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