Workgroup: QUIC
Internet-Draft: draft-ietf-quic-v2-04
Published: 1 June 2022
Intended Status: Standards Track
Expires: 3 December 2022
Authors: M. Duke
Google LLC

QUIC Version 2

Abstract

This document specifies QUIC version 2, which is identical to QUIC version 1 except for some trivial details. Its purpose is to combat various ossification vectors and exercise the version negotiation framework. It also serves as a template for the minimum changes in any future version of QUIC.

Note that "version 2" is an informal name for this proposal that indicates it is the second standards-track QUIC version. The protocol specified here uses a version number other than 2 in the wire image.

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.org/quic-v2/draft-ietf-quic-v2.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-quic-v2/.

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/.

Source for this draft and an issue tracker can be found at https://github.com/quicwg/quic-v2.

Status of This Memo

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 3 December 2022.

Copyright Notice

Table of Contents

1. Introduction

QUIC [QUIC] has numerous extension points, including the version number that occupies the second through fifth bytes of every long header (see [QUIC-INVARIANTS]). If experimental versions are rare, and QUIC version 1 constitutes the vast majority of QUIC traffic, there is the potential for middleboxes to ossify on the version bytes always being 0x00000001.

In QUIC version 1, Initial packets are encrypted with the version-specific salt as described in Section 5.2 of [QUIC-TLS]. Protecting Initial packets in this way allows observers to inspect their contents, which includes the TLS Client Hello or Server Hello messages. Again, there is the potential for middleboxes to ossify on the version 1 key derivation and packet formats.

Finally [QUIC-VN] provides two mechanisms for endpoints to negotiate the QUIC version to use. The "incompatible" version negotiation method can support switching from any QUIC version to any other version with full generality, at the cost of an additional round-trip at the start of the connection. "Compatible" version negotiation eliminates the round-trip penalty but levies some restrictions on how much the two versions can differ semantically.

QUIC version 2 is meant to mitigate ossification concerns and exercise the version negotiation mechanisms. The only change is a tweak to key derivation inputs to enforce full key separation. Any endpoint that supports two versions needs to implement version negotiation to protect against downgrade attacks.

2. 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.

3. Differences with QUIC Version 1

QUIC version 2 endpoints **MUST** implement the QUIC version 1 specification as described in [QUIC], [QUIC-TLS], and [QUIC-LOSSRECOVERY]. However, the following differences apply in version 2.

3.1. Version Field

The Version field of long headers is 0x709a50c4.

RFC Editor's Note: Please remove the sentence below prior to publication of a final version of this document.

This version number will not change in subsequent versions of this draft, unless there is a behavior change that breaks compatibility.

3.2. Long Header Packet Types

All version 2 long header packet types are different. The Type field values are:

* Initial: 0b01

* 0-RTT: 0b10

* Handshake: 0b11

* Retry: 0b00

3.3. Cryptography changes

3.3.1. Initial Salt

The salt used to derive Initial keys in Section 5.2 of [QUIC-TLS] changes to:

```
initial_salt = 0xa707c203a59b47184a1d62ca570406ea7ae3e5d3
```

3.3.2. HKDF Labels

The labels used in [QUIC-TLS] to derive packet protection keys (Section 5.1), header protection keys (Section 5.4), Retry Integrity Tag keys (Section 5.8), and key updates (Section 6.1) change from "quic key" to "quicv2 key", from "quic iv" to "quicv2 iv", from "quic hp" to "quicv2 hp", and from "quic ku" to "quicv2 ku", to meet the guidance for new versions in Section 9.6 of that document.

3.3.3. Retry Integrity Tag

The key and nonce used for the Retry Integrity Tag (Section 5.8 of [QUIC-TLS]) change to:

```
secret =
  0x3425c20cf88779df2ff71e8abfa78249891e763bbed2f13c048343d348c060e2
key = 0xba858dc7b43de5dbf87617ff4ab253db
nonce = 0x141b99c239b03e785d6a2e9f
```

4. Version Negotiation Considerations

QUIC version 2 is not intended to deprecate version 1. Endpoints that support version 2 might continue support for version 1 to maximize compatibility with other endpoints. In particular, HTTP clients often use Alt-Svc [RFC7838] to discover QUIC support. As this mechanism does not currently distinguish between QUIC versions, HTTP servers that support multiple versions reduce the probability of incompatibility and the cost associated with QUIC version negotiation or TCP fallback. For example, an origin advertising support for "h3" in Alt-Svc **SHOULD** support QUIC version 1 as it was the original QUIC version used by HTTP/3 and therefore some clients will only support that version.

Any QUIC endpoint that supports QUIC version 2 **MUST** send, process, and validate the version_information transport parameter specified in [QUIC-VN] to prevent version downgrade attacks.

Note that version 2 meets that document's definition of a compatible version with version 1, and version 1 is compatible with version 2. Therefore, servers can use compatible negotiation to switch a connection between the two versions. Endpoints that support both versions **SHOULD** support compatible version negotiation to avoid a round trip.

4.1. Compatible Negotiation Requirements

Compatible version negotiation between versions 1 and 2 follow the same requirements in either direction. This section uses the terms "original version" and "negotiated version" from [QUIC-VN].

If the server sends a Retry packet, it **MUST** use the original version. The client ignores Retry packets using other versions. The client **MUST NOT** use a different version in the subsequent Initial that contains the Retry token. The server **MAY** encode the QUIC version in its Retry token to validate that the client did not switch versions, and drop the packet if it switched.

QUIC version 2 uses the same transport parameters to authenticate the Retry as QUIC version 1. After switching to a negotiated version after a Retry, the server **MUST** include the relevant transport parameters to validate that the server sent the Retry and the connection IDs used in the exchange, as described in Section 7.3 of [QUIC].

The server **SHOULD** start sending its Initial packets using the negotiated version as soon as it decides to change. Before the server is able to process transport parameters from the client, it might need to respond to Initial packets from the client. For these packets the server uses the original version.

Once the client has processed a packet using the negotiated version, it **SHOULD** send subsequent Initial packets using that version. The server **MUST NOT** discard its original version Initial receive keys until it successfully processes a packet with the negotiated version.

Both endpoints **MUST** send Handshake or 1-RTT packets using the negotiated version. An endpoint **MUST** drop packets using any other version. Endpoints have no need to generate the keying material that would allow them to decrypt or authenticate these packets.

The client **MUST NOT** send 0-RTT packets using the negotiated version, even after processing a packet of that version from the server. Servers can apply original version 0-RTT packets to a connection without additional considerations.

5. TLS Resumption and NEW_TOKEN Tokens

TLS session tickets and NEW_TOKEN tokens are specific to the QUIC version of the connection that provided them. Clients **SHOULD NOT** use a session ticket or token from a QUIC version 1 connection to initiate a QUIC version 2 connection, or vice versa.

Servers **MUST** validate the originating version of any session ticket or token and not accept one issued from a different version. A rejected ticket results in falling back to a full TLS handshake, without 0-RTT. A rejected token results in the client address remaining unverified, which limits the amount of data the server can send.

After compatible version negotiation, any resulting session ticket maps to the negotiated version rather than original one.

6. Ossification Considerations

QUIC version 2 provides protection against some forms of ossification. Devices that assume that all long headers will encode version 1, or that the version 1 Initial key derivation formula will remain version-invariant, will not correctly process version 2 packets.

However, many middleboxes such as firewalls focus on the first packet in a connection, which will often remain in the version 1 format due to the considerations above.

Clients interested in combating firewall ossification can initiate a connection using version 2 if they are either reasonably certain the server supports it, or are willing to suffer a round-trip penalty if they are incorrect. In particular, a server that issues a session ticket for version 2 indicates an intent to maintain version 2 support while the ticket remains valid, even if support cannot be guaranteed.

7. Applicability

This version of QUIC provides no change from QUIC version 1 relating to the capabilities available to applications. Therefore, all Application Layer Protocol Negotiation (ALPN) ([RFC7301]) codepoints specified to operate over QUIC version 1 can also operate over this version of QUIC. In particular, both the "h3" [I-D.ietf-quic-http] and "doq" [RFC9250] ALPNs can operate over QUIC version 2.

Unless otherwise stated, all QUIC extensions defined to work with version 1 also work with version 2.

8. Security Considerations

QUIC version 2 introduces no changes to the security or privacy properties of QUIC version 1.

The mandatory version negotiation mechanism guards against downgrade attacks, but downgrades have no security implications, as the version properties are identical.

9. IANA Considerations

This document requests that IANA add the following entry to the QUIC version registry:

Value: 0x709a50c4

Status: provisional

Specification: This Document

Change Controller: IETF

Contact: QUIC WG

10. References

10.1. Normative References

[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-LOSSRECOVERY] Iyengar, J., Ed. and I. Swett, Ed., "QUIC Loss Detection and Congestion Control", RFC 9002, DOI 10.17487/RFC9002, May 2021, <https://www.rfc-editor.org/rfc/rfc9002>.

[QUIC-TLS] Thomson, M., Ed. and S. Turner, Ed., "Using TLS to Secure QUIC", RFC 9001, DOI 10.17487/RFC9001, May 2021, <https://www.rfc-editor.org/rfc/rfc9001>.

[QUIC-VN] Schinazi, D. and E. Rescorla, "Compatible Version Negotiation for QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-version-negotiation-07, 5 April 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-version-negotiation-07>.

[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>.

10.2. Informative References

[I-D.ietf-quic-http] Bishop, M., "Hypertext Transfer Protocol Version 3 (HTTP/3)", Work in Progress, Internet-Draft, draft-ietf-quic-http-34, 2 February 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-34>.

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

[RFC7301] 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>.

[RFC7838]
Nottingham, M., McManus, P., and J. Reschke, "HTTP Alternative Services", RFC 7838, DOI 10.17487/RFC7838, April 2016, <https://www.rfc-editor.org/rfc/rfc7838>.

[RFC9250] Huitema, C., Dickinson, S., and A. Mankin, "DNS over Dedicated QUIC Connections", RFC 9250, DOI 10.17487/RFC9250, May 2022, <https://www.rfc-editor.org/rfc/rfc9250>.

Appendix A. Sample Packet Protection

This section shows examples of packet protection so that implementations can be verified incrementally. Samples of Initial packets from both client and server plus a Retry packet are defined. These packets use an 8-byte client-chosen Destination Connection ID of 0x8394c8f03e515708. Some intermediate values are included. All values are shown in hexadecimal.

A.1. Keys

The labels generated during the execution of the HKDF-Expand-Label function (that is, HkdfLabel.label) and part of the value given to the HKDF-Expand function in order to produce its output are:

client in: 00200f746c73313320636c69656e7420696e00

server in: 00200f746c7331332073657276657220696e00

quicv2 key: 001010746c733133207175696337632206b657900

quicv2 iv: 000c0f746c7331332071756963763220697600

quicv2 hp: 00100f746c7331332071756963763220687000

The initial secret is common:

```
initial_secret = HKDF-Extract(initial_salt, cid)
    = ddfcb7b82a430b7845210ad64b406977
      ed51b269a14bc69aa9ea9b366fa3b06b
```

The secrets for protecting client packets are:

```
client_initial_secret
    = HKDF-Expand-Label(initial_secret, "client in", "", 32)
    = 9fe72e1452e91f551b770005054034e4
      7575d4a0fb4c27b7c6cb303a338423ae

key = HKDF-Expand-Label(client_initial_secret, "quicv2 key", "", 16)
    = 95df2be2e8d549c82e996fc9339f4563

iv  = HKDF-Expand-Label(client_initial_secret, "quicv2 iv", "", 12)
    = ea5e3c95f933db14b7020ad8

hp  = HKDF-Expand-Label(client_initial_secret, "quicv2 hp", "", 16)
    = 091efb735702447d07908f6501845794
```

The secrets for protecting server packets are:

```
server_initial_secret
    = HKDF-Expand-Label(initial_secret, "server in", "", 32)
    = 3c9bf6a9c1c8c71819876967bd8b979e
      fd98ec665edf27f22c06e9845ba0ae2f

key = HKDF-Expand-Label(server_initial_secret, "quicv2 key", "", 16)
    = 15d5b4d9a2b8916aa39b1bfe574d2aad

iv  = HKDF-Expand-Label(server_initial_secret, "quicv2 iv", "", 12)
    = a85e7ac31cd275cbb095c626

hp  = HKDF-Expand-Label(server_initial_secret, "quicv2 hp", "", 16)
    = b13861cfadbb9d11ff942dd80c8fc33b
```

A.2. Client Initial

The client sends an Initial packet. The unprotected payload of this packet contains the following CRYPTO frame, plus enough PADDING frames to make a 1162-byte payload:

```
060040f1010000ed0303ebf8fa56f129 39b9584a3896472ec40bb863cfd3e868
04fe3a47f06a2b69484c000004130113 02010000c000000010000e00000b6578
616d706c652e636f6dff01000100000a 00080006001d00170018001000070005
04616c706e0005000501000000000033 00260024001d00209370b2c9caa47fba
baf4559fedba753de171fa71f50f1ce1 5d43e994ec74d748002b000302030400
0d0010000e0403050306030203080408 050806002d00020101001c0002400100
3900320408ffffffffffffffff050480 00ffff07048000ffff08011001048000
75300901100f088394c8f03e51570806 048000ffff
```

The unprotected header indicates a length of 1182 bytes: the 4-byte packet number, 1162 bytes of frames, and the 16-byte authentication tag. The header includes the connection ID and a packet number of 2:

```
d3709a50c4088394c8f03e5157080000449e00000002
```

Protecting the payload produces output that is sampled for header protection. Because the header uses a 4-byte packet number encoding, the first 16 bytes of the protected payload is sampled and then applied to the header as follows:

```
sample = 23b8e610589c83c92d0e97eb7a6e5003

mask = AES-ECB(hp, sample)[0..4]
     = 8e4391d84a

header[0] ^= mask[0] & 0x0f
     = dd
header[18..21] ^= mask[1..4]
     = 4391d848
header = dd709a50c4088394c8f03e5157080000449e4391d848
```

The resulting protected packet is:

```
dd709a50c4088394c8f03e5157080000 449e4391d84823b8e610589c83c92d0e
97eb7a6e5003f57764c5c7f0095ba54b 90818f1bfeecc1c97c54fc731edbd2a2
44e3b1e639a9bc75ed545b98649343b2 53615ec6b3e4df0fd2e7fe9d691a09e6
a144b436d8a2c088a404262340dfd995 ec3865694e3026ecd8c6d2561a5a3667
2a1005018168c0f081c10e2bf14d550c 977e28bb9a759c57d0f7ffb1cdfb40bd
774dec589657542047dffefa56fc8089 a4d1ef379c81ba3df71a05ddc7928340
775910feb3ce4cbcfd8d253edd05f161 458f9dc44bea017c3117cca7065a315d
eda9464e672ec80c3f79ac993437b441 ef74227ecc4dc9d597f66ab0ab8d214b
55840c70349d7616cbe38e5e1d052d07 f1fedb3dd3c4d8ce295724945e67ed2e
efcd9fb52472387f318e3d9d233be7df c79d6bf6080dcbbb41feb180d7858849
7c3e439d38c334748d2b56fd19ab364d 057a9bd5a699ae145d7fdbc8f5777518
1b0a97c3bdedc91a555d6c9b8634e106 d8c9ca45a9d5450a7679edc545da9102
5bc93a7cf9a023a066ffadb9717ffaf3 414c3b646b5738b3cc4116502d18d79d
8227436306d9b2b3afc6c785ce3c817f eb703a42b9c83b59f0dcef1245d0b3e4
0299821ec19549ce489714fe2611e72c d882f4f70dce7d3671296fc045af5c9f
630d7b49a3eb821bbca60f1984dce664 91713bfe06001a56f51bb3abe92f7960
547c4d0a70f4a962b3f05dc25a34bbe8 30a7ea4736d3b0161723500d82beda9b
e3327af2aa413821ff678b2a876ec4b0 0bb605ffcc3917ffdc279f187daa2fce
8cde121980bba8ec8f44ca562b0f1319 14c901cfbd847408b778e6738c7bb5b1
b3f97d01b0a24dcca40e3bed29411b1b a8f60843c4a241021b23132b9500509b
9a3516d4a9dd41d3bacbcd426b451393 521828afedcf20fa46ac24f44a8e2973
30b16705d5d5f798eff9e9134a065979 87a1db4617caa2d93837730829d4d89e
16413be4d8a8a38a7e6226623b64a820 178ec3a66954e10710e043ae73dd3fb2
715a0525a46343fb7590e5eac7ee55fc 810e0d8b4b8f7be82cd5a214575a1b99
629d47a9b281b61348c8627cab38e2a6 4db6626e97bb8f77bdcb0fee476aedd7
ba8f5441acaab00f4432edab3791047d 9091b2a753f035648431f6d12f7d6a68
1e64c861f4ac911a0f7d6ec0491a78c9 f192f96b3a5e7560a3f056bc1ca85983
67ad6acb6f2e034c7f37beeb9ed470c4 304af0107f0eb919be36a86f68f37fa6
1dae7aff14decd67ec3157a11488a14f ed0142828348f5f608b0fe03e1f3c0af
3acca0ce36852ed42e220ae9abf8f890 6f00f1b86bff8504c8f16c784fd52d25
e013ff4fda903e9e1eb453c1464b1196 6db9b28e8f26a3fc419e6a60a48d4c72
14ee9c6c6a12b68a32cac8f61580c64f 29cb6922408783c6d12e725b014fe485
cd17e484c5952bf99bc94941d4b1919d 04317b8aa1bd3754ecbaa10ec227de85
40695bf2fb8ee56f6dc526ef366625b9 1aa4970b6ffa5c8284b9b5ab852b905f
9d83f5669c0535bc377bcc05ad5e48e2 81ec0e1917ca3c6a471f8da0894bc82a
c2a8965405d6eef3b5e293a88fda203f 09bdc72757b107ab14880eaa3ef7045b
580f4821ce6dd325b5a90655d8c5b55f 76fb846279a9b518c5e9b9a21165c509
3ed49baaacadf1f21873266c767f6769
```

A.3. Server Initial

The server sends the following payload in response, including an ACK frame, a CRYPTO frame, and no PADDING frames:

```
02000000000600405a020000560303ee fce7f7b37ba1d1632e96677825ddf739
88cfc79825df566dc5430b9a045a1200 130100002e00330024001d00209d3c94
0d89690b84d08a60993c144eca684d10 81287c834d5311bcf32bb9da1a002b00
020304
```

The header from the server includes a new connection ID and a 2-byte packet number encoding for a packet number of 1:

```
d1709a50c40008f067a5502a4262b50040750001
```

As a result, after protection, the header protection sample is taken starting from the third protected byte:

```
sample = ebb7972fdce59d50e7e49ff2a7e8de76
mask   = 41103f438e
header = d0709a50c40008f067a5502a4262b5004075103e
```

The final protected packet is then:

```
d0709a50c40008f067a5502a4262b500 4075103e63b4ebb7972fdce59d50e7e4
9ff2a7e8de76b0cd8c10100a1f13d549 dd6fe801588fb14d279bef8d7c53ef62
66a9a7a1a5f2fa026c236a5bf8df5aa0 f9d74773aeccfffe910b0f76814b5e33
f7b7f8ec278d23fd8c7a9e66856b8bbe 72558135bca27c54d63fcc902253461c
fc089d4e6b9b19
```

A.4. Retry

This shows a Retry packet that might be sent in response to the Initial packet in Appendix A.2. The integrity check includes the client-chosen connection ID value of 0x8394c8f03e515708, but that value is not included in the final Retry packet:

```
cf709a50c40008f067a5502a4262b574 6f6b656e1dc71130cd1ed39d6efcee5c
85806501
```

A.5. ChaCha20-Poly1305 Short Header Packet

This example shows some of the steps required to protect a packet with a short header. This example uses AEAD_CHACHA20_POLY1305.

In this example, TLS produces an application write secret from which a server uses HKDF-Expand-Label to produce four values: a key, an IV, a header protection key, and the secret that will be used after keys are updated (this last value is not used further in this example).

```
secret
    = 9ac312a7f877468ebe69422748ad00a1
      5443f18203a07d6060f688f30f21632b

key = HKDF-Expand-Label(secret, "quicv2 key", "", 32)
    = 3bfcddd72bcf02541d7fa0dd1f5f9eee
      a817e09a6963a0e6c7df0f9a1bab90f2

iv  = HKDF-Expand-Label(secret, "quicv2 iv", "", 12)
    = a6b5bc6ab7dafce30ffff5dd

hp  = HKDF-Expand-Label(secret, "quicv2 hp", "", 32)
    = d659760d2ba434a226fd37b35c69e2da
      8211d10c4f12538787d65645d5d1b8e2

ku  = HKDF-Expand-Label(secret, "quicv2 ku", "", 32)
    = c69374c49e3d2a9466fa689e49d476db
      5d0dfbc87d32ceeaa6343fd0ae4c7d88
```

The following shows the steps involved in protecting a minimal packet with an empty Destination Connection ID. This packet contains a single PING frame (that is, a payload of just 0x01) and has a packet number of 654360564. In this example, using a packet number of length 3 (that is, 49140 is encoded) avoids having to pad the payload of the packet; PADDING frames would be needed if the packet number is encoded on fewer bytes.

```
pn                 = 654360564 (decimal)
nonce              = a6b5bc6ab7dafce328ff4a29
unprotected header = 4200bff4
payload plaintext  = 01
payload ciphertext = 0ae7b6b932bc27d786f4bc2bb20f2162ba
```

The resulting ciphertext is the minimum size possible. One byte is skipped to produce the sample for header protection.

```
sample = e7b6b932bc27d786f4bc2bb20f2162ba
mask   = 97580e32bf
header = 5558b1c6
```

The protected packet is the smallest possible packet size of 21 bytes.

```
packet = 5558b1c60ae7b6b932bc27d786f4bc2bb20f2162ba
```

Appendix B. Acknowledgments

The author would like to thank Lucas Pardue, David Schinazi, and Martin Thomson for their helpful suggestions.

Appendix C. Changelog

RFC Editor's Note: Please remove this section prior to publication of a final version of this document.

C.1. since draft-ietf-quic-v2-03

* a v2 session ticket is an indication of v2 support
* a v1-only extension is theoretically possible
* editorial fixes

C.2. since draft-ietf-quic-v2-02

* Editorial comments from Lucas Pardue

C.3. since draft-ietf-quic-v2-01

* Ban use of NEW_TOKEN tokens across versions
* version-info transport parameter required for all v2 endpoints
* Explicitly list known ALPN compatibility

C.4. since draft-ietf-quic-v2-00

* Expanded requirements for compatible version negotiation
* Added test vectors
* Greased the packet type codepoints
* Random version number
* Clarified requirement to use QUIC-VN
* Banned use of resumption tokens across versions

C.5. since draft-duke-quic-v2-02

* Converted to adopted draft
* Deleted references to QUIC improvements
* Clarified status of QUIC extensions

C.6. since draft-duke-quic-v2-01

* Made the final version number TBD.

*Added ALPN considerations

C.7. since draft-duke-quic-v2-00

*Added provisional versions for interop

*Change the v1 Retry Tag secret

*Change labels to create full key separation

Author's Address

Martin Duke
Google LLC

Email: martin.h.duke@gmail.com