QUIC Working Group
Internet-Draft
Intended status: Informational
Expires: September 12, 2019

D. Schinazi
Google LLC
E. Rescorla
Mozilla

March 11, 2019

Compatible Version Negotiation for QUIC
draft-schinazi-quic-version-negotiation-00

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 from a set of QUIC versions which share a compatible Initial format without incurring an extra round trip.

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 September 12, 2019.

Table of Contents

1. Introduction

QUIC [I-D.ietf-quic-transport] 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. In principle the VN packet could 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, experience with negotiation of previous IETF protocols indicates that this is probably not the most common scenario:

1. Implementations do not generally want to incur an extra round trip to negotiate versions.

2. Most incremental versions are broadly similar to the the previous version, and so the version negotiation mechanism can be built on the assumption that the version advertisement and selection is common to the versions to be negotiated.

This specification describes a simple version negotiation mechanism which exploits property (2) and can negotiate between the set of "compatible" versions in a single round trip. Negotiation between totally disjoint versions - if it ever proves to be necessary - is left as a topic for future work.

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 practice, this will generally be the oldest version the client supports); that Initial packet then lists all of the other compatible versions (Section 5) that the client supports in the supported_compatible_versions field of its transport parameters (Section 4). The server then selects its preferred version and responds with that version in all of its future packets (except for Retry, as below). It also inserts the selected version in the negotiated_compatible_version field of its transport parameters.

The server 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 versions that it offered and that it matches the value 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.

If the server receives an Initial packet with a version it does not understand this will cause a connection failure and the server SHOULD send a Version Negotiation packet as defined in [I-D.ietf-quic-transport].

4. Compatible Versions Transport Parameter

This document adds a new transport parameter, CompatibleVersions:

```
struct {
      select (Handshake.msg_type) {
         case client_hello:
            QuicVersion supported_compatible_versions<4..2^8-4>;

         case encrypted_extensions:
            QuicVersion negotiated_compatible_version;
      }
} CompatibleVersions;
```

The client's "supported_compatible_versions" parameter lists the versions it supports in decreasing order of preference. The server's "negotiated_compatible_version" parameter lists the version it has selected. If the client does not send this transport parameter, the server MUST assume that the client only supports the version it used for the Initial packet and MUST NOT send its own parameter.

Clients MAY include versions following the pattern 0x?a?a?a?a in their supported_compatible_versions. Those versions are reserved to exercise version negotiation (see the Versions section of [I-D.ietf-quic-transport]), and MUST be ignored by the server when parsing supported_compatible_versions.

5. Compatible Versions

Two versions of QUIC A and B are "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 [I-D.ietf-quic-transport]).

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.

6. 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.

7. IANA Considerations

If this document is approved, IANA shall assign the identifier TBD for the "compatible_versions" transport parameter.

8. Normative References

[I-D.ietf-quic-transport]
Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-18 (work in progress), January 2019.

[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