Workgroup: Network Working Group
Internet-Draft:
draft-ietf-quic-manageability-17
Published: 11 July 2022
Intended Status: Informational
Expires: 12 January 2023
Authors: M. Kuehlewind
Ericsson
B. Trammell
Google Switzerland GmbH

Manageability of the QUIC Transport Protocol

Abstract

This document discusses manageability of the QUIC transport protocol, focusing on the implications of QUIC's design and wire image on network operations involving QUIC traffic. Its intended audience is network operators and equipment vendors who rely on the use of transport-aware network functions.

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 12 January 2023.

Table of Contents

1. Introduction

QUIC [QUIC-TRANSPORT] is a new transport protocol that is encapsulated in UDP. QUIC integrates TLS [QUIC-TLS] to encrypt all payload data and most control information. QUIC version 1 was designed primarily as a transport for HTTP, with the resulting protocol being known as HTTP/3 [QUIC-HTTP].

This document provides guidance for network operations that manage QUIC traffic. This includes guidance on how to interpret and utilize information that is exposed by QUIC to the network, requirements and assumptions of the QUIC design with respect to network treatment, and a description of how common network management practices will be impacted by QUIC.

QUIC is an end-to-end transport protocol. No information in the protocol header, even that which can be inspected, is meant to be mutable by the network. This is achieved through integrity protection of the wire image [WIRE-IMAGE]. Encryption of most control signaling means that less information is visible to the network than is the case with TCP.

Integrity protection can also simplify troubleshooting, because none of the nodes on the network path can modify transport layer information. However, it does imply that in-network operations that depend on modification of data are not possible without the cooperation of an QUIC endpoint. This might be possible with the introduction of a proxy which authenticates as an endpoint. Proxy operations are not in scope for this document.

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.

2. Features of the QUIC Wire Image

In this section, we discuss 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, the field that identifies

the QUIC version in some packets, and the format of the Version Negotiation Packet, are both inspectable and invariant [QUIC-INVARIANTS].

This document describes version 1 of the QUIC protocol, whose wire image is fully defined in [QUIC-TRANSPORT] and [QUIC-TLS]. Features of the wire image described herein may change in future versions of the protocol, except when specified as an invariant [QUIC-INVARIANTS], and cannot be used to identify QUIC as a protocol or to infer the behavior of future versions of QUIC.

Appendix A provides non-normative guidance on the identification of QUIC version 1 packets compared to some pre-standard versions.

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 is the Header Form bit, and indicates which type of header is present. The purpose of this bit is invariant across QUIC versions.

The long header exposes more information. In version 1 of QUIC, 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 contain only an optional destination connection ID and the spin bit for RTT measurement. In version 1 of QUIC, they are used after connection establishment.

The following information is exposed in QUIC packet headers in all versions of QUIC:

- version number: the version number is present in the long header, and identifies the version used for that packet. During Version Negotiation (see Section 17.2.1 of [QUIC-TRANSPORT] and Section 2.8), the version number field has a special value (0x00000000) that identifies the packet as a Version Negotiation packet. QUIC version 1 uses version 0x00000001. Operators should expect to observe packets with other version numbers as a result of various Internet experiments, future standards, and greasing. All deployed versions are maintained in an IANA registry (see Section 22.2 of [QUIC-TRANSPORT]).

- 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 4.4 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.

In version 1 of QUIC, the following additional information is exposed:

* "fixed bit": The second-most-significant bit of the first octet of most QUIC packets of the current version is set to 1, enabling endpoints to demultiplex with other UDP-encapsulated protocols. Even though this bit is fixed in the version 1 specification, endpoints might use an extension that varies the bit. Therefore, observers cannot reliably use it as an identifier for QUIC.

* latency spin bit: The third-most-significant bit of the first octet in the short packet header for version 1. The spin bit is set by endpoints such that tracking edge transitions can be used to passively observe end-to-end RTT. See Section 3.8.2 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.

* 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, version 1 of QUIC cryptographically obfuscates other information in the packet headers:

* 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 long headers, while it is implicit (depending on destination connection ID length) in short headers. 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 to support 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; see Section 12.2 of [QUIC-TRANSPORT]. 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 17.2 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 for all other IETF transports [RFC7605], there is no guarantee that a specific application will use a given registered port, or that a given port carries traffic belonging to the respective registered service, especially when application layer information is encrypted. For example, [QUIC-HTTP] specifies the use of Alt-Svc for discovery of HTTP/3 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 zero-length, 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.

The QUIC handshake can normally be recognized on the wire through at least four datagrams we'll call "Client Initial", "Server Initial",

and "Client Completion", and "Server 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.

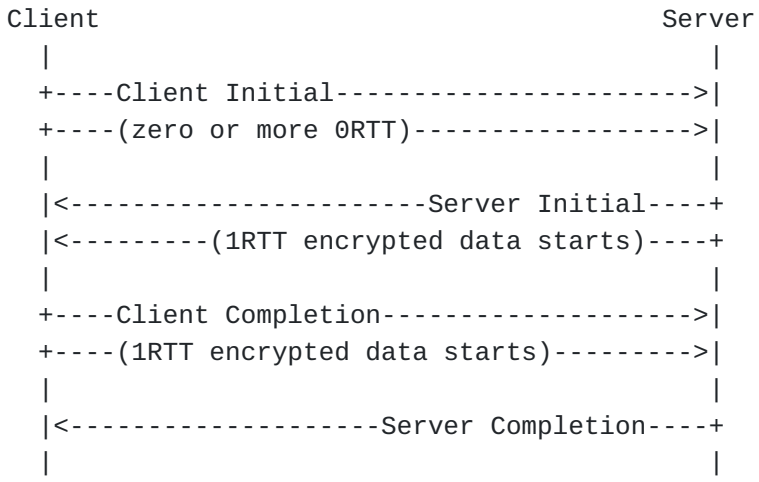

Figure 1: General communication pattern visible in the QUIC handshake

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

The Client Completion datagram contains at least one Handshake packet and some also include an Initial packet.

Datagrams that contain a Client Initial Packet (Client Initial, Server Initial, and some Client Completion) contain at least 1200 octets of UDP payload. This protects against amplification attacks and verifies that the network path meets the requirements for the minimum QUIC IP packet size; see Section 14 of [QUIC-TRANSPORT]. This is accomplished by either adding PADDING frames within the Initial packet, coalescing other packets with the Initial packet, or leaving unused payload in the UDP packet after the Initial packet. A

network path needs to be able to forward at least this size of packet for QUIC to be used.

The content of Client 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. They are therefore considered visible in this 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 Client Completion Figure 4 and the Server Completion Figure 5 datagrams finish these first two contexts, by sending the final acknowledgment and finishing the transmission of CRYPTO frames.

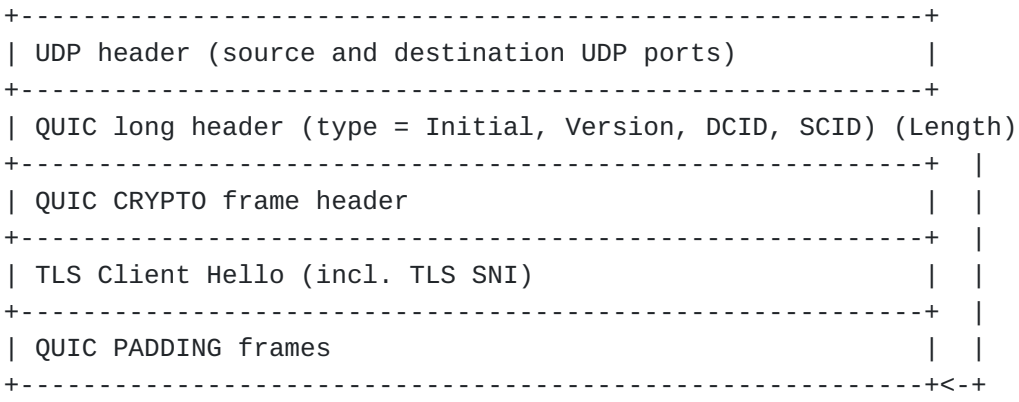

Figure 2: Typical Client Initial datagram pattern without 0-RTT

The Client Initial datagram exposes version number, source and destination connection IDs without encryption. Information in the TLS Client Hello frame, including any TLS Server Name Indication (SNI) present, is obfuscated using the Initial secret. Note that the location of PADDING is implementation-dependent, and PADDING frames might not appear in a coalesced Initial packet.

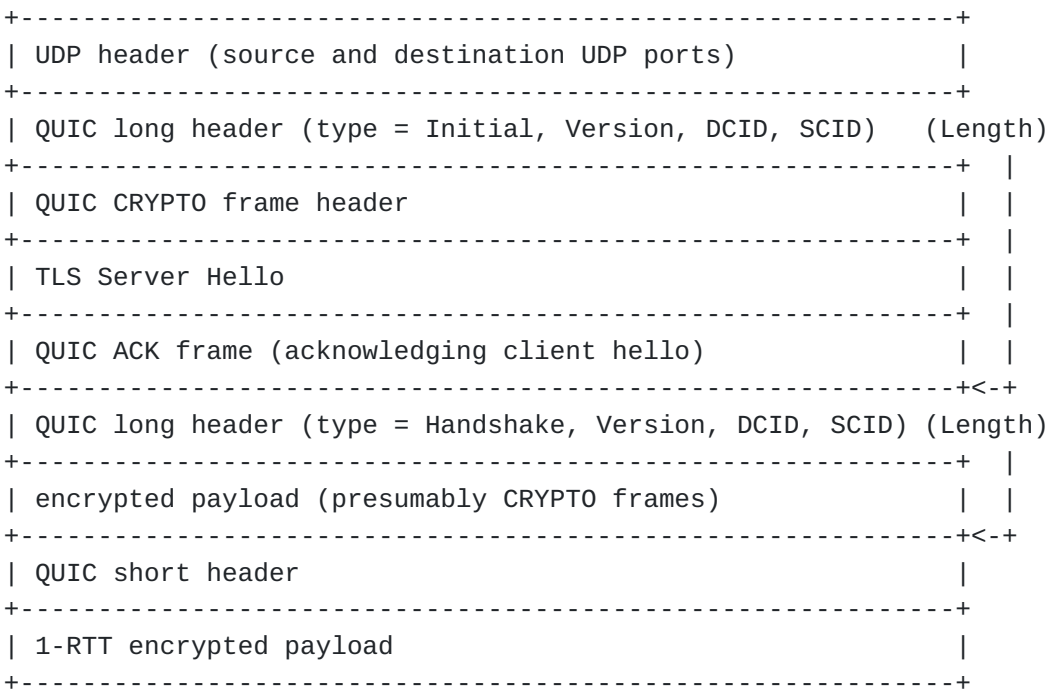

Figure 3: Typical Server Initial datagram pattern

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

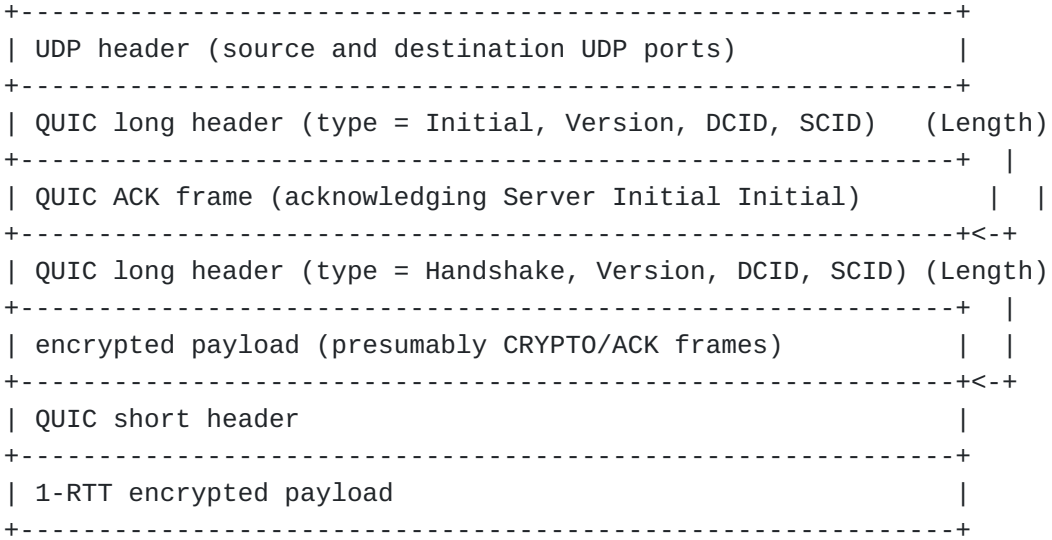

Figure 4: Typical Client Completion datagram pattern

The Client 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 3.8.

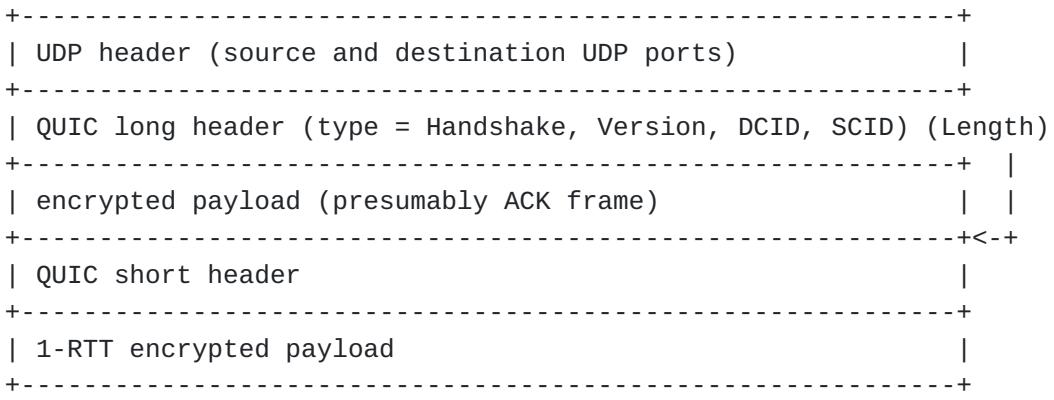

Figure 5: Typical Server Completion datagram pattern

Similar to Client Completion, Server 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 Client Initial datagram, as shown in Figure 6.

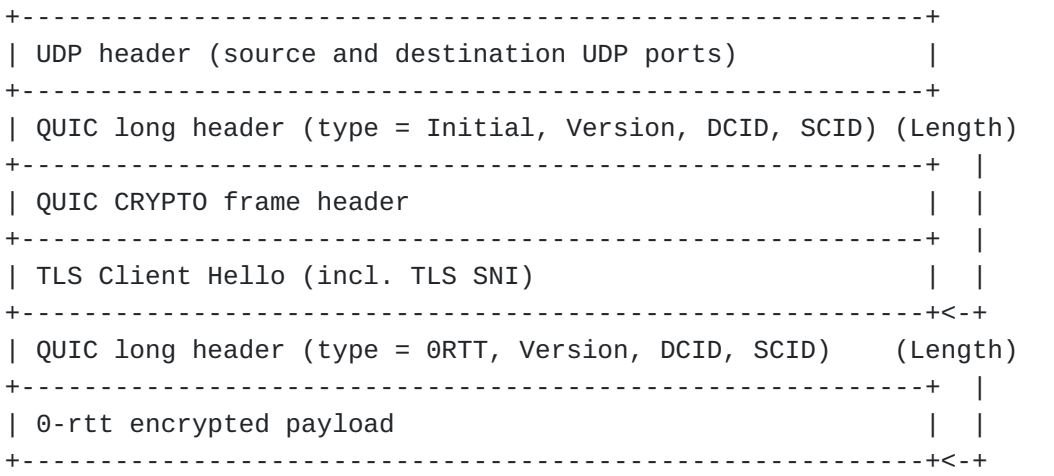

Figure 6: Typical 0-RTT Client Initial datagram pattern

In a 0-RTT Client Initial 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 Initial datagram, containing the rest of the 0-RTT data. The amount of 0-RTT protected data that can be sent in the first round is limited by the initial congestion window, typically around 10 packets (see Section 7.2 of [QUIC-RECOVERY]).

2.5. Integrity Protection of the Wire Image

As soon as the cryptographic context is established, all information in the QUIC header, including exposed information, is integrity-protected. Further, information that was exposed in packets sent before the cryptographic context was established is validated during the cryptographic handshake. Therefore, devices on path cannot alter any information or bits in QUIC packets. Such alterations would cause the integrity check to fail, which results in the receiver discarding the packet. Some parts of Initial packets could be altered by removing and re-applying the authenticated encryption without immediate discard at the receiver. However, the cryptographic handshake validates most fields and any modifications in those fields will result in connection establishment failing later on.

2.6. Connection ID and Rebinding

The connection ID in the QUIC packet headers allows association of QUIC packets using information independent of the five-tuple. This allows rebinding of a connection after one of one endpoint experienced an address change - usually the client. Further it can be used by in-network devices to ensure that related 5-tuple flows are appropriately balanced together.

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 supplied via encrypted frames (see Section 5.1 of [QUIC-TRANSPORT]). Therefore, observing a new connection ID does not necessary indicate a new connection.

[QUIC_LB] specifies algorithms for encoding the server mapping in a connection ID in order to share this information with selected on-path devices such as load balancers. Server mappings should only be exposed to selected entities. Uncontrolled exposure would allow linkage of multiple IP addresses to the same host if the server also supports migration which opens an attack vector on specific servers or pools. The best way to obscure an encoding is to appear random to any other observers, which is most rigorously achieved with encryption. As a result any attempt to infer information from specific parts of a connection ID is unlikely to be useful.

2.7. Packet Numbers

The packet number field is always present in the QUIC packet header in version 1; 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 packets are used by the server to indicate that a requested version from the client is not supported (see Section 6 of [QUIC-TRANSPORT]. Version Negotiation packets are not intrinsically protected, but future QUIC versions will use later encrypted messages to verify that they were authentic. Therefore any modification of this list will be detected and may cause the endpoints to terminate the connection attempt.

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 IDs of the Client Initial packet are reflected to provide a proof of return-routability. Therefore, changing this 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 near future, and is therefore not recommended. In addition, due to the speed of evolution of the protocol, devices that attempt to distinguish QUIC traffic from non-QUIC traffic for purposes of network admission control should admit all QUIC traffic regardless of version.

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/3 uses UDP port 443 by default, although URLs referring to resources available over HTTP/3 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 Section 5 of [RFC7605].

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 and Section 17 of [QUIC-TRANSPORT]), this method of recognizing QUIC traffic is not reliable. First, it only provides one bit of information and is prone to collision with UDP-based protocols other than those considered in [RFC7983]. 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 during the handshake via the use of an extension.

Even though transport parameters transmitted in the client's Initial packet are observable by the network, they cannot be modified by the network without risking connection failure. Further, the reply from the server cannot be observed, so observers on the network cannot know which parameters are actually in use.

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: for QUIC version 1, if the version number in the Initial packet from a client is the same as the version number in the Initial packet of the server response, that version has been accepted by both endpoints to be used for the rest of the connection.

The negotiated version cannot be identified for flows for which a handshake is not observed, such as in the case of connection migration; however, it might be possible to associate a flow with a flow for which a version has been identified; see Section 3.5.

3.1.2. First Packet Identification for Garbage Rejection

A related question is whether the first packet of a given flow on a port known to be associated with QUIC is a valid QUIC packet. This determination supports in-network filtering of garbage UDP packets (reflection attacks, random backscatter, etc.). 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 bits in the first byte may have different meanings in future versions of the protocol.

3.2. Connection Confirmation

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

Connection establishment uses Initial and Handshake packets containing a TLS handshake, and Retry packets that do not contain parts of the handshake. Connection establishment can therefore be detected using heuristics similar to those used to detect TLS over TCP. A client initiating a connection may also send data in 0-RTT packets directly after the Initial packet containing the TLS Client Hello. Since these packets may be reordered in the network, 0-RTT packets could be seen before the Initial packet.

Note that in this version of QUIC, clients send Initial packets before servers do, servers send Handshake packets before clients do, and only clients send Initial packets with tokens. Therefore, an endpoint can be identified as a client or server by an on-path observer. An attempted connection after Retry can be detected by correlating the contents of the Retry packet with the Token and the Destination Connection ID fields of the new Initial packet.

3.3. 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 not recommended to use as a way to construe internals of QUIC's operation as those mechanisms can change, e.g., due to the use of extensions.

3.4. Server Name Indication (SNI)

The client's TLS ClientHello may contain a Server Name Indication (SNI) [RFC6066] extension, 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 an Application-Layer Protocol Negotiation (ALPN) [RFC7301] extension,

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 contents of the ClientHello in TLS 1.3 [TLS-ECH]. This would make SNI-based application identification impossible by on-path observation for QUIC and other protocols that use TLS.

3.4.1. Extracting Server Name Indication (SNI) Information

If the ClientHello is not encrypted, it can be derived from the client's Initial packet by calculating the Initial secret to decrypt the packet payload and parsing the QUIC CRYPTO Frame containing the TLS ClientHello.

As both the derivation of the Initial secret and the structure of the Initial packet itself are version-specific, the first step is always to parse the version number (second to sixth bytes of the long header). Note that only long header packets carry the version number, so it is necessary to also check if the 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 packet to 1. To parse these versions, example code is provided in the appendix (see Appendix A). 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 bits of the header are both set to 0. Then the Destination Connection ID needs to be extracted to calculate the Initial secret using the version-specific Initial salt, as described in Section 5.2 of [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, the packet number length, the source connection ID length, and 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], but is obfuscated as described in Section 5.4 of [QUIC-TLS]. The source connection ID length is specified in the byte after the destination connection ID. The token length, which follows the source connection ID, is a variable-length integer as specified in Section 16 of [QUIC-TRANSPORT].

After decryption, the client's Initial packet can be parsed to detect the CRYPTO frame that contains the TLS ClientHello, which

then can be parsed similarly to TLS over TCP connections. The client's Initial packet may contain other frames, so the first bytes of each frame need to be checked to identify the frame type, and if needed skip over it. Note that the length of the frames is dependent on the frame type. In QUIC version 1, the packet is expected to contain only CRYPTO frames and optionally PADDING frames. PADDING frames, each consisting of a single zero byte, may occur before, after, or between CRYPTO frames. There might be multiple CRYPTO frames. Finally, an extension might define additional frame types which could be present.

Note that subsequent Initial packets might contain a Destination Connection ID other than the one used to generate the Initial secret. Therefore, attempts to decrypt these packets using the procedure above might fail unless the Initial secret is retained by the observer.

3.5. Flow Association

The QUIC connection ID (see Section 2.6) is designed to allow a coordinating on-path device, such as a load-balancer, to associate two flows when one of the endpoints changes address or port. This change can be due to NAT rebinding or address migration.

The connection ID must change upon intentional address change by an endpoint, and connection ID negotiation is encrypted, so it is not possible for a passive observer to link intended changes of address using the connection ID.

When one endpoint unintentionally changes its address, as is the case with NAT rebinding, an on-path observer may be able to use the connection ID to associate the flow on the new address with the flow on the old address.

A network function that attempts to use the connection ID to associate flows must be robust to the failure of this technique. Since the connection ID may change multiple times during the lifetime of a connection, packets with the same five-tuple but different connection IDs might or might not belong to the same connection. Likewise, packets with the same connection ID but different five-tuples might not belong to the same connection, either.

Connection IDs should be treated as opaque; see Section 4.4 for caveats regarding connection ID selection at servers.

3.6. 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 Section 4.2.

3.7. 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.

3.8. Round-Trip Time (RTT) Measurement

The 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 Section 17.3.1 of [QUIC-TRANSPORT].

3.8.1. Measuring Initial RTT

In the common case, the delay between the client's Initial packet (containing the TLS ClientHello) and the server's Initial packet (containing the TLS ServerHello) represents the RTT component on the path between the observer and the server. The delay between the server's first Handshake packet and the Handshake packet sent by the client represents the RTT component on the path between the observer and the client. While the client may send 0-RTT packets after the Initial packet during 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.

3.8.2. Using the Spin Bit for Passive RTT Measurement

The spin bit provides a version-specific method to measure per-flow RTT from observation points on the network path throughout the duration of a connection. See Section 17.4 of [QUIC-TRANSPORT] for the definition of the spin bit in Version 1 of QUIC. Endpoint participation in spin bit signaling is optional. 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, 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. This mechanism follows the principles of protocol measurability laid out in [IPIM].

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.

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.

Raw RTT samples generated using these techniques can be processed in various ways to generate useful network performance metrics. A simple linear smoothing or moving minimum filter can be applied to the stream of RTT samples to get a more stable estimate of application-experienced RTT. RTT samples measured from the spin bit can also be used to generate RTT distribution information, including minimum RTT (which approximates network RTT over longer time windows) and RTT variance (which approximates jitter as seen by the application).

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. Passive Network Performance Measurement and Troubleshooting

Limited RTT measurement is possible by passive observation of QUIC traffic; see Section 3.8. 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.

4.2. 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 3.6) means that this state must be purged either through timers or through least-recently-used eviction, depending on application requirements.

While QUIC has no clear network-visible end-of-connection signal and therefore does require timer-based state removal, the QUIC handshake

indicates confirmation by both ends of a valid bidirectional transmission. As soon as the handshake completed, timers should be set long enough to also allow for short idle time during a valid transmission.

[RFC4787] requires a timeout that is not less than 2 minutes for most UDP traffic. However, in practice, timers are sometimes lower, in the range of 30 to 60 seconds. In contrast, [RFC5382] recommends a timeout of more than 2 hours for TCP, given that TCP is a connection-oriented protocol with well- defined closure semantics.

Even though QUIC has explicitly been designed tolerate NAT rebindings, decreasing the NAT timeout is not recommended, as it may negatively impact application performance or incentivize endpoints to send very frequent keep-alive packets. Instead it is recommended, even when lower timers are used for other UDP traffic, to use a timer of at least two minutes for QUIC traffic.

If state is removed too early, this could lead to black-holing of incoming packets after a short idle period. To detect this situation, a timer at the client needs to expire before a re-establishment can happen (if at all), which would lead to unnecessary long delays in an otherwise working connection.

Furthermore, not all endpoints use routing architectures where connections will survive a port or address change. So even when the client revives the connection, a NAT rebinding can cause a routing mismatch where a packet is not even delivered to the server that might support address migration. For these reasons, the limits in [RFC4787] are important to avoid black-holing of packets (and hence avoid interrupting the flow of data to the client), especially where devices are able to distinguish QUIC traffic from other UDP payloads.

The QUIC header optionally contains a connection ID which could provide additional entropy beyond the 5-tuple. 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 after the handshake, and this renegotiation is not visible to the path. Therefore using the connection ID as a flow key field for stateful treatment of flows is not recommended as connection ID changes will cause undetectable and unrecoverable loss of state in the middle of a connection. Specially, the use of the connection ID for functions that require state to make a forwarding decison is not viable as it will break connectivity or at minimum cause long timeout-based delays before this problem is detected by the endpoints and the connection can potentially be re-established.

Use of connection IDs is specifically discouraged for NAT applications. If a NAT hits an operational limit, it is recommended to rather drop the initial packets of a flow (see also Section 4.5), which potentially triggers a fallback to TCP. Use of the connection ID to multiplex multiple connections on the same IP address/port pair is not a viable solution as it risks connectivity breakage, in case the connection ID changes.

4.3. Address Rewriting to Ensure Routing Stability

While QUIC's migration capability makes it possible for an server to survive address changes, this does not work if the routers or switches in the server infrastructure route using the address-port 4-tuple. If infrastructure routes on addresses only, NAT rebinding or address migration will cause packets to be delivered to the wrong server. [QUIC_LB] describes a way to addresses this problem by coordinating the selection and use of connection IDs between load-balancers and servers.

Applying address translation at a middlebox to maintain a stable address-port mapping for flows based on connection ID might seem like a solution to this problem. However, hiding information about the change of the IP address or port conceals important and security-relevant information from QUIC endpoints and as such would facilitate amplification attacks (see Section 9 of [QUIC-TRANSPORT]). A NAT function that hides peer address changes prevents the other end from detecting and mitigating attacks as the endpoint cannot verify connectivity to the new address using QUIC PATH_CHALLENGE and PATH_RESPONSE frames.

In addition, a change of IP address or port is also an input signal to other internal mechanisms in QUIC. When a path change is detected, path-dependent variables like congestion control parameters will be reset protecting the new path from overload.

Therefore, the use of address rewriting to ensure routing stability can open QUIC up to various attacks, as it conceals client address changes, and as such masks important signals that drive security mechanisms.

4.4. Server Cooperation with Load Balancers

In the case of networking architectures that include load balancers, the connection ID can be used as 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]. [QUIC_LB] describes a system for coordinating selection and use of connection IDs between load-balancers and servers.

4.5. Filtering Behavior

[RFC4787] describes possible packet filtering behaviors that relate to NATs but is often also used is other scenarios where packet filtering is desired. Though the guidance there holds, a particularly unwise behavior is to admit a handful of UDP packets and then make a decision as to whether or not to filter it. QUIC applications are encouraged to fail over to TCP if early packets do not arrive at their destination [I-D.ietf-quic-applicability], as QUIC is based on UDP and there are known blocks of UDP traffic (see Section 4.6). Admitting a few packets allows the QUIC endpoint to determine that the path accepts QUIC. Sudden drops afterwards will result in slow and costly timeouts before abandoning the connection.

4.6. UDP Blocking or Throttling

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. Some networks therefore block UDP traffic. With increased deployment of QUIC, there is also an increased need to allow UDP traffic on ports used for QUIC. However, if UDP is generally enabled on these ports, UDP flood attacks may also use the same ports. One possible response to this threat is to throttle UDP traffic on the network, allocating a fixed portion of the network capacity to UDP and blocking UDP datagrams over that cap. As the portion of QUIC traffic compared to TCP is also expected to increase over time, using such a limit is not recommended but if done, limits might need to be adapted dynamically.

Further, if UDP traffic is desired to be throttled, it is recommended to block individual QUIC flows entirely rather than dropping packets randomly. When the handshake is blocked, QUIC-capable applications may failover to TCP However, blocking a random fraction of QUIC packets across 4-tuples will allow many QUIC handshakes to complete, preventing a TCP failover, but the connections will suffer from severe packet loss (see also Section 4.5). Therefore UDP throttling should be realized by per-flow policing as opposed to per-packet policing. Note that this per-flow policing should be stateless to avoid problems with stateful treatment of QUIC flows (see Section 4.2), for example blocking a portion of the space of values of a hash function over the addresses and ports in the UDP datagram. While QUIC endpoints are often able to survive address changes, e.g. by NAT rebindings, blocking a portion of the traffic based on 5-tuple hashing increases the risk of black-holing an active connection when the address changes.

4.7. DDoS Detection and Mitigation

On-path observation of the transport headers of packets can be used for various security functions. For example, Denial of Service (DOS) and Distributed DOS (DDOS) attacks against the infrastructure or against an endpoint can be detected and mitigated by characterising anomalous traffic. Other uses include support for security audits (e.g., verifying the compliance with ciphersuites); client and application fingerprinting for inventory; and to provide alerts for network intrusion detection and other next generation firewall functions.

Current practices in detection and mitigation of DDoS attacks generally involve classification of incoming traffic (as packets, flows, or some other aggregate) into "good" (productive) and "bad" (DDoS) traffic, and then differential treatment of this traffic to forward only good traffic. 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].

Efficient classification of this DDoS traffic in the mitigation environment is key to the success of this approach. Limited first-packet garbage detection as in Section 3.1.2 and stateful tracking of QUIC traffic as in Section 4.2 above may be useful during classification.

Note that the use of a connection ID to support connection migration renders 5-tuple based filtering insufficient to detect active flows and requires more state to be maintained by DDoS defense systems if support of migration of QUIC flows is desired. For the common case of NAT rebinding, where the client's address changes without the client's intent or knowledge, DDoS defense systems can detect a change in the client's endpoint address by linking flows based on the server's connection IDs. However, QUIC's linkability resistance ensures that a deliberate connection migration is accompanied by a change in the connection ID. In this case, the connection ID can not be used to distinguish valid, active traffic from new attack traffic.

It is also possible for endpoints to directly support security functions such as DoS classification and mitigation. Endpoints can cooperate with an in-network device directly by e.g. sharing information about connection IDs.

Another potential method could use an on-path network device that relies on pattern inferences in the traffic and heuristics or machine learning instead of processing observed header information.

However, it is questionable whether connection migrations must be supported during a DDoS attack. While unintended migration without a connection ID change can be more easily supported, it might be acceptable to not support migrations of active QUIC connections that are not visible to the network functions performing the DDoS detection. As soon as the connection blocking is detected by the client, the client may be able to 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.

Beyond in-network DDoS protection mechanisms, TCP syncookies [RFC4937] are a well-established method of mitigating some kinds of TCP DDoS attacks. QUIC Retry packets are the functional analogue to syncookies, forcing clients to prove possession of their IP address before committing server state. However, there are safeguards in QUIC against unsolicited injection of these packets by intermediaries who do not have consent of the end server. See [QUIC_LB] for standard ways for intermediaries to send Retry packets on behalf of consenting servers.

4.8. Quality of Service handling and ECMP

It is expected that any QoS handling in the network, e.g. based on use of DiffServ Code Points (DSCPs) [RFC2475] as well as Equal-Cost Multi-Path (ECMP) routing, is applied on a per flow-basis (and not per-packet) and as such that all packets belonging to the same QUIC connection get uniform treatment. Using ECMP to distribute packets from a single flow across multiple network paths or any other non-uniform treatment of packets belong to the same connection could result in variations in order, delivery rate, and drop rate. As feedback about loss or delay of each packet is used as input to the congestion controller, these variations could adversely affect performance.

Depending of the loss recovery mechanism implemented, QUIC may be more tolerant of packet re-ordering than traditional TCP traffic (see Section 2.7). However, it cannot be known by the network which exact recovery mechanism is used and therefore reordering tolerance should be considered as unknown.

4.9. Handling ICMP Messages

Datagram Packetization Layer PMTU Discovery (PLPMTUD) can be used by QUIC to probe for the supported PMTU. PLPMTUD optionally uses ICMP messages (e.g., IPv6 Packet Too Big messages). Given known attacks with the use of ICMP messages, the use of PLPMTUD in QUIC has been designed to safely use but not rely on receiving ICMP feedback (see Section 14.2.1. of [QUIC-TRANSPORT]).

Networks are recommended to forward these ICMP messages and retain as much of the original packet as possible without exceeding the minimum MTU for the IP version when generating ICMP messages as recommended in [RFC1812] and [RFC4443].

4.10. Guiding Path MTU

Some networks support 1500-byte packets, but can only do so by fragmenting at a lower layer before traversing a smaller MTU segment, and then reassembling. This is permissible even when the IP layer is IPv6 or IPv4 with the DF bit set, because it occurs below the IP layer. However, this process can add to compute and memory costs, leading to a bottleneck that limits network capacity. In such networks this generates a desire to influence a majority of senders to use smaller packets, so that the limited reassembly capacity is not exceeded.

For TCP, MSS clamping (Section 3.2 of [RFC4459]) is often used to change the sender's maximum TCP segment size, but QUIC requires a different approach. Section 14 of [QUIC-TRANSPORT] advises senders to probe larger sizes using Datagram Packetization Layer PMTU Discovery ([DPLPMTUD]) or Path Maximum Transmission Unit Discovery (PMTUD: [RFC1191] and [RFC8201]). This mechanism will encourage senders to approach the maximum size, which could cause fragmentation with a network segment that they may not be aware of.

If path performance is limited when sending larger packets, an on-path device should support a maximum packet size for a specific transport flow and then consistently drop all packets that exceed the configured size when the inner IPv4 packet has DF set, or IPv6 is used. Endpoints can cache PMTU information between IP flows, in the IP-layer cache, so short-term consistency between the PMTU for flows can help avoid an endpoint using a PMTU that is inefficient.

Networks with configurations that would lead to fragmentation of large packets should drop such packets rather than fragmenting them. Network operators who plan to implement a more selective policy may start by focussing on QUIC. QUIC flows cannot always be easily distinguished from other UDP traffic, but we assume at least some portion of QUIC traffic can be identified (see Section 3.1). For QUIC endpoints using DPLPMTUD it is recommended for the path to drop a packet larger than the supported size. A QUIC probe packet is used to discover the PMTU. If lost, this does not impact the flow of QUIC data.

IPv4 routers generate an ICMP message when a packet is dropped because the link MTU was exceeded. [RFC8504] specifies how an IPv6 node generates an ICMPv6 Packet Too Big message (PTB) in this case. PMTUD relies upon an endpoint receiving such PTB messages [RFC8201],

whereas DPLPMTUD does not reply upon these messages, but still can optionally use these to improve performance Section 4.6 of [DPLPMTUD].

Since a network cannot know in advance which discovery method a QUIC endpoint is using, it should always send a PTB message in addition to dropping the oversized packet. A generated PTB message should be compliant with the validation requirements of Section 14.2.1 of [QUIC-TRANSPORT], otherwise it will be ignored by DPLPMTUD. This will likely provide the right signal for the endpoint to keep the packet size small and thereby avoid network fragmentation for that flow entirely.

5. IANA Considerations

This document has no actions for IANA.

6. Security Considerations

QUIC is an encrypted and authenticated transport. That means, once the cryptographic handshake is complete, QUIC endpoints discard most packets that are not authenticated, greatly limiting the ability of an attacker to interfere with existing connections.

However, some information is still observerable, as supporting manageability of QUIC traffic inherently involves tradeoffs with the confidentiality of QUIC's control information; this entire document is therefore security-relevant.

More security considerations for QUIC are discussed in [QUIC-TRANSPORT] and [QUIC-TLS], generally considering active or passive attackers in the network as well as attacks on specific QUIC mechanism.

Version Negotiation packets do not contain any mechanism to prevent version downgrade attacks. However, future versions of QUIC that use Version Negotiation packets are require to define a mechanism that is robust against version downgrade attacks. Therefore a network node should not attempt to impact version selection, as version downgrade may result in connection failure.

7. Contributors

The following people have contributed text to sections of this document:

*Dan Druta

*Martin Duke

*Igor Lubashev

*David Schinazi

*Gorry Fairhurst

*Chris Box

8. Acknowledgments

Thanks to Thomas Fossati, Jana Iygengar, Marcus Ihlar for their early reviews and feedback. Special thanks also to Martin Thomson and Martin Duke for their detailed reviews and input. And thanks to Sean Turner, Mike Bishop, Ian Swett, and Nick Banks for their last call reviews.

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. References

9.1. Normative References

[QUIC-TLS] Thomson, M. and S. Turner, "Using TLS to Secure QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-tls-34, 14 January 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-tls-34>.

[QUIC-TRANSPORT] Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", Work in Progress, Internet-Draft, draft-ietf-quic-transport-34, 14 January 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-34>.

9.2. Informative References

[DOTS-ARCH] Mortensen, A., Reddy, T., Andreasen, F., Teague, N., and R. Compton, "DDoS Open Threat Signaling (DOTS) Architecture", Work in Progress, Internet-Draft, draft-ietf-dots-architecture-18, 6 March 2020, <https://datatracker.ietf.org/doc/html/draft-ietf-dots-architecture-18>.

[DPLPMTUD] Fairhurst, G., Jones, T., Tüxen, M., Rüngeler, I., and T. Völker, "Packetization Layer Path MTU Discovery for Datagram Transports", RFC 8899, DOI 10.17487/RFC8899, September 2020, <https://www.rfc-editor.org/rfc/rfc8899>.

[I-D.ietf-quic-applicability]
Kuehlewind, M. and B. Trammell, "Applicability of the QUIC Transport Protocol", Work in Progress, Internet-Draft, draft-ietf-quic-applicability-16, 6 April 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-applicability-16>.

[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-16, 6 April 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-applicability-16>.

[QUIC-HTTP] Bishop, M., "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", Work in Progress, Internet-Draft, draft-ietf-quic-invariants-13, 14 January 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-invariants-13>.

[QUIC-RECOVERY] Iyengar, J. and I. Swett, "QUIC Loss Detection and Congestion Control", Work in Progress, Internet-Draft, draft-ietf-quic-recovery-34, 14 January 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-recovery-34>.

[QUIC_LB] Duke, M., Banks, N., and C. Huitema, "QUIC-LB: Generating Routable QUIC Connection IDs", Work in Progress, Internet-Draft, draft-ietf-quic-load-balancers-13, 28 March 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-load-balancers-13>.

[RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, DOI 10.17487/RFC1191, November 1990, <https://www.rfc-editor.org/rfc/rfc1191>.

[RFC1812] Baker, F., Ed., "Requirements for IP Version 4 Routers", RFC 1812, DOI 10.17487/RFC1812, June 1995, <https://www.rfc-editor.org/rfc/rfc1812>.

[RFC2475] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and W. Weiss, "An Architecture for Differentiated

Services", RFC 2475, DOI 10.17487/RFC2475, December 1998, <https://www.rfc-editor.org/rfc/rfc2475>.

[RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", STD 89, RFC 4443, DOI 10.17487/RFC4443, March 2006, <https://www.rfc-editor.org/rfc/rfc4443>.

[RFC4459] Savola, P., "MTU and Fragmentation Issues with In-the-Network Tunneling", RFC 4459, DOI 10.17487/RFC4459, April 2006, <https://www.rfc-editor.org/rfc/rfc4459>.

[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/rfc/rfc4787>.

[RFC4937] Arberg, P. and V. Mammoliti, "IANA Considerations for PPP over Ethernet (PPPoE)", RFC 4937, DOI 10.17487/RFC4937, June 2007, <https://www.rfc-editor.org/rfc/rfc4937>.

[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/rfc/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/rfc/rfc6066>.

[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>.

[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/rfc/rfc7605>.

[RFC7983] Petit-Huguenin, M. and G. Salgueiro, "Multiplexing Scheme Updates for Secure Real-time Transport Protocol (SRTP) Extension for Datagram Transport Layer Security (DTLS)", RFC 7983, DOI 10.17487/RFC7983, September 2016, <https://www.rfc-editor.org/rfc/rfc7983>.

[RFC8201] McCann, J., Deering, S., Mogul, J., and R. Hinden, Ed., "Path MTU Discovery for IP version 6", STD 87, RFC 8201,

DOI 10.17487/RFC8201, July 2017, <https://www.rfc-editor.org/rfc/rfc8201>.

[RFC8504] Chown, T., Loughney, J., and T. Winters, "IPv6 Node Requirements", BCP 220, RFC 8504, DOI 10.17487/RFC8504, January 2019, <https://www.rfc-editor.org/rfc/rfc8504>.

[TLS-ECH] Rescorla, E., Oku, K., Sullivan, N., and C. A. Wood, "TLS Encrypted Client Hello", Work in Progress, Internet-Draft, draft-ietf-tls-esni-14, 13 February 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-esni-14>.

[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/rfc/rfc8546>.

Appendix A. 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.

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

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")
}
```

A.1. 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)
```

Authors' Addresses

Mirja Kuehlewind
Ericsson

Email: mirja.kuehlewind@ericsson.com

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

Email: ietf@trammell.ch