Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 22, 2016

M. Thomson
Mozilla
R. Hamilton
Google

March 21, 2016

Porting QUIC to Transport Layer Security (TLS)
draft-thomson-quic-tls-00

Abstract

The QUIC experiment defines a custom security protocol. This was necessary to gain handshake latency improvements. This document describes how that security protocol might be replaced with TLS.

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 http://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 22, 2016.

Copyright Notice

Table of Contents

1. Introduction 2
1.1. Notational Conventions 3
2. Protocol Overview 3
2.1. Handshake Overview 4
3. QUIC over TLS Structure 5
4. Mapping of QUIC to QUIC over TLS 6
4.1. Protocol and Version Negotiation 7
4.2. Source Address Validation 8
5. Record Protection 8
5.1. TLS Handshake Encryption 9
5.2. Key Update 9
5.3. Sequence Number Reconstruction 10
5.4. Alternative Design: Exporters 10
6. Pre-handshake QUIC Messages 11
6.1. QUIC Extension 11
6.2. Unprotected Frames Prior to Handshake Completion 15
6.2.1. STREAM Frames 15
6.2.2. ACK Frames 15
6.2.3. WINDOW_UPDATE Frames 15
6.2.4. FEC Packets 16
6.3. Protected Frames Prior to Handshake Completion 16
7. Connection ID 17
8. Security Considerations 18
9. IANA Considerations 18
10. References 18
10.1. Normative References 18
10.2. Informative References 18
Appendix A. Acknowledgments 19
Authors' Addresses 19

1. Introduction

QUIC [I-D.tsvwg-quic-protocol] provides a multiplexed transport for HTTP [RFC7230] semantics that provides several key advantages over HTTP/1.1 [RFC7230] or HTTP/2 [RFC7540] over TCP [RFC0793].

The custom security protocol designed for QUIC provides critical latency improvements for connection establishment. Absent packet loss, most new connections can be established with a single round trip; on subsequent connections between the same client and server, the client can often send application data immediately, that is, zero

round trip setup. TLS 1.3 uses a similar design and aims to provide the same set of improvements.

This document describes how the standardized TLS 1.3 might serve as a security layer for QUIC. The same design could work for TLS 1.2,

though few of the benefits QUIC provides would be realized due to the handshake latency in versions of TLS prior to 1.3.

Alternative Designs: There are other designs that are possible; and many of these alternative designs are likely to be equally good. The point of this document is to articulate a coherent single design. Notes like this throughout the document are used describe points where alternatives were considered.

Note: This is a rough draft. Many details have not been ironed out. Ryan is not responsible for any errors or omissions.

1.1. Notational Conventions

The words "MUST", "MUST NOT", "SHOULD", and "MAY" are used in this document. It's not shouting; when they are capitalized, they have the special meaning defined in [RFC2119].

2. Protocol Overview

QUIC [I-D.tsvwg-quic-protocol] can be separated into several modules:

1. The basic frame envelope describes the common packet layout. This layer includes connection identification, version negotiation, and includes the indicators that allow the framing, public reset, and FEC modules to be identified.

2. The public reset is an unprotected frame that allows an intermediary (an entity that is not part of the security context) to request the termination of a QUIC connection.

3. The forward error correction (FEC) module provides redundant entropy that allows for frames to be repaired in event of loss.

4. Framing comprises most of the QUIC protocol. Framing provides a number of different types of frame, each with a specific purpose.

Framing supports frames for both congestion management and stream multiplexing. Framing additionally provides a liveness testing capability (the PING frame).

5. Crypto provides confidentiality and integrity protection for frames. All frames are protected after the handshake completes on stream 1. Prior to this, data is protected with the 0-RTT keys.

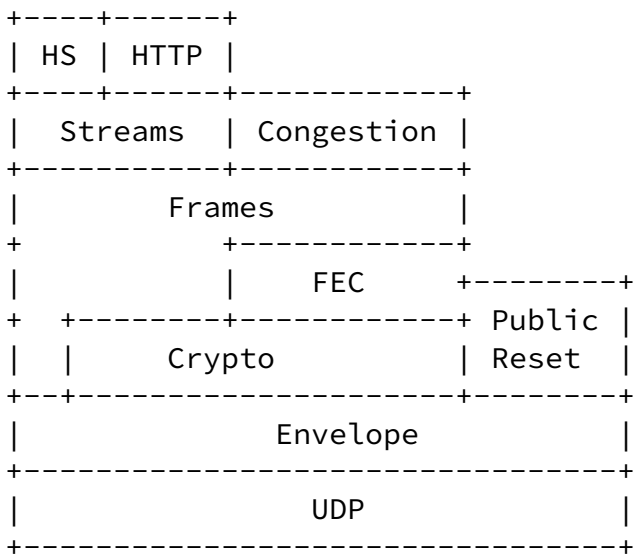
6. Multiplexed streams are the primary payload of QUIC. These provide reliable, in-order delivery of data and are used to carry the encryption handshake and transport parameters (stream 1),

HTTP header fields (stream 3), and HTTP requests and responses. Frames for managing multiplexing include those for creating and destroying streams as well as flow control and priority frames.

7. Congestion management includes packet acknowledgment and other signal required to ensure effective use of available link capacity.

8. HTTP mapping provides an adaptation to HTTP that is based on HTTP/2.

The relative relationship of these components are pictorally represented in Figure 1.



*HS = Crypto Handshake

Figure 1: QUIC Structure

This document describes a replacement of the cryptographic parts of QUIC. This includes the handshake messages that are exchanged on stream 1, plus the record protection that is used to encrypt and authenticate all other frames.

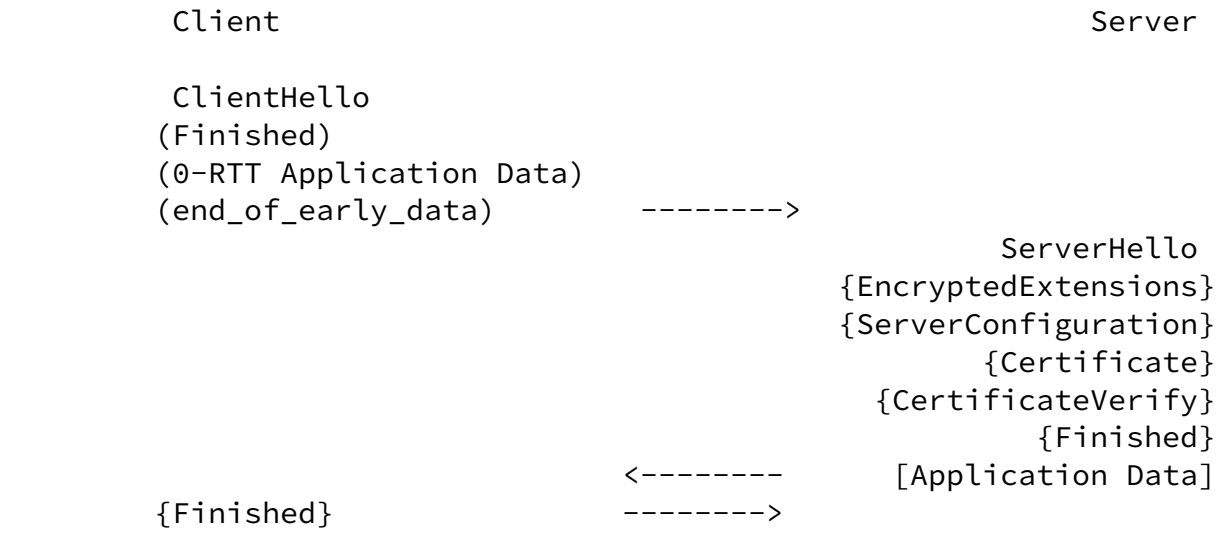
2.1. Handshake Overview

TLS 1.3 provides two basic handshake modes of interest to QUIC:

- A full handshake in which the client is able to send application data after one round trip and the server immediately after receiving the first message from the client.

- A 0-RTT handshake in which the client uses information about the server to send immediately. This data can be replayed by an attacker so it MUST NOT carry a self-contained trigger for any non-idempotent action.

A simplified TLS 1.3 handshake with 0-RTT application data is shown in Figure 2, see [I-D.ietf-tls-tls13] for more options.



```
[Application Data]        <------->      [Application Data]
```

Figure 2: TLS Handshake with 0-RTT

Two additional variations on this basic handshake exchange are relevant to this document:

- The server can respond to a ClientHello with a HelloRetryRequest, which adds an additional round trip prior to the basic exchange. This is needed if the server wishes to request a different key exchange key from the client. HelloRetryRequest might also be used to verify that the client is correctly able to receive packets on the address it claims to have (see Section 4.2).

- A pre-shared key mode can be used for subsequent handshakes to avoid public key operations. This might be the basis for 0-RTT, even if the remainder of the connection is protected by a new Diffie-Hellman exchange.

3. QUIC over TLS Structure

QUIC completes its cryptographic handshake on stream 1, which means that the negotiation of keying material happens within the QUIC protocol. QUIC over TLS does the same, relying on the ordered

delivery guarantees provided by QUIC to ensure that the TLS handshake packets are delivered reliably and in order.

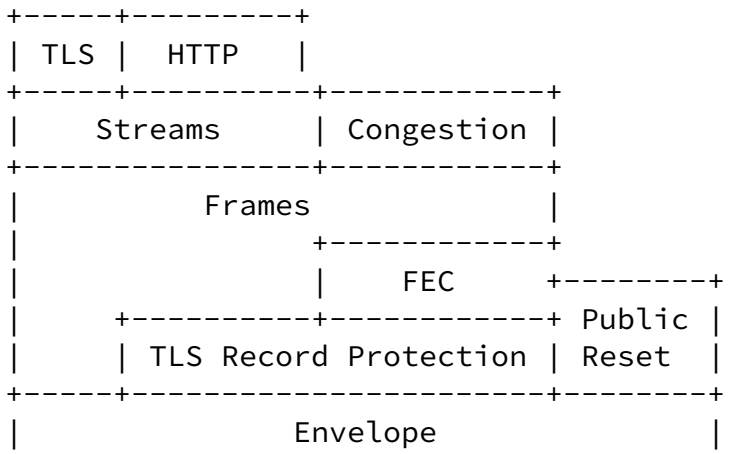


Figure 3: QUIC over TLS

In this design the QUIC envelope carries QUIC frames until the TLS handshake completes. After the handshake successfully completes the key exchange, QUIC frames are then protected by TLS record protection.

QUIC stream 1 is used to exchange TLS handshake packets. QUIC provides for reliable and in-order delivery of the TLS handshake messages.

Prior to the completion of the TLS handshake, QUIC frames can be exchanged. However, these frames are not authenticated or confidentiality protected. Section 6 covers some of the implications of this design.

Alternative Design: TLS could be used to protect the entire QUIC envelope. QUIC version negotiation could be subsumed by TLS and ALPN [RFC7301]. The only unprotected packets are then public resets and ACK frames, both of which could be given first octet values that would easily distinguish them from other TLS packets. This requires that the QUIC sequence numbers be moved to the outside of the record.

4. Mapping of QUIC to QUIC over TLS

Several changes to the structure of QUIC are necessary to make a layered design practical.

These changes produce the handshake shown in Figure 4. In this handshake, QUIC STREAM frames on stream 1 carry the TLS handshake. QUIC is responsible for ensuring that the handshake packets are re-sent in case of loss and that they can be ordered correctly.

QUIC operates without any record protection until the handshake completes, just as TLS over TCP does not include record protection for the handshake messages. Once complete, QUIC frames and forward

error control (FEC) messages are encapsulated in using TLS record protection.

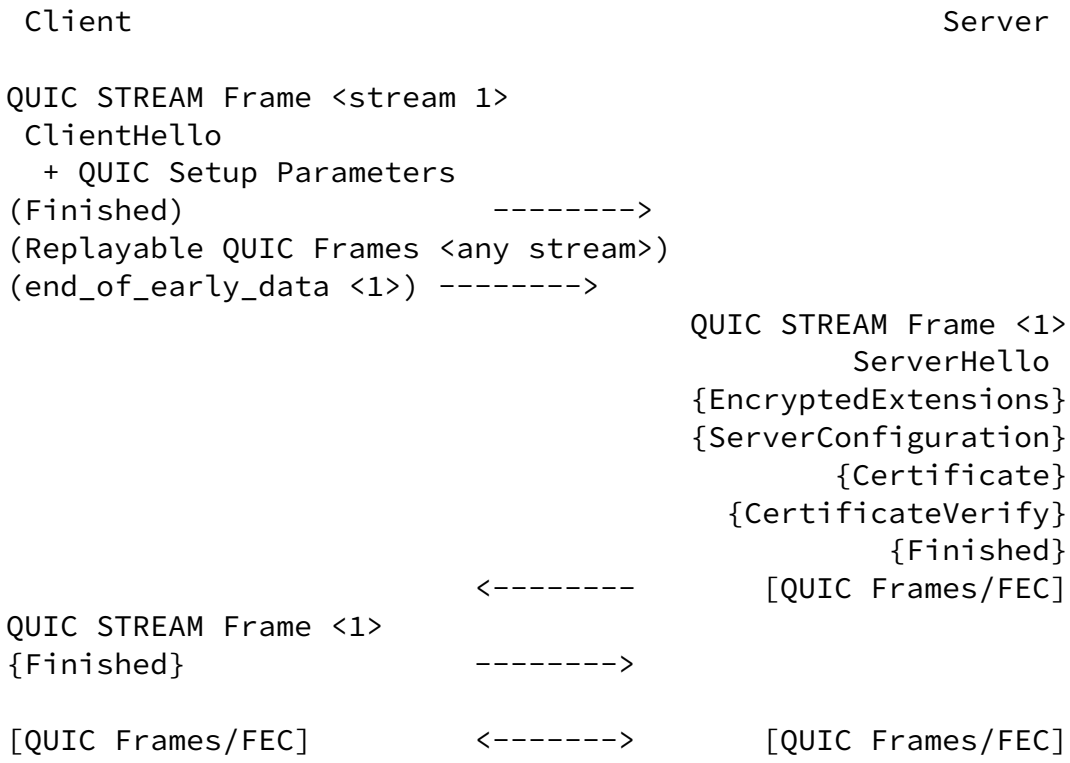


Figure 4: QUIC over TLS Handshake

The remainder of this document describes the changes to QUIC and TLS that allow the protocols to operate together.

4.1. Protocol and Version Negotiation

The QUIC version negotiation mechanism is used to negotiate the version of QUIC that is used prior to the completion of the handshake. However, this packet is not authenticated, enabling an active attacker to force a version downgrade.

To ensure that a QUIC version downgrade is not forced by an attacker, version information is copied into the TLS handshake, which provides integrity protection for the QUIC negotiation. This doesn't prevent version downgrade during the handshake, though it does prevent a

connection from completing with a downgraded version, see

Section 6.1.

ISSUE: The QUIC version negotiation has poor performance in the event that a client is forced to downgrade from their preferred version.

4.2. Source Address Validation

QUIC implementations describe a source address token. This is an opaque blob that a server provides to clients when they first use a given source address. The client returns this token in subsequent messages as a return routeability check. That is, the client returns this token to prove that it is able to receive packets at the source address that it claims.

Since this token is opaque and consumed only by the server, it can be included in the TLS 1.3 configuration identifier for 0-RTT handshakes. Servers that use 0-RTT are advised to provide new configuration identifiers after every handshake to avoid passive linkability of connections from the same client.

A server that is under load might include the same information in the cookie extension/field of a HelloRetryRequest. (Note: the current version of TLS 1.3 does not include the ability to include a cookie in HelloRetryRequest.)

5. Record Protection

Each TLS record is encapsulated in the QUIC envelope. This provides length information, which means that the length field can be dropped from the TLS record.

The sequence number used by TLS record protection is changed to deal with the potential for packets to be dropped or lost. The QUIC sequence number is used in place of the monotonically increasing TLS record sequence number. This means that the TLS record protection employed is closer to DTLS in both its form and the guarantees that are provided.

QUIC has a single, contiguous sequence number space. In comparison, TLS restarts its sequence number each time that record protection keys are changed. The sequence number restart in TLS ensures that a compromise of the current traffic keys does not allow an attacker to truncate the data that is sent after a key update by sending additional packets under the old key (causing new packets to be discarded).

QUIC does not rely on there being a continuous sequence of application data packets; QUIC uses authenticated repair mechansims that operate above the layer of encryption. QUIC can therefore operate without restarting sequence numbers.

5.1. TLS Handshake Encryption

TLS 1.3 adds encryption for handshake messages. This introduces an additional transition between different record protection keys during the handshake. A consequence of this is that it becomes more important to explicitly identify the transition from one set of keys to the next (see Section 5.2).

5.2. Key Update

Each time that the TLS record protection keys are changed, the message initiating the change could be lost. This results in subsequent packets being indecipherable to the peer that receives them. Key changes happen at the conclusion of the handshake and and immediately after a KeyUpdate message.

TLS relies on an ordered, reliable transport and therefore provides no other mechanism to ensure that a peer receives the message initiating a key change prior to receiving the subsequent messages that are protected using the new key. A similar mechanism here would introduce head-of-line blocking.

The simplest solution here is to steal a single bit from the unprotected part of the QUIC header that signals key updates, similar to how DTLS signals the epoch on each packet. The epoch bit is encoded into 0x80 of the QUIC public flags.

Each time the epoch bit changes, an attempt is made to update the keys used to read. Peers are prohibited from sending multiple KeyUpdate messages until they see a reciprocal KeyUpdate to prevent the chance that a transition is undetected as a result of two changes in this bit.

The transition from cleartext to encrypted packets is exempt from this limit of one key change. Two key changes occur during the handshake. The server sends packets in the clear, plus packets protected using handshake and application data keys. With only a single bit available to discriminate between keys, packets protected with the application data keys will have the same bit value as cleartext packets. This condition will be easily identified and handled, likely by discarding the application data, since the

encrypted packets will be highly unlikely to be valid.

5.3. Sequence Number Reconstruction

Each peer maintains a 48-bit send sequence number that is incremented with each packet that is sent (even retransmissions). The least significant 8-, 16-, 32-, or 48-bits of this number is encoded in the QUIC sequence number field in every packet. A 16-bit send epoch number is maintained; the epoch is incremented each time new record protection keying material is used. The least significant bit of the epoch number is encoded into the epoch bit (0x80) of the QUIC public flags.

A receiver maintains the same values, but recovers values based on the packets it receives. This is based on the sequence number of packets that it has received. A simple scheme predicts the receive sequence number of an incoming packet by incrementing the sequence number of the most recent packet to be successfully decrypted by one and expecting the sequence number to be within a range centered on that value. The receive epoch value is incremented each time that the epoch bit (0x80) changes.

The sequence number used for record protection is the 64-bit value obtained by concatenating the epoch and sequence number, both in network byte order.

5.4. Alternative Design: Exporters

An exporter could be used to provide keying material for a QUIC-specific record protection. This could draw on the selected cipher suite and the TLS record protection design so that the overall effort required to design and analyze is kept minimal.

One concern with using exporters is that TLS doesn't define an exporter for use prior to the end of the handshake. That means the creation of a special exporter for use in protecting 0-RTT data. That's a pretty sharp object to leave lying around, and it's not clear what the properties we could provide. (That doesn't mean that there wouldn't be demand for such a thing, the possibility has already been raised.)

An exporter-based scheme might opt not to use the handshake traffic keys to protect QUIC packets during the handshake, relying instead on separate protection for the TLS handshake records. This complicates implementations somewhat, so an exporter might still be used.

In the end, using an exporter doesn't alter the design significantly. Given the risks, a modification to the record protocol is probably safer.

6. Pre-handshake QUIC Messages

Implementations MUST NOT exchange data on any stream other than stream 1 prior to the TLS handshake completing. However, QUIC requires the use of several types of frame for managing loss detection and recovery. In addition, it might be useful to use the data acquired during the exchange of unauthenticated messages for congestion management.

The actions that a peer takes as a result of receiving an unauthenticated packet needs tobe limited. In particular, state established by these packets cannot be retained once record protection commences.

There are several approaches possible for dealing with unauthenticated packets prior to handshake completion:

- discard and ignore them
- use them, but reset any state that is established once the handshake completes
- use them and authenticate them afterwards; failing the handshake if they can't be authenticated
- save them and use them when they can be properly authenticated
- treat them as a fatal error

Different strategies are appropriate for different types of data. This document proposes that all strategies are possible depending on the type of message.

- Transport parameters and options are made usable and authenticated as part of the TLS handshake (see Section 6.1).

- Most unprotected messages are treated as fatal errors when received except for the small number necessary to permit the handshake to complete (see Section 6.2).

- Protected packets can be discarded, but can be saved and later used (see Section 6.3).

6.1. QUIC Extension

A client describes characteristics of the transport protocol it intends to conduct with the server in a new QUIC-specific extension in its ClientHello. The server uses this information to determine

whether it wants to continue the connection, request source address validation, or reject the connection. Having this information unencrypted permits this check to occur prior to committing the resources needed to complete the initial key exchange.

If the server decides to complete the connection, it generates a corresponding response and includes it in the EncryptedExtensions message.

These parameters are not confidentiality-protected when sent by the client, but the server response is protected by the handshake traffic keys. The entire exchange is integrity protected once the handshake completes.

This information is not used by TLS, but can be passed to the QUIC protocol as initialization parmeters.

The "quic_parameters" extension contains a declarative set of parameters that establish QUIC operating parameters and constrain the behaviour of a peer. The connection identifier and version are first negotiated using QUIC, and are included in the TLS handshake in order to provide integrity protection.

```
enum {
    receive_buffer(0),
    (65535)
} QuicTransportParameterType;

struct {
    QuicTransportParameterType type;
    uint32 value;
} QuicTransportParameter;

uint32 QuicVersion;

enum {
    (65535)
} QuicOption;

struct {
    uint64 connection_id;
```

```
        QuicVersion quic_version;
        QuicVersion supported_quic_versions<0..2^8-1>;
        uint32 connection_initial_window;
        uint32 stream_initial_window;
        uint32 implicit_shutdown_timeout;
        QuicTransportParameter transport_parameters<0..2^16-1>;
        QuicOption options<0..2^8-2>;
    } QuicParametersExtension;
```

This extension MUST be included if a QUIC version is negotiated. A server MUST NOT negotiate QUIC if this extension is not present.

Based on the values offered by a client a server MAY use the values in this extension to determine whether it wants to continue the connection, request source address validation, or reject the connection. Since this extension is initially unencrypted, the server can use the information prior to committing the resources needed to complete a key exchange.

If the server decides to use QUIC, this extension MUST be included in the EncryptedExtensions message.

The parameters are:

connection_id: The 64-bit connection identifier for the connection, as selected by the client.

quic_version: The currently selected QUIC version that is used for the connection. This is the version negotiated using the unauthenticated QUIC version negotiation (Section 4.1).

supported_quic_versions: This is a list of supported QUIC versions for each peer. A client sends an empty list if the version of QUIC being used is their preferred version; however, a client MUST include their preferred version if this was not negotiated using QUIC version negotiation. A server MUST include all versions that it supports in this list.

connection_initial_window: The initial value for the connection flow control window for the endpoint, in octets.

connection_initial_window: The initial value for the flow control

window of new streams created by the peer endpoint, in octets.

implicit_shutdown_timeout: The time, in seconds, that a connection can remain idle before being implicitly shutdown.

transport_parameters: A list of parameters for the QUIC connection, expressed as key-value pairs of arbitrary length. The QuicTransportParameterType identifies each parameter; duplicate types are not permitted and MUST be rejected with a fatal illegal_parameter alert. Type values are taken from a single space that is shared by all QUIC versions.

ISSUE: There is currently no way to update the value of parameters once the connection has started. QUIC crypto provided a SCFG message that could be sent after the connection was established.

options: A list of options that can be negotiated for a given connection. These are set during the initial handshake and are fixed thereafter. These options are used to enable or disable optional features in the protocol. The set of features that are supported across different versions might vary. A client SHOULD include all options that it is willing to use. The server MAY select any subset of those options that apply to the version of QUIC that it selects. Only those options selected by the server are available for use.

Note: This sort of optional behaviour seems like it could be accommodated adequately by defining new versions of QUIC for each experiment. However, as an evolving protocol, multiple experiments need to be conducted concurrently and continuously. The options parameter provides a flexible way to regulate which experiments are enabled on a per-connection basis.

6.2. Unprotected Frames Prior to Handshake Completion

This section describes the handling of messages that are sent and received prior to the completion of the TLS handshake.

Sending and receiving unprotected messages is hazardous. Unless expressly permitted, receipt of an unprotected message of any kind MUST be treated as a fatal error.

6.2.1. STREAM Frames

"STREAM" frames for stream 1 are permitted. These carry the TLS handshake messages.

Receiving unprotected "STREAM" frames that do not contain TLS handshake messages MUST be treated as a fatal error.

6.2.2. ACK Frames

"ACK" frames are permitted prior to the handshake being complete. However, an unauthenticated "ACK" frame can only be used to obtain NACK ranges. Timestamps MUST NOT be included in an unprotected ACK frame, since these might be modified by an attacker with the intent of altering congestion control response. Information on FEC-revived packets is redundant, since use of FEC in this phase is prohibited.

"ACK" frames MAY be sent a second time once record protection is enabled. Once protected, timestamps can be included.

Editor's Note: This prohibition might be a little too strong, but this is the only obviously safe option. If the amount of damage that an attacker can do by modifying timestamps is limited, then it might be OK to permit the inclusion of timestamps. Note that an attacker need not be on-path to inject an ACK.

6.2.3. WINDOW_UPDATE Frames

Sending a "WINDOW_UPDATE" on stream 1 might be necessary to permit the completion of the TLS handshake, particularly in cases where the certification path is lengthy. To avoid stalling due to flow control exhaustion, "WINDOW_UPDATE" frames with stream 1 are permitted.

Receiving a "WINDOW_UPDATE" frame on streams other than 1 MUST be treated as a fatal error.

Stream 1 is exempt from the connection-level flow control window.

The position of the flow control window MUST be reset to defaults once the TLS handshake is complete. This might result in the window position for either the connection or stream 1 being smaller than the number of octets that have been sent on those streams. A "WINDOW_UPDATE" frame might therefore be necessary to prevent the connection from being stalled.

Note: This is only potentially problematic for servers, who might need to send large certificate chains. In other cases, this is unlikely given that QUIC - like HTTP [RFC7230] - is a protocol where the server is unable to exercise the opportunity TLS presents to send first.

If a server has a large certificate chain, or later modifications or extensions to QUIC permit the server to send first, a client might reduce the chance of stalling due to flow control in this first round trip by setting larger values for the initial stream and connection flow control windows using the "quic_parameters" extension (Section 6.1).

Editor's Note: Unlike "ACK", the prohibition on "WINDOW_UPDATE" is much less of an imposition on implementations. And, given that a spurious "WINDOW_UPDATE" might be used to create a great deal of memory pressure on an endpoint, the restriction seems justifiable. Besides, I understand this one a lot better.

6.2.4. FEC Packets

FEC packets MUST NOT be sent prior to completing the TLS handshake. Endpoints MUST treat receipt of an unprotected FEC packet as a fatal error.

6.3. Protected Frames Prior to Handshake Completion

Due to reordering and loss, protected packets might be received by an endpoint before the final handshake messages are received. If these can be decrypted successfully, such packets MAY be stored and used once the handshake is complete.

Unless expressly permitted below, encrypted packets MUST NOT be used prior to completing the TLS handshake, in particular the receipt of a valid Finished message and any authentication of the peer. If packets are processed prior to completion of the handshake, an attacker might use the willingness of an implementation to use these packets to mount attacks.

TLS handshake messages are covered by record protection during the

handshake, once key agreement has completed. This means that

protected messages need to be decrypted to determine if they are TLS handshake messages or not. Similarly, "ACK" and "WINDOW_UPDATE" frames might be needed to successfully complete the TLS handshake.

Any timestamps present in "ACK" frames MUST be ignored rather than causing a fatal error. Timestamps on protected frames MAY be saved and used once the TLS handshake completes successfully.

An endpoint MUST save the last protected "WINDOW_UPDATE" frame it receives for each stream and apply the values once the TLS handshake completes.

Editor's Note: Ugh. This last one is pretty ugly. Maybe we should just make the TLS handshake exempt from flow control up to the Finished message. Then we can prohibit unauthenticated "WINDOW_UPDATE" messages. We would still likely want to account for the packets sent and received, since to do otherwise would create some hairy special cases. That means that stalling is possible, but it means that we can avoid ugly rules like the above.

7. Connection ID

The QUIC connection identifier serves to identify a connection and to allow a server to resume an existing connection from a new client address in case of mobility events. However, this creates an identifier that a passive observer [RFC7258] can use to correlate connections.

TLS 1.3 offers connection resumption using pre-shared keys, which also allows a client to send 0-RTT application data. This mode could be used to continue a connection rather than rely on a publicly visible correlator. This only requires that servers produce a new ticket on every connection and that clients do not resume from the same ticket more than once.

The advantage of relying on 0-RTT modes for mobility events is that this is also more robust. If the new point of attachment results in contacting a new server instance - one that lacks the session state - then a fallback is easy.

The main drawback with a clean restart or anything resembling a restart is that accumulated state can be lost. Aside from progress on incomplete requests, the state of the HPACK header compression table could be quite valuable. Existing QUIC implementations use the connection ID to route packets to the server that is handling the connection, which avoids this sort of problem.

A lightweight state resurrection extension might be used to avoid having to recreate any expensive state.

8. Security Considerations

There are likely to be some real clangers here eventually, but the current set of issues is well captured in the relevant sections of the main text.

Never assume that because it isn't in the security considerations section it doesn't affect security. Most of this document does.

9. IANA Considerations

This document has no IANA actions. Yet.

10. References

10.1. Normative References

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-11 (work in progress), December 2015.

[I-D.tsvwg-quic-protocol]
Hamilton, R., Iyengar, J., Swett, I., and A. Wilk, "QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2", draft-tsvwg-quic-protocol-02 (work in progress), January 2016.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119,

DOI 10.17487/RFC2119, March 1997,
<http://www.rfc-editor.org/info/rfc2119>.

[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, <http://www.rfc-editor.org/info/rfc7301>.

10.2. Informative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <http://www.rfc-editor.org/info/rfc793>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <http://www.rfc-editor.org/info/rfc7258>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <http://www.rfc-editor.org/info/rfc7540>.

Appendix A. Acknowledgments

Christian Huitema's knowledge of QUIC is far better than my own. This would be even more inaccurate and useless if not for his assistance. This document has variously benefited from a long series of discussions with Ryan Hamilton, Jana Iyengar, Adam Langley, Roberto Peon, Ian Swett, and likely many others who are merely forgotten by a faulty meat computer.

Authors' Addresses

Martin Thomson

Mozilla

Email: martin.thomson@gmail.com

Ryan Hamilton
Google

Email: rch@google.com