Workgroup: Network Working Group
Internet-Draft:
draft-ietf-quic-manageability-08
Published: 2 November 2020
Intended Status: Informational
Expires: 6 May 2021
Authors: M. Kuehlewind
Ericsson
B. Trammell
Google

Manageability of the QUIC Transport Protocol

Abstract

This document discusses manageability of the QUIC transport protocol, focusing on caveats impacting network operations involving QUIC traffic. Its intended audience is network operators, as well as content providers that rely on the use of QUIC-aware middleboxes, e.g. for load balancing.

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 6 May 2021.

Table of Contents

1. Introduction

QUIC [QUIC-TRANSPORT] is a new transport protocol currently under development in the IETF QUIC working group, focusing on support of semantics as needed for HTTP/2 [QUIC-HTTP]. Based on current deployment practices, QUIC is encapsulated in UDP and encrypted by default. The current version of QUIC integrates TLS [QUIC-TLS] to encrypt all payload data and most control information.

Given that QUIC is an end-to-end transport protocol, all information in the protocol header, even that which can be inspected, is not meant to be mutable by the network, and is therefore integrity-protected. While less information is visible to the network than for TCP, integrity protection can also simplify troubleshooting because none of the nodes on the network path can modify the transport layer information.

This document provides guidance for network operation on the management of QUIC traffic. This includes guidance on how to interpret and utilize information that is exposed by QUIC to the network as well as explaining requirement and assumptions that the QUIC protocol design takes toward the expected network treatment. It also discusses how common network management practices will be impacted by QUIC.

Since QUIC's wire image [WIRE-IMAGE] is integrity protected and not modifiable on path, in-network operations are not possible without terminating the QUIC connection, for instance using a back-to-back proxy. Proxy operations are not in scope for this document. QUIC proxies must be fully-fledged QUIC endpoints, implementing the transport as defined in [QUIC-TRANSPORT] and [QUIC-TLS] as well as proxy-relevant semantics for the application(s) running over QUIC (e.g. HTTP/3 as defined in [QUIC-HTTP]).

Network management is not a one-size-fits-all endeavour: practices considered necessary or even mandatory within enterprise networks with certain compliance requirements, for example, would be impermissible on other networks without those requirements. This document therefore does not make any specific recommendations as to which practices should or should not be applied; for each practice, it describes what is and is not possible with the QUIC transport protocol as defined.

QUIC is at the moment very much a moving target. This document refers the state of the QUIC working group drafts as well as to changes under discussion, via issues and pull requests in GitHub current as of the time of writing.

1.1. Notational 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.

2. Features of the QUIC Wire Image

In this section, we discusses those aspects of the QUIC transport protocol that have an impact on the design and operation of devices that forward QUIC packets. Here, we are concerned primarily with the unencrypted part of QUIC's wire image [WIRE-IMAGE], which we define as the information available in the packet header in each QUIC packet, and the dynamics of that information. Since QUIC is a versioned protocol, the wire image of the header format can also change from version to version. However, at least the mechanism by which a receiver can determine which version is used and the meaning and location of fields used in the version negotiation process is invariant [QUIC-INVARIANTS].

This document describes only version 1 the QUIC protocol, whose wire image is fully defined in [QUIC-TRANSPORT] and [QUIC-TLS]. Note that features of the wire image described herein and in those documents may change in future versions of the protocol, and cannot be used to identify QUIC as a protocol or to infer the behavior of future versions of QUIC. Section 9.1 provides non-normative guidance on the identification of QUIC version 1 packets compared to other deployed versions at the date if publication.

2.1. QUIC Packet Header Structure

QUIC packets may have either a long header, or a short header. The first bit of the QUIC header us the Header Form bit, and indicates which type of header is present.

The long header exposes more information. It is used during connection establishment, including version negotiation, retry, and 0-RTT data. It contains a version number, as well as source and destination connection IDs for grouping packets belonging to the same flow. The definition and location of these fields in the QUIC long header are invariant for future versions of QUIC, although future versions of QUIC may provide additional fields in the long header [QUIC-INVARIANTS].

Short headers are used after connection establishment, and contain only an optional destination connection ID and the spin bit for RTT measurement.

The following information is exposed in QUIC packet headers:

- "fixed bit": the second most significant bit of the first octet most QUIC packets of the current version is currently set to 1, for demultiplexing with other UDP-encapsulated protocols.

- latency spin bit: the third most significant bit of first octet in the short packet header. The spin bit is set by endpoints such that tracking edge transitions can be used to passively observe end-to-end RTT. See Section 3.7.2 for further details.

- header type: the long header has a 2 bit packet type field following the Header Form and fixed bits. Header types correspond to stages of the handshake; see Section 17.2 of [QUIC-TRANSPORT] for details.

- version number: the version number present in the long header, and identifies the version used for that packet. Note that during Version Negotiation (see Section 2.8, and Section 17.2.1 of [QUIC-TRANSPORT], the version number field has a special value (0x00000000) that identifies the packet as a Version Negotiation packet. QUIC versions that start with 0xff are IETF drafts. QUIC versions that start with 0x0000 are reserved for IETF consensus documents, for example the QUIC version 1 is expected to use version 0x00000001.

- source and destination connection ID: short and long packet headers carry a destination connection ID, a variable-length field that can be used to identify the connection associated with a QUIC packet, for load-balancing and NAT rebinding purposes; see Section 4.3 and Section 2.6. Long packet headers additionally carry a source connection ID. The source connection ID corresponds to the destination connection ID the source would like to have on packets sent to it, and is only present on long packet headers. On long header packets, the length of the connection IDs is also present; on short header packets, the length of the destination connection ID is implicit.

- length: the length of the remaining QUIC packet after the length field, present on long headers. This field is used to implement coalesced packets during the handshake (see Section 2.2).

- token: Initial packets may contain a token, a variable-length opaque value optionally sent from client to server, used for validating the client's address. Retry packets also contain a token, which can be used by the client in an Initial packet on a subsequent connection attempt. The length of the token is explicit in both cases.

Retry (Section 17.2.5 of [QUIC-TRANSPORT]) and Version Negotiation (Section 17.2.1 of [QUIC-TRANSPORT]) packets are not encrypted or obfuscated in any way. For other kinds of packets, other information in the packet headers is cryptographically obfuscated:

- packet number: All packets except Version Negotiation and Retry packets have an associated packet number; however, this packet number is encrypted, and therefore not of use to on-path observers. The offset of the packet number is encoded in the header for packets with long headers, while it is implicit (depending on Destination Connection ID length) in short header packets. The length of the packet number is cryptographically obfuscated.

- key phase: The Key Phase bit, present in short headers, specifies the keys used to encrypt the packet, supporting key rotation. The Key Phase bit is cryptographically obfuscated.

2.2. Coalesced Packets

Multiple QUIC packets may be coalesced into a UDP datagram, with a datagram carrying one or more long header packets followed by zero or one short header packets. When packets are coalesced, the Length fields in the long headers are used to separate QUIC packets. The length header field is variable length and its position in the header is also variable depending on the length of the source and destination connection ID. See Section 4.6 of [QUIC-TRANSPORT].

2.3. Use of Port Numbers

Applications that have a mapping for TCP as well as QUIC are expected to use the same port number for both services. However, as with TCP-based services, especially when application layer information is encrypted, there is no guarantee that a specific application will use the registered port, or the used port is carrying traffic belonging to the respective registered service. For example, [QUIC-TRANSPORT] specifies the use of Alt-Svc for discovery of QUIC/HTTP services on other ports.

Further, as QUIC has a connection ID, it is also possible to maintain multiple QUIC connections over one 5-tuple. However, if the connection ID is not present in the packet header, all packets of the 5-tuple belong to the same QUIC connection.

2.4. The QUIC handshake

New QUIC connections are established using a handshake, which is distinguishable on the wire and contains some information that can be passively observed.

To illustrate the information visible in the QUIC wire image during the handshake, we first show the general communication pattern visible in the UDP datagrams containing the QUIC handshake, then examine each of the datagrams in detail.

In the nominal case, the QUIC handshake can be recognized on the wire through at least four datagrams we'll call "QUIC Client Hello", "QUIC Server Hello", and "Initial Completion", and "Handshake Completion", for purposes of this illustration, as shown in Figure 1.

Packets in the handshake belong to three separate cryptographic and transport contexts ("Initial", which contains observable payload, and "Handshake" and "1-RTT", which do not). QUIC packets in separate contexts during the handshake are generally coalesced (see Section 2.2) in order to reduce the number of UDP datagrams sent during the handshake.

As shown here, the client can send 0-RTT data as soon as it has sent its Client Hello, and the server can send 1-RTT data as soon as it has sent its Server Hello.

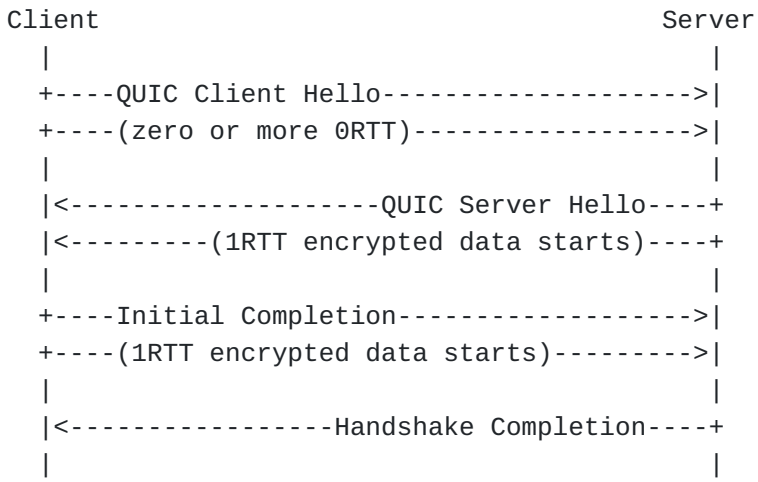

Figure 1: General communication pattern visible in the QUIC handshake

A typical handshake starts with the client sending of a QUIC Client Hello datagram as shown in Figure 2, which elicits a QUIC Server Hello datagram as shown in Figure 3 typically containing three packets: an Initial packet with the Server Hello, a Handshake packet with the rest of the server's side of the TLS handshake, and initial 1-RTT data, if present.

The content of QUIC Initial packets are encrypted using Initial Secrets, which are derived from a per-version constant and the client's destination connection ID; they are therefore observable by any on-path device that knows the per-version constant; we therefore

consider these as visible in our illustration. The content of QUIC Handshake packets are encrypted using keys established during the initial handshake exchange, and are therefore not visible.

Initial, Handshake, and the Short Header packets transmitted after the handshake belong to cryptographic and transport contexts. The Initial Completion Figure 4 and the Handshake Completion Figure 5 datagrams finish these first two contexts, by sending the final acknowledgment and finishing the transmission of CRYPTO frames.

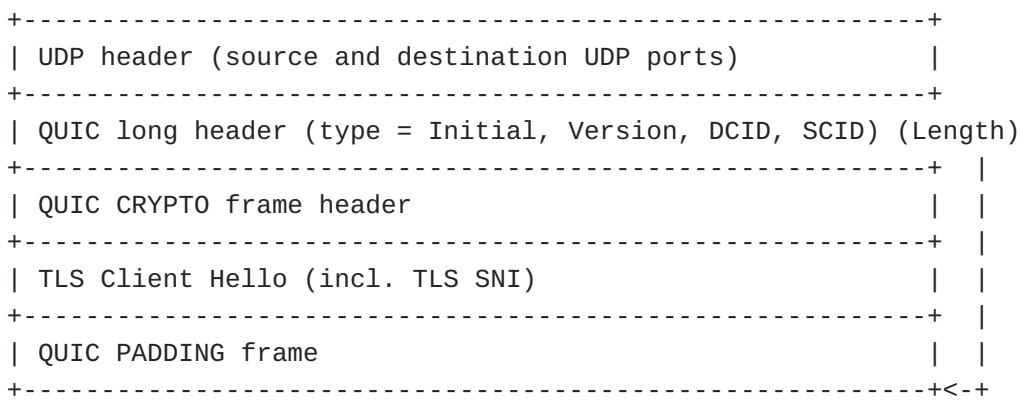

Figure 2: Typical 1-RTT QUIC Client Hello datagram pattern

The Client Hello datagram exposes version number, source and destination connection IDs in the clear. Information in the TLS Client Hello frame, including any TLS Server Name Indication (SNI) present, is obfuscated using the Initial secret. The QUIC PADDING frame shown here may be present to ensure the Client Hello datagram has a minimum size of 1200 octets, to mitigate the possibility of handshake amplification. Note that the location of PADDING is implementation-dependent, and PADDING frames may not appear in the Initial packet in a coalesced packet.

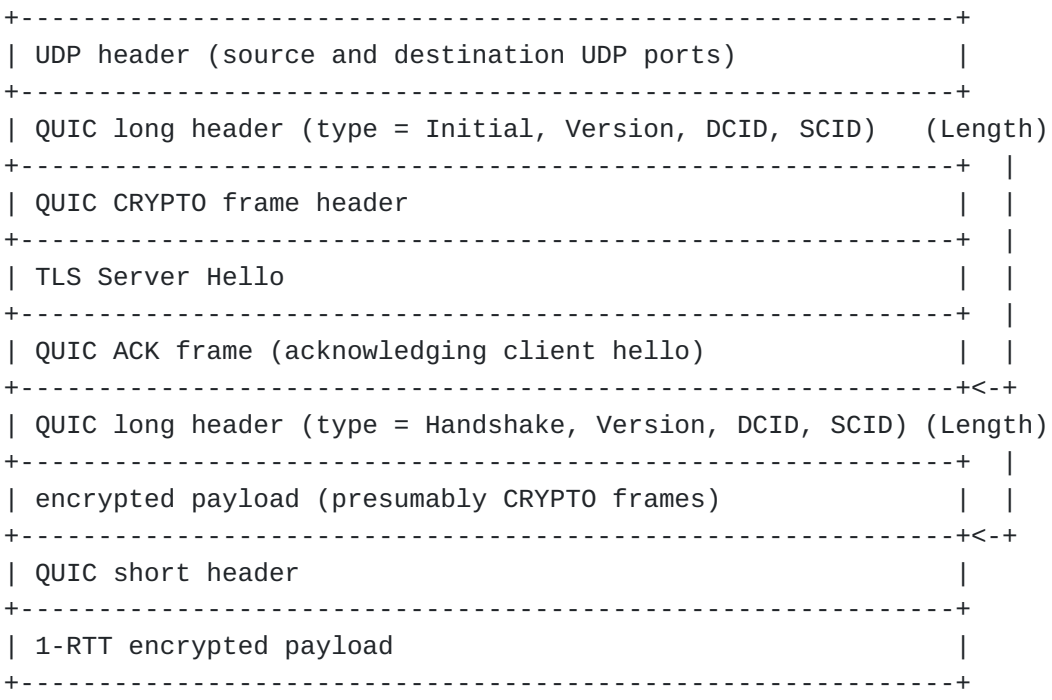

Figure 3: Typical QUIC Server Hello datagram pattern

The Server Hello datagram also exposes version number, source and destination connection IDs and information in the TLS Server Hello message which is obfuscated using the Initial secret.

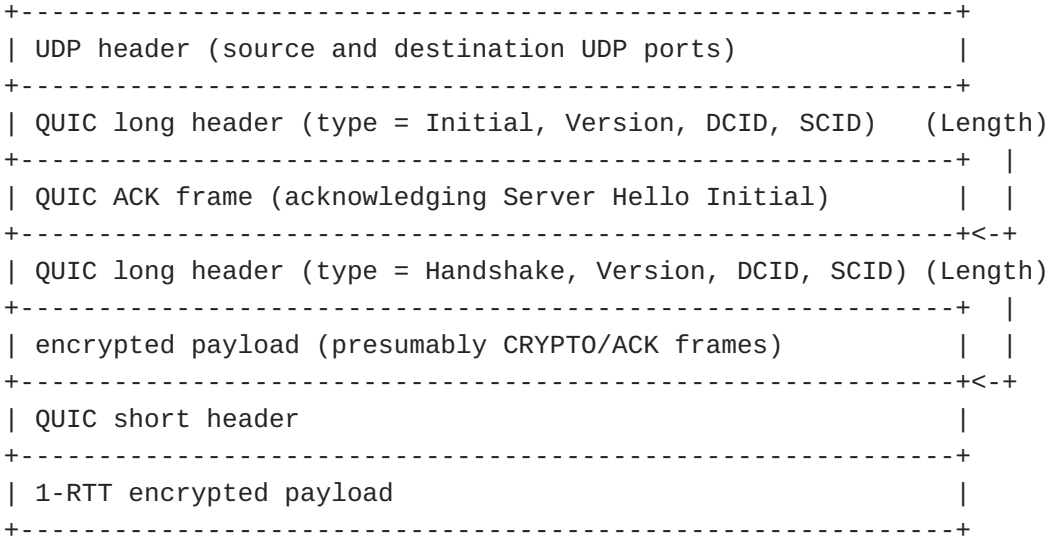

Figure 4: Typical QUIC Initial Completion datagram pattern

The Initial Completion datagram does not expose any additional information; however, recognizing it can be used to determine that a handshake has completed (see Section 3.2), and for three-way handshake RTT estimation as in Section 3.7.

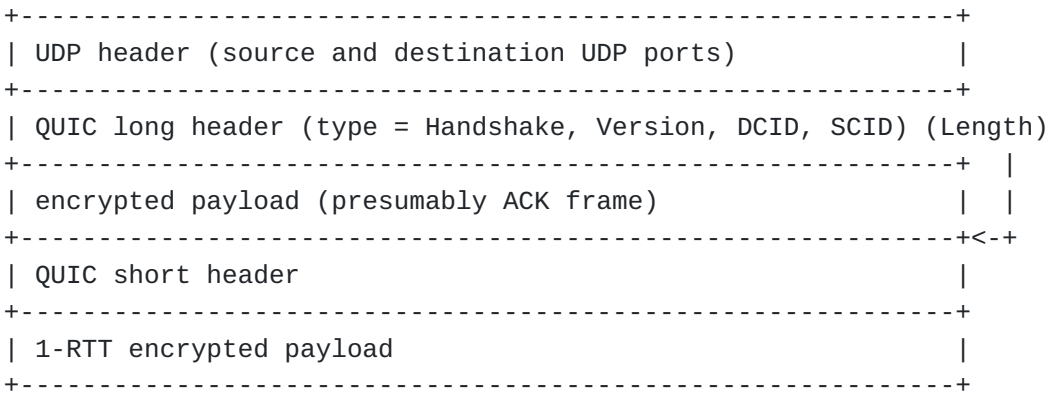

Figure 5: Typical QUIC Handshake Completion datagram pattern

Similar to Initial Completion, Handshake Completion also exposes no additional information; observing it serves only to determine that the handshake has completed.

When the client uses 0-RTT connection resumption, 0-RTT data may also be seen in the QUIC Client Hello datagram, as shown in Figure 6.

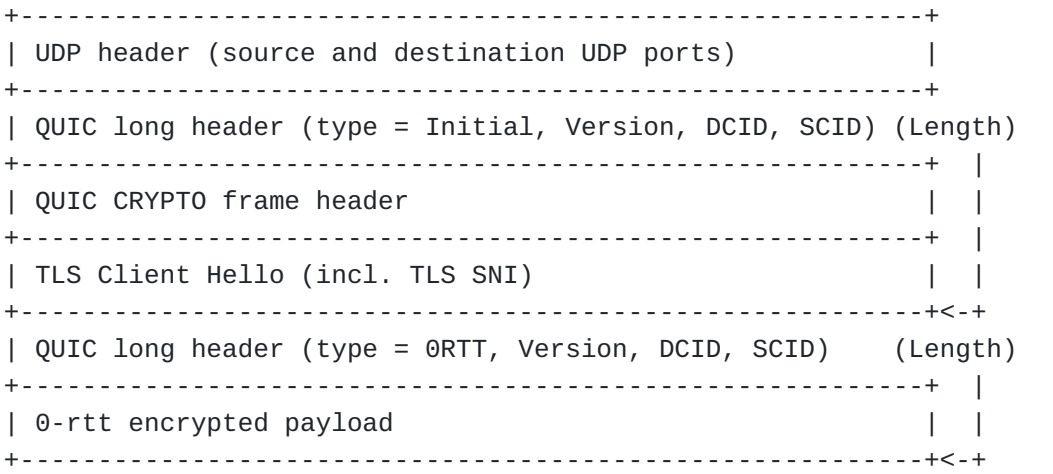

Figure 6: Typical 0-RTT QUIC Client Hello datagram pattern

In a 0-RTT QUIC Client Hello datagram, the PADDING frame is only present if necessary to increase the size of the datagram with 0RTT data to at least 1200 bytes. Additional datagrams containing only 0-RTT protected long header packets may be sent from the client to the server after the Client Hello datagram, containing the rest of the 0-RTT data. The amount of 0-RTT protected data is limited by the initial congestion window, typically around 10 packets [RFC6928].

2.5. Integrity Protection of the Wire Image

As soon as the cryptographic context is established, all information in the QUIC header, including information exposed in the packet header, is integrity protected. Further, information that was sent and exposed in handshake packets sent before the cryptographic context was established are validated later during the cryptographic handshake. Therefore, devices on path MUST NOT change any information or bits in QUIC packet headers, since alteration of header information will lead to a failed integrity check at the receiver, and can even lead to connection termination.

2.6. Connection ID and Rebinding

The connection ID in the QUIC packet headers allows routing of QUIC packets at load balancers on other than five-tuple information, ensuring that related flows are appropriately balanced together; and to allow rebinding of a connection after one of the endpoint's addresses changes - usually the client's, in the case of the HTTP binding. Client and server negotiate connection IDs during the handshake; typically, however, only the server will request a connection ID for the lifetime of the connection. Connection IDs for either endpoint may change during the lifetime of a connection, with the new connection ID being negotiated via encrypted frames. See Section 5.1 of [QUIC-TRANSPORT].

Server-generated connection IDs should seek to obscure any encoding, of routing identities or any other information. Exposing the server mapping would allow linkage of multiple IP addresses to the same host if the server also supports migration. Furthermore, this opens an attack vector on specific servers or pools.

The best way to obscure an encoding is to appear random to observers, which is most rigorously achieved with encryption. Even when encrypted, a scheme could embed the unencrypted length of the Connection ID in the Connection ID itself, instead of remembering it, e.g. by using the first few bits to indicate a certain size of a well-known set of possible sizes with multiple values that indicate the same size but are selected randomly.

[QUIC_LB] further specified possible algorithms to generate Connection IDs at load balancers.

2.7. Packet Numbers

The packet number field is always present in the QUIC packet header; however, it is always encrypted. The encryption key for packet number protection on handshake packets sent before cryptographic context establishment is specific to the QUIC version, while packet number protection on subsequent packets uses secrets derived from

the end-to-end cryptographic context. Packet numbers are therefore not part of the wire image that is visible to on-path observers.

2.8. Version Negotiation and Greasing

Version negotiation is not protected, given the used protection mechanism can change with the version. However, the choices provided in the list of version in the Version Negotiation packet will be validated as soon as the cryptographic context has been established. Therefore any manipulation of this list will be detected and will cause the endpoints to terminate the connection.

Also note that the list of versions in the Version Negotiation packet may contain reserved versions. This mechanism is used to avoid ossification in the implementation on the selection mechanism. Further, a client may send a Initial Client packet with a reserved version number to trigger version negotiation. In the Version Negotiation packet the connection ID and packet number of the Client Initial packet are reflected to provide a proof of return-routability. Therefore changing these information will also cause the connection to fail.

QUIC is expected to evolve rapidly, so new versions, both experimental and IETF standard versions, will be deployed in the Internet more often than with traditional Internet- and transport-layer protocols. Using a particular version number to recognize valid QUIC traffic is likely to persistently miss a fraction of QUIC flows and completely fail in the multi-year timeframe so therefore not recommended.

3. Network-visible information about QUIC flows

This section addresses the different kinds of observations and inferences that can be made about QUIC flows by a passive observer in the network based on the wire image in Section 2. Here we assume a bidirectional observer (one that can see packets in both directions in the sequence in which they are carried on the wire) unless noted.

3.1. Identifying QUIC traffic

The QUIC wire image is not specifically designed to be distinguishable from other UDP traffic.

The only application binding defined by the IETF QUIC WG is HTTP/3 [QUIC-HTTP] at the time of this writing; however, many other applications are currently being defined and deployed over QUIC, so an assumption that all QUIC traffic is HTTP/3 is not valid. HTTP over QUIC uses UDP port 443 by default, although URLs referring to resources available over HTTP over QUIC may specify alternate port

numbers. Simple assumptions about whether a given flow is using QUIC based upon a UDP port number may therefore not hold; see also [RFC7605] section 5.

While the second most significant bit (0x40) of the first octet is set to 1 in most QUIC packets of the current version (see Section 2.1), this method of recognizing QUIC traffic is NOT RECOMMENDED. First, it only provides one bit of information and is quite prone to collide with UDP-based protocols other than those that this static bit is meant to allow multiplexing with. Second, this feature of the wire image is not invariant [QUIC-INVARIANTS] and may change in future versions of the protocol, or even be negotiated after handshake via future transport parameters.

3.1.1. Identifying Negotiated Version

An in-network observer assuming that a set of packets belongs to a QUIC flow can infer the version number in use by observing the handshake: an Initial packet with a given version from a client to which a server responds with an Initial packet with the same version implies acceptance of that version.

Negotiated version cannot be identified for flows for which a handshake is not observed, such as in the case of connection migration; however, these flows can be associated with flows for which a version has been identified; see Section 3.4.

This document focuses on QUIC Version 1, and this section applies only to packets belonging to Version 1 QUIC flows; for purposes of on-path observation, it assumes that these packets have been identified as such through the observation of a version negotiation.

3.1.2. Rejection of Garbage Traffic

A related question is whether a first packet of a given flow on known QUIC-associated port is a valid QUIC packet, in order to support in-network filtering of garbage UDP packets (reflection attacks, random backscatter). While heuristics based on the first byte of the packet (packet type) could be used to separate valid from invalid first packet types, the deployment of such heuristics is not recommended, as packet types may have different meanings in future versions of the protocol.

3.2. Connection confirmation

Connection establishment uses Initial, Handshake, and Retry packets containing a TLS handshake. Connection establishment can therefore be detected using heuristics similar to those used to detect TLS over TCP. A client using 0-RTT connection may also send data packets in 0-RTT Protected packets directly after the Initial packet

containing the TLS Client Hello. Since these packets may be reordered in the network, note that 0-RTT Protected data packets may be seen before the Initial packet.

Note that clients send Initial packets before servers do, servers send Handshake packets before clients do, and only clients send Initial packets with tokens, so the sides of a connection can be generally be confirmed by an on-path observer. An attempted connection after Retry can be detected by correlating the token on the Retry with the token on the subsequent Initial packet.

3.3. Application Identification

The cleartext TLS handshake may contain Server Name Indication (SNI) [RFC6066], by which the client reveals the name of the server it intends to connect to, in order to allow the server to present a certificate based on that name. It may also contain information from Application-Layer Protocol Negotiation (ALPN) [RFC7301], by which the client exposes the names of application-layer protocols it supports; an observer can deduce that one of those protocols will be used if the connection continues.

Work is currently underway in the TLS working group to encrypt the SNI in TLS 1.3 [TLS-ESNI]. If used with QUIC, this would make SNI-based application identification impossible through passive measurement.

3.3.1. Extracting Server Name Indication (SNI) Information

If the SNI is not encrypted it can be derived from the QUIC Initial packet by calculating the Initial Secret to decrypt the packet payload and parse the QUIC CRYPTO Frame containing the TLS ClientHello.

As both the initial salt for the Initial Secret as well as CRYPTO frame itself are version-specific, the first step is always to parse the version number (second to sixth byte of the long header). Note that only long header packets carry the version number, so it is necessary to also check the if first bit of the QUIC packet is set to 1, indicating a long header.

Note, that proprietary QUIC versions, that have been deployed before standardization, might not set the first bit in a QUIC long header packets to 1. To parse these versions example code is provided in the appendix (see Section 9.1), however, it is expected that these versions will gradually disappear over time.

When the version has been identified as QUIC version 1, the packet type needs to be verified as an Initial packet by checking that the third and fourth bit of the header are both set to 0. Then the

Client Destination Connection ID needs to be extracted to calculate the Initial Secret together with the version specific initial salt, as described in [QUIC-TLS]. The length of the connection ID is indicated in the 6th byte of the header followed by the connection ID itself.

To determine the end of the header and find the start of the payload further the packet number length, the source connection ID length, as well as the token length need to be extracted. The packet number length is defined by the seventh and eight bits of the header as described in section 17.2. of [QUIC-TRANSPORT]. The source connection ID length is specified in the byte after the destination connection ID. And the token length, which follows the source connection ID, is a variable length integer as specified in section 16 of [QUIC-TRANSPORT].

Finally after decryption, the Initial Client packet can be parsed to detect the CRYPTO frame that contains the TLS Client Hello, which then can be respectively parsed similar as for all other TLS connections. The Initial client packet may contain other frames, so the first byte of each frame need to be checked to identify the frame type and the skip over the frame. Note that the length of the frames is dependent on the frame type. Usually for QUIC version 1, the packet is expected to only carry the CRYPTO frame and optionally padding frames. However, padding which is one byte of zeros, may also occur before or after the CRYPTO frame.

3.4. Flow association

The QUIC Connection ID (see Section 2.6) is designed to allow an on-path device such as a load-balancer to associate two flows as identified by five-tuple when the address and port of one of the endpoints changes; e.g. due to NAT rebinding or server IP address migration. An observer keeping flow state can associate a connection ID with a given flow, and can associate a known flow with a new flow when when observing a packet sharing a connection ID and one endpoint address (IP address and port) with the known flow.

However, since the connection ID may change multiple times during the lifetime of a flow, and the negotiation of connection ID changes is encrypted, packets with the same 5-tuple but different connection IDs may or may not belong to the same connection.

The connection ID value should be treated as opaque; see Section 4.3 for caveats regarding connection ID selection at servers.

3.5. Flow teardown

QUIC does not expose the end of a connection; the only indication to on-path devices that a flow has ended is that packets are no longer

observed. Stateful devices on path such as NATs and firewalls must therefore use idle timeouts to determine when to drop state for QUIC flows, see further section Section 4.1.

3.6. Flow symmetry measurement

QUIC explicitly exposes which side of a connection is a client and which side is a server during the handshake. In addition, the symmetry of a flow (whether primarily client-to-server, primarily server-to-client, or roughly bidirectional, as input to basic traffic classification techniques) can be inferred through the measurement of data rate in each direction. While QUIC traffic is protected and ACKs may be padded, padding is not required.

3.7. Round-Trip Time (RTT) Measurement

Round-trip time of QUIC flows can be inferred by observation once per flow, during the handshake, as in passive TCP measurement; this requires parsing of the QUIC packet header and recognition of the handshake, as illustrated in Section 2.4. It can also be inferred during the flow's lifetime, if the endpoints use the spin bit facility described below and in [QUIC-TRANSPORT], section 17.3.1.

3.7.1. Measuring initial RTT

In the common case, the delay between the Initial packet containing the TLS Client Hello and the Handshake packet containing the TLS Server Hello represents the RTT component on the path between the observer and the server. The delay between the TLS Server Hello and the Handshake packet containing the TLS Finished message sent by the client represents the RTT component on the path between the observer and the client. While the client may send 0-RTT Protected packets after the Initial packet during 0-RTT connection re-establishment, these can be ignored for RTT measurement purposes.

Handshake RTT can be measured by adding the client-to-observer and observer-to-server RTT components together. This measurement necessarily includes any transport and application layer delay (the latter mainly caused by the asymmetric crypto operations associated with the TLS handshake) at both sides.

3.7.2. Using the Spin Bit for Passive RTT Measurement

The spin bit provides an additional method to measure per-flow RTT from observation points on the network path throughout the duration of a connection. Endpoint participation in spin bit signaling is optional in QUIC. That is, while its location is fixed in this version of QUIC, an endpoint can unilaterally choose to not support "spinning" the bit. Use of the spin bit for RTT measurement by devices on path is only possible when both endpoints enable it. Some

endpoints may disable use of the spin bit by default, others only in specific deployment scenarios, e.g. for servers and clients where the RTT would reveal the presence of a VPN or proxy. To avoid making these connections identifiable based on the usage of the spin bit, it is recommended that all endpoints randomly disable "spinning" for at least one eighth of connections, even if otherwise enabled by default. An endpoint not participating in spin bit signaling for a given connection can use a fixed spin value for the duration of the connection, or can set the bit randomly on each packet sent.

When in use and a QUIC flow sends data continuously, the latency spin bit in each direction changes value once per round-trip time (RTT). An on-path observer can observe the time difference between edges (changes from 1 to 0 or 0 to 1) in the spin bit signal in a single direction to measure one sample of end-to-end RTT.

Note that this measurement, as with passive RTT measurement for TCP, includes any transport protocol delay (e.g., delayed sending of acknowledgements) and/or application layer delay (e.g., waiting for a response to be generated). It therefore provides devices on path a good instantaneous estimate of the RTT as experienced by the application. A simple linear smoothing or moving minimum filter can be applied to the stream of RTT information to get a more stable estimate.

However, application-limited and flow-control-limited senders can have application and transport layer delay, respectively, that are much greater than network RTT. When the sender is application-limited and e.g. only sends small amount of periodic application traffic, where that period is longer than the RTT, measuring the spin bit provides information about the application period, not the network RTT.

Since the spin bit logic at each endpoint considers only samples from packets that advance the largest packet number, signal generation itself is resistant to reordering. However, reordering can cause problems at an observer by causing spurious edge detection and therefore inaccurate (i.e., lower) RTT estimates, if reordering occurs across a spin-bit flip in the stream.

Simple heuristics based on the observed data rate per flow or changes in the RTT series can be used to reject bad RTT samples due to lost or reordered edges in the spin signal, as well as application or flow control limitation; for example, QoF [TMA-QOF] rejects component RTTs significantly higher than RTTs over the history of the flow. These heuristics may use the handshake RTT as an initial RTT estimate for a given flow. Usually such heuristics would also detect if the spin is either constant or randomly set for a connection.

An on-path observer that can see traffic in both directions (from client to server and from server to client) can also use the spin bit to measure "upstream" and "downstream" component RTT; i.e, the component of the end-to-end RTT attributable to the paths between the observer and the server and the observer and the client, respectively. It does this by measuring the delay between a spin edge observed in the upstream direction and that observed in the downstream direction, and vice versa.

4. Specific Network Management Tasks

In this section, we review specific network management and measurement techniques and how QUIC's design impacts them.

4.1. Stateful treatment of QUIC traffic

Stateful treatment of QUIC traffic (e.g., at a firewall or NAT middlebox) is possible through QUIC traffic and version identification (Section 3.1) and observation of the handshake for connection confirmation (Section 3.2). The lack of any visible end-of-flow signal (Section 3.5) means that this state must be purged either through timers or through least-recently-used eviction, depending on application requirements.

[RFC4787] recommends a 2 minute timeout interval for UDP, however, often timer are lower in the range of 15 to 30 second. In constrast [RFC5382] recommends a timeout of more than 2 hours for TCP, given TCP is a connection-oriented protocol with well defined closure semantics. For network devices that are QUIC-aware, it is recommended to also use longer timeouts for QUIC traffic, as QUIC is connection-oriented and as such a handshake packet from the server indicates the willingness of the server to communicate with the client.

The QUIC header optionally contains a Connection ID which can be used as additional entropy beyond the 5-tuple, if needed. The QUIC handshake needs to be observed in order to understand whether the Connection ID is present and what length it has. However, Connection IDs may be renegotiated during a connection, and this renegotiation is not visible to the path. Keying state off the Connection ID may therefore cause undetectable and unrecoverable loss of state in the middle of a connection. Use of Connection ID specifically discouraged for NAT applications.

4.2. Passive network performance measurement and troubleshooting

Limited RTT measurement is possible by passive observation of QUIC traffic; see Section 3.7. No passive measurement of loss is possible with the present wire image. Extremely limited observation of

upstream congestion may be possible via the observation of CE markings on ECN-enabled QUIC traffic.

4.3. Server cooperation with load balancers

In the case of content distribution networking architectures including load balancers, the connection ID provides a way for the server to signal information about the desired treatment of a flow to the load balancers. Guidance on assigning connection IDs is given in [QUIC-APPLICABILITY].

4.4. DDoS Detection and Mitigation

Current practices in detection and mitigation of Distributed Denial of Service (DDoS) attacks generally involves classification of incoming traffic (as packets, flows, or some other aggregate) into "good" (productive) and "bad" (DDoS) traffic, then differential treatment of this traffic to forward only good traffic, to the extent possible. This operation is often done in a separate specialized mitigation environment through which all traffic is filtered; a generalized architecture for separation of concerns in mitigation is given in [DOTS-ARCH].

Key to successful DDoS mitigation is efficient classification of this traffic in the mitigation environment. Limited first-packet garbage detection as in Section 3.1.2 and stateful tracking of QUIC traffic as in Section 4.1 above may be useful during classification.

Note that the use of a connection ID to support connection migration renders 5-tuple based filtering insufficient and requires more state to be maintained by DDoS defense systems. For the common case of NAT rebinding, DDoS defense systems can detect a change in client's endpoint address by linking flows based on the first 8 bytes of the server's connection IDs, provided the server is using at least 8-bytes-long connection IDs. QUIC's linkability resistance ensures that a deliberate connection migration is accompanied by a change in the connection ID and necessitate that connection ID aware DDoS defense system must have the same information about connection IDs as the load balancer [I-D.ietf-quic-load-balancers]. This may be complicated where mitigation and load balancing environments are logically separate.

It is questionable whether connection migrations must be supported during a DDoS attack. If the connection migration is not visible to the network that performs the DDoS detection, an active, migrated QUIC connection may be blocked by such a system under attack. As soon as the connection blocking is detected by the client, the client may rely on the fast resumption mechanism provided by QUIC.

When clients migrate to a new path, they should be prepared for the migration to fail and attempt to reconnect quickly.

4.5. UDP Policing

Today, UDP is the most prevalent DDoS vector, since it is easy for compromised non-admin applications to send a flood of large UDP packets (while with TCP the attacker gets throttled by the congestion controller) or to craft reflection and amplification attacks. Networks should therefore be prepared for UDP flood attacks on ports used for QUIC traffic. One possible response to this threat is to police UDP traffic on the network, allocating a fixed portion of the network capacity to UDP and blocking UDP datagram over that cap.

The recommended way to police QUIC packets is to either drop them all or to throttle them based on the hash of the UDP datagram's source and destination addresses, blocking a portion of the hash space that corresponds to the fraction of UDP traffic one wishes to drop. When the handshake is blocked, QUIC-capable applications may failover to TCP (at least applications using well-known UDP ports). However, blindly blocking a significant fraction of QUIC packets will allow many QUIC handshakes to complete, preventing a TCP failover, but the connections will suffer from severe packet loss.

4.6. Distinguishing acknowledgment traffic

Some deployed in-network functions distinguish pure-acknowledgment (ACK) packets from packets carrying upper-layer data in order to attempt to enhance performance, for example by queueing ACKs differently or manipulating ACK signaling. Distinguishing ACK packets is trivial in TCP, but not supported by QUIC, since acknowledgment signaling is carried inside QUIC's encrypted payload, and ACK manipulation is impossible. Specifically, heuristics attempting to distinguish ACK-only packets from payload-carrying packets based on packet size are likely to fail, and are emphatically NOT RECOMMENDED.

4.7. QoS support and ECMP

[EDITOR'S NOTE: this is a bit speculative; keep?]

QUIC does not provide any additional information on requirements on Quality of Service (QoS) provided from the network. QUIC assumes that all packets with the same 5-tuple {dest addr, source addr, protocol, dest port, source port} will receive similar network treatment. That means all stream that are multiplexed over the same QUIC connection require the same network treatment and are handled by the same congestion controller. If differential network treatment is desired, multiple QUIC connections to the same server might be

used, given that establishing a new connection using 0-RTT support is cheap and fast.

QoS mechanisms in the network MAY also use the connection ID for service differentiation, as a change of connection ID is bound to a change of address which anyway is likely to lead to a re-route on a different path with different network characteristics.

Given that QUIC is more tolerant of packet re-ordering than TCP (see Section 2.7), Equal-cost multi-path routing (ECMP) does not necessarily need to be flow based. However, 5-tuple (plus eventually connection ID if present) matching is still beneficial for QoS given all packets are handled by the same congestion controller.

5. IANA Considerations

This document has no actions for IANA.

6. Security Considerations

Supporting manageability of QUIC traffic inherently involves tradeoffs with the confidentiality of QUIC's control information; this entire document is therefore security-relevant.

7. Contributors

Dan Druta contributed text to Section 4.4. Igor Lubashev contributed text to Section 4.3 on the use of the connection ID for load balancing. Marcus Ilhar contributed text to Section 3.7 on the use of the spin bit. The pseudo provided in the appendix is based on input provided by David Schinazi.

8. Acknowledgments

Thanks to Martin Thomson and Martin Duke for contributing by reviewing and providing text proposals.

This work is partially supported by the European Commission under Horizon 2020 grant agreement no. 688421 Measurement and Architecture for a Middleboxed Internet (MAMI), and by the Swiss State Secretariat for Education, Research, and Innovation under contract no. 15.0268. This support does not imply endorsement.

9. Appendix

This appendix uses the following conventions: array[i] - one byte at index i of array array[i:j] - subset of array starting with index i (inclusive) up to j-1 (inclusive) array[i:] - subset of array starting with index i (inclusive) up to the end of the array

9.1. Distinguishing IETF QUIC and Google QUIC Versions

This section contains algorithms that allows parsing versions from both Google QUIC and IETF QUIC. These mechanisms will become irrelevant when IETF QUIC is fully deployed and Google QUIC is deprecated.

Note that other than this appendix, nothing in this document applies to Google QUIC. And the purpose of this appendix is merely to distinguish IETF QUIC from any versions of Google QUIC.

Conceptually, a Google QUIC version is an opaque 32bit field. When we refer to a version with four printable characters, we use its ASCII representation: for example, Q050 refers to {'Q', '0', '5', '0'} which is equal to {0x51, 0x30, 0x35, 0x30}. Otherwise, we use its hexadecimal representation: for example, 0xff00001d refers to {0xff, 0x00, 0x00, 0x1d}.

QUIC versions that start with 'Q' or 'T' followed by three digits are Google QUIC versions. Versions up to and including 43 are documented by <https://docs.google.com/document/d/1WJvyZflAO2pq77yOLbp9NsGjC1CHetAXV8I0fQe-B_U/preview>. Versions Q046, Q050, T050, and T051 are not fully documented, but this appendix should contain enough information to allow parsing Client Hellos for those versions.

To extract the version number itself, one needs to look at the first byte of the QUIC packet, in other words the first byte of the UDP payload.

```
first_byte = packet[0]
first_byte_bit1 = ((first_byte & 0x80) != 0)
first_byte_bit2 = ((first_byte & 0x40) != 0)
first_byte_bit3 = ((first_byte & 0x20) != 0)
first_byte_bit4 = ((first_byte & 0x10) != 0)
first_byte_bit5 = ((first_byte & 0x08) != 0)
first_byte_bit6 = ((first_byte & 0x04) != 0)
first_byte_bit7 = ((first_byte & 0x02) != 0)
first_byte_bit8 = ((first_byte & 0x01) != 0)
if (first_byte_bit1) {
  version = packet[1:5]
} else if (first_byte_bit5 && !first_byte_bit2) {
  if (!first_byte_bit8) {
    abort("Packet without version")
  }
  if (first_byte_bit5) {
    version = packet[9:13]
  } else {
    version = packet[5:9]
  }
} else {
  abort("Packet without version")
}
```

9.2. Extracting the CRYPTO frame

```
counter = 0
while (payload[counter] == 0) {
  counter += 1
}
first_nonzero_payload_byte = payload[counter]
fnz_payload_byte_bit3 = ((first_nonzero_payload_byte & 0x20) != 0)

if (first_nonzero_payload_byte != 0x06) {
  abort("Unexpected frame")
}
if (payload[counter+1] != 0x00) {
  abort("Unexpected crypto stream offset")
}
counter += 2
if ((payload[counter] & 0xc0) == 0) {
  crypto_data_length = payload[counter]
  counter += 1
} else {
  crypto_data_length = payload[counter:counter+2]
  counter += 2
}
crypto_data = payload[counter:counter+crypto_data_length]
ParseTLS(crypto_data)
```

10. References

10.1. Normative References

[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>.

10.2. Informative References

[Ding2015] Ding, H. and M. Rabinovich, "TCP Stretch Acknowledgments and Timestamps - Findings and Impliciations for Passive RTT Measurement (ACM Computer Communication Review)", July 2015, <http://www.sigcomm.org/sites/default/files/ccr/papers/2015/July/0000000-0000002.pdf>.

[DOTS-ARCH] Mortensen, A., Reddy.K, T., Andreasen, F., Teague, N., and R. Compton, "Distributed-Denial-of-Service Open Threat Signaling (DOTS) Architecture", Work in Progress, Internet-Draft, draft-ietf-dots-architecture-18, 6 March 2020, <http://www.ietf.org/internet-drafts/draft-ietf-dots-architecture-18.txt>.

[I-D.ietf-quic-load-balancers]
Duke, M. and N. Banks, "QUIC-LB: Generating Routable QUIC Connection IDs", Work in Progress, Internet-Draft, draft-ietf-quic-load-balancers-05, 30 October 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-load-balancers-05.txt>.

[IPIM] Allman, M., Beverly, R., and B. Trammell, "In-Protocol Internet Measurement (arXiv preprint 1612.02902)", 9 December 2016, <https://arxiv.org/abs/1612.02902>.

[QUIC-APPLICABILITY]
Kuehlewind, M. and B. Trammell, "Applicability of the QUIC Transport Protocol", Work in Progress, Internet-Draft, draft-ietf-quic-applicability-07, 8 July 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-applicability-07.txt>.

[QUIC-HTTP] Bishop, M., "Hypertext Transfer Protocol Version 3 (HTTP/3)", Work in Progress, Internet-Draft, draft-ietf-quic-http-32, 20 October 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-http-32.txt>.

[QUIC-INVARIANTS] Thomson, M., "Version-Independent Properties of QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-invariants-11, 24 September 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-invariants-11.txt>.

[QUIC-TLS] Thomson, M. and S. Turner, "Using TLS to Secure QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-tls-32, 20 October 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-tls-32.txt>.

[QUIC-TRANSPORT] Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", Work in Progress, Internet-Draft, draft-ietf-quic-transport-32, 20 October 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-32.txt>.

[QUIC_LB] Duke, M. and N. Banks, "QUIC-LB: Generating Routable QUIC Connection IDs", Work in Progress, Internet-Draft, draft-ietf-quic-load-balancers-05, 30 October 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-load-balancers-05.txt>.

[RFC4787] Audet, F., Ed. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, DOI 10.17487/RFC4787, January 2007, <https://www.rfc-editor.org/info/rfc4787>.

[RFC5382] Guha, S., Ed., Biswas, K., Ford, B., Sivakumar, S., and P. Srisuresh, "NAT Behavioral Requirements for TCP", BCP 142, RFC 5382, DOI 10.17487/RFC5382, October 2008, <https://www.rfc-editor.org/info/rfc5382>.

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <https://www.rfc-editor.org/info/rfc6066>.

[RFC6928] Chu, J., Dukkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, DOI 10.17487/RFC6928, April 2013, <https://www.rfc-editor.org/info/rfc6928>.

[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/info/rfc7301>.

[RFC7605] Touch, J., "Recommendations on Using Assigned Transport Port Numbers", BCP 165, RFC 7605, DOI 10.17487/RFC7605, August 2015, <https://www.rfc-editor.org/info/rfc7605>.

[TLS-ESNI] Rescorla, E., Oku, K., Sullivan, N., and C. Wood, "TLS Encrypted Client Hello", Work in Progress, Internet-Draft, draft-ietf-tls-esni-08, 16 October 2020, <http://www.ietf.org/internet-drafts/draft-ietf-tls-esni-08.txt>.

[TMA-QOF] Trammell, B., Gugelmann, D., and N. Brownlee, "Inline Data Integrity Signals for Passive Measurement (in Proc. TMA 2014)", April 2014.

[WIRE-IMAGE] Trammell, B. and M. Kuehlewind, "The Wire Image of a Network Protocol", RFC 8546, DOI 10.17487/RFC8546, April 2019, <https://www.rfc-editor.org/info/rfc8546>.

Authors' Addresses

Mirja Kuehlewind
Ericsson

Email: mirja.kuehlewind@ericsson.com

Brian Trammell
Google
Gustav-Gull-Platz 1
CH- 8004 Zurich
Switzerland

Email: ietf@trammell.ch