Stream: Internet Engineering Task Force (IETF)
RFC: 9312
Category: Informational
Published: September 2022
ISSN: 2070-1721
Authors: M. Kühlewind, *Ericsson*
B. Trammell, *Google Switzerland GmbH*

RFC 9312
Manageability of the QUIC Transport Protocol

Abstract

This document discusses manageability of the QUIC transport protocol and focuses on the implications of QUIC's design and wire image on network operations involving QUIC traffic. It is intended as a "user's manual" for the wire image to provide guidance for network operators and equipment vendors who rely on the use of transport-aware network functions.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are candidates for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9312.

Copyright Notice

Table of Contents

1. Introduction 3
2. Features of the QUIC Wire Image 4
2.1. QUIC Packet Header Structure 4
2.2. Coalesced Packets 6
2.3. Use of Port Numbers 6
2.4. The QUIC Handshake 7
2.5. Integrity Protection of the Wire Image 10
2.6. Connection ID and Rebinding 11
2.7. Packet Numbers 11
2.8. Version Negotiation and Greasing 11
3. Network-Visible Information about QUIC Flows 12
3.1. Identifying QUIC Traffic 12
3.1.1. Identifying Negotiated Version 13
3.1.2. First Packet Identification for Garbage Rejection 13
3.2. Connection Confirmation 13
3.3. Distinguishing Acknowledgment Traffic 14
3.4. Server Name Indication (SNI) 14
3.4.1. Extracting Server Name Indication (SNI) Information 14
3.5. Flow Association 15
3.6. Flow Teardown 16
3.7. Flow Symmetry Measurement 16
3.8. Round-Trip Time (RTT) Measurement 16
3.8.1. Measuring Initial RTT 16
3.8.2. Using the Spin Bit for Passive RTT Measurement 16
4. Specific Network Management Tasks 18
4.1. Passive Network Performance Measurement and Troubleshooting 18
4.2. Stateful Treatment of QUIC Traffic 18
4.3. Address Rewriting to Ensure Routing Stability 19

4.4. Server Cooperation with Load Balancers 20
4.5. Filtering Behavior 20
4.6. UDP Blocking, Throttling, and NAT Binding 20
4.7. DDoS Detection and Mitigation 21
4.8. Quality of Service Handling and ECMP Routing 22
4.9. Handling ICMP Messages 22
4.10. Guiding Path MTU 23
5. IANA Considerations 24
6. Security Considerations 24
7. References 24
7.1. Normative References 24
7.2. Informative References 24
Acknowledgments 27
Contributors 28
Authors' Addresses 28

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; therefore, no information in the protocol header is intended to be mutable by the network. This property is enforced through integrity protection of the wire image [WIRE-IMAGE]. Encryption of most transport-layer control signaling means that less information is visible to the network in comparison to TCP.

Integrity protection can also simplify troubleshooting at the end points as none of the nodes on the network path can modify transport layer information. However, it means in-network operations that depend on modification of data (for examples, see [RFC9065]) are not possible

without the cooperation of a QUIC endpoint. Such cooperation might be possible with the introduction of a proxy that authenticates as an endpoint. Proxy operations are not in scope for this document.

Network management is not a one-size-fits-all endeavor; for example, practices considered necessary or even mandatory within enterprise networks with certain compliance requirements would be impermissible on other networks without those requirements. Therefore, presence of a particular practice in this document should not be construed as a recommendation to apply it. For each practice, this document describes what is and is not possible with the QUIC transport protocol as defined.

This document focuses solely on network management practices that observe traffic on the wire. For example, replacement of troubleshooting based on observation with active measurement techniques is therefore out of scope. A more generalized treatment of network management operations on encrypted transports is given in [RFC9065].

QUIC-specific terminology used in this document is defined in [QUIC-TRANSPORT].

2. Features of the QUIC Wire Image

This section discusses aspects of the QUIC transport protocol that have an impact on the design and operation of devices that forward QUIC packets. Therefore, this section is primarily considering the unencrypted part of QUIC's wire image [WIRE-IMAGE], which is defined 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 addresses 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.

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. It contains a version number, as well as Source and Destination Connection IDs for associating packets with a QUIC connection. 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].

In version 1 of QUIC, the long header is used during connection establishment to transmit CRYPTO handshake data, perform version negotiation, retry, and send 0-RTT data.

Short headers are used after a connection establishment in version 1 of QUIC and expose only an optional Destination Connection ID and the initial flags byte with the spin bit for RTT measurement.

The following information is exposed in QUIC packet headers in all versions of QUIC (as specified in [QUIC-INVARIANTS]):

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 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 [RFC7801]. An IANA registry contains the values of all standardized versions of QUIC, and may contain some proprietary versions (see Section 22.2 of [QUIC-TRANSPORT]). However, other versions of QUIC can be expected to be seen in the Internet at any given time.

Source and Destination Connection ID: Short and long headers carry a Destination Connection ID, which is a variable-length field. If the Destination Connection ID is not zero length, it can be used to identify the connection associated with a QUIC packet for load balancing and NAT rebinding purposes; see Sections 4.4 and 2.6. Long packet headers additionally carry a Source Connection ID. The Source Connection ID is only present on long headers and indicates the Destination Connection ID that the other endpoint should use when sending packets. 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, as it is known from preceding long header packets.

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

"Fixed Bit": In version 1 of QUIC, the second-most-significant bit of the first octet is set to 1, unless the packet is a Version Negotiation packet or an extension is used that modifies the usage of this bit. If the bit is set to 1, it enables endpoints to easily 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 [QUIC-GREASE]. 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 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. Retry packets are integrity protected. Transport parameters are used to authenticate the contents of Retry packets later in the handshake. For other kinds of packets, version 1 of QUIC cryptographically protects 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 can be decoded in long headers while it is implicit (depending on Destination Connection ID length) in short headers. The length of the packet number is cryptographically protected.

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 protected.

2.2. Coalesced Packets

Multiple QUIC packets may be coalesced into a single 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 field is a variable-length field, and its position in the header also varies depending on the lengths of the Source and Destination Connection IDs; see Section 17.2 of [QUIC-TRANSPORT].

2.3. Use of Port Numbers

Applications that have a mapping for TCP and 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 the HTTP Alternative Services mechanism [RFC7838] 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 (protocol, source, and destination IP address and source and destination port). However, if the connection ID is zero length, all packets of the 5-tuple likely belong to the same QUIC connection.

2.4. The QUIC Handshake

New QUIC connections are established using a handshake that is distinguishable on the wire (see Section 3.1 for details) 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, we examine each of the datagrams in detail.

The QUIC handshake can normally be recognized on the wire through four flights of datagrams labeled "Client Initial", "Server Initial", "Client Completion", and "Server Completion" as illustrated in Figure 1.

A handshake starts with the client sending one or more datagrams containing Initial packets (detailed in Figure 2), which elicits the Server Initial response (detailed in Figure 3), which typically contains three types of packets: Initial packet(s) with the beginning of the server's side of the TLS handshake, Handshake packet(s) with the rest of the server's portion of the TLS handshake, and 1-RTT packet(s), if present.

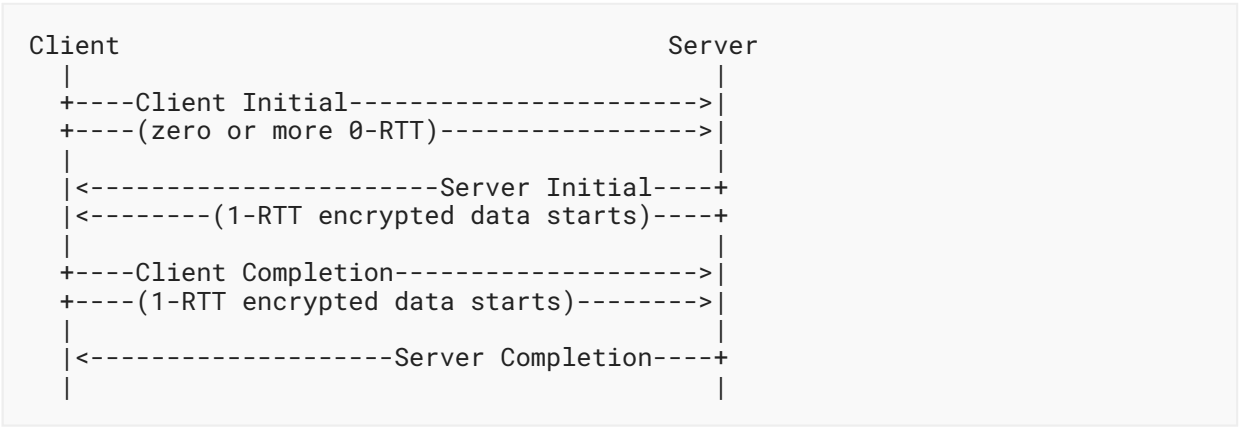

Figure 1: General Communication Pattern Visible in the QUIC Handshake

As shown here, the client can send 0-RTT data as soon as it has sent its ClientHello and the server can send 1-RTT data as soon as it has sent its ServerHello. The Client Completion flight contains at least one Handshake packet and could also include an Initial packet. During the handshake, QUIC packets in separate contexts can be coalesced (see Section 2.2) in order to reduce the number of UDP datagrams sent during the handshake.

Handshake packets can arrive out-of-order without impacting the handshake as long as the reordering was not accompanied by extensive delays that trigger a spurious Probe Timeout (Section 6.2 of [QUIC-RECOVERY]). If QUIC packets get lost or reordered, packets belonging to the same flight might not be observed in close time succession, though the sequence of the flights will not change because one flight depends upon the peer's previous flight.

Datagrams that contain an 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 packets of at least this size for QUIC to be used.

The content of Initial packets is encrypted using Initial Secrets, which are derived from a per-version constant and the client's Destination Connection ID. That content is therefore observable by any on-path device that knows the per-version constant and is considered visible in this illustration. The content of QUIC Handshake packets is encrypted using keys established during the initial handshake exchange and is therefore not visible.

Initial, Handshake, and 1-RTT packets belong to different cryptographic and transport contexts. The Client Completion (Figure 4) and the Server Completion (Figure 5) flights conclude the Initial and Handshake contexts by sending final acknowledgments and CRYPTO frames.

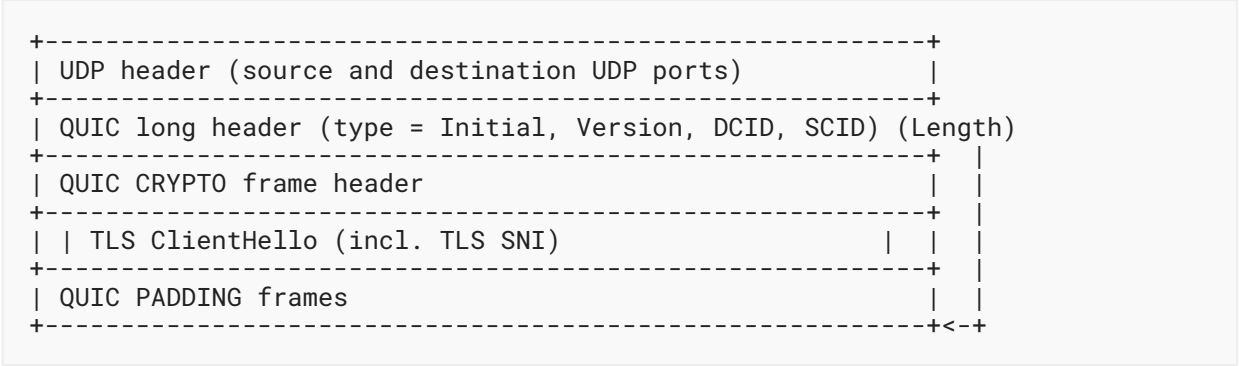

Figure 2: Example Client Initial Datagram Without 0-RTT

A Client Initial packet exposes the Version, Source, and Destination Connection IDs without encryption. The payload of the Initial packet is protected using the Initial secret. The complete TLS ClientHello, including any TLS Server Name Indication (SNI) present, is sent in one or more CRYPTO frames across one or more QUIC Initial packets.

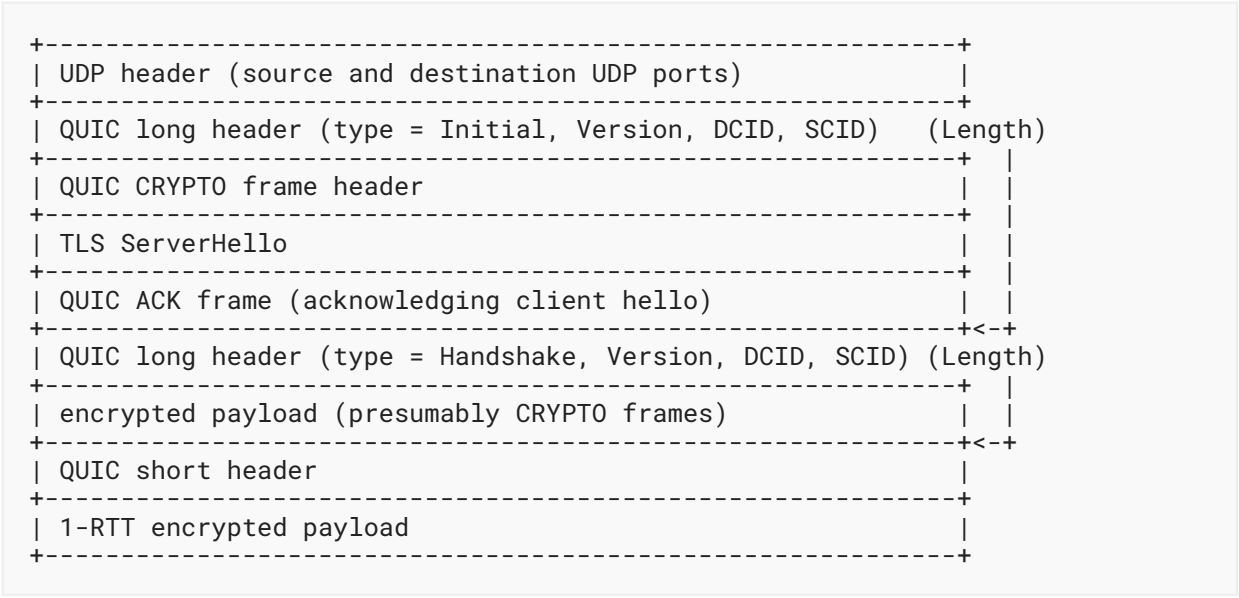

Figure 3: Coalesced Server Initial Datagram Pattern

The Server Initial datagram also exposes the version number and the Source and Destination Connection IDs in the clear; the payload of the Initial packet is protected using the Initial secret.

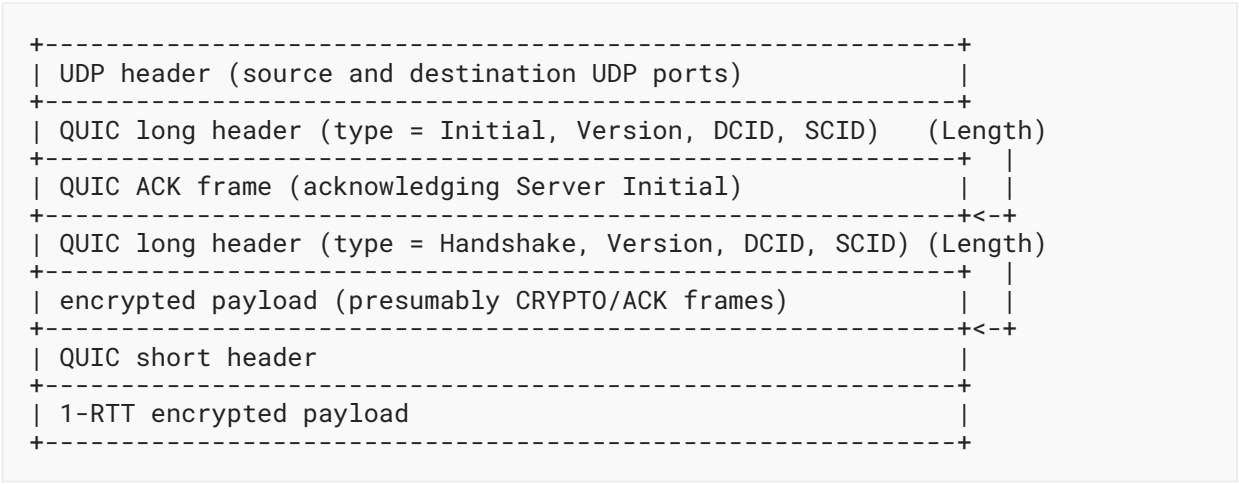

Figure 4: Coalesced Client Completion Datagram Pattern

The Client Completion flight does not expose any additional information; however, as the Destination Connection ID is server-selected, it usually is not the same ID that is sent in the Client Initial. Client Completion flights contain 1-RTT packets that indicate the handshake has completed (see Section 3.2) on the client and for three-way handshake RTT estimation as in Section 3.8.

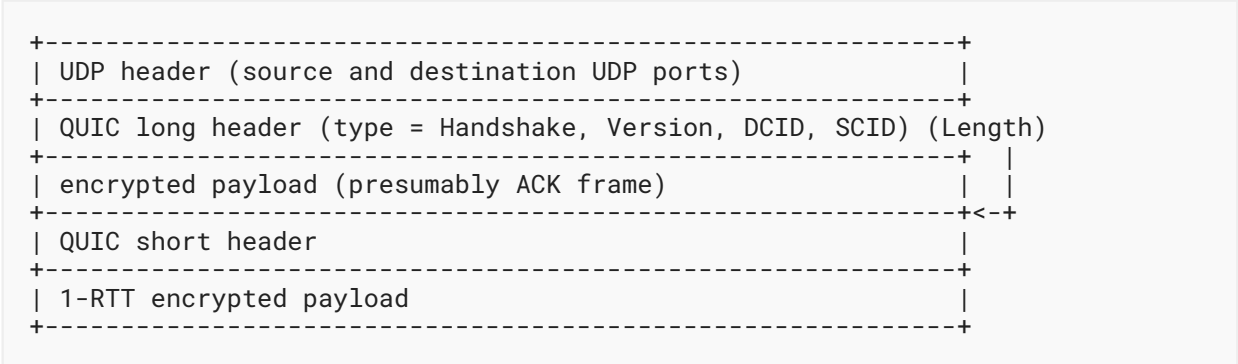

Figure 5: Coalesced Server Completion Datagram Pattern

Similar to Client Completion, Server Completion does not expose additional information; observing it serves only to determine that the handshake has completed.

When the client uses 0-RTT data, the Client Initial flight can also include one or more 0-RTT packets as shown in Figure 6.

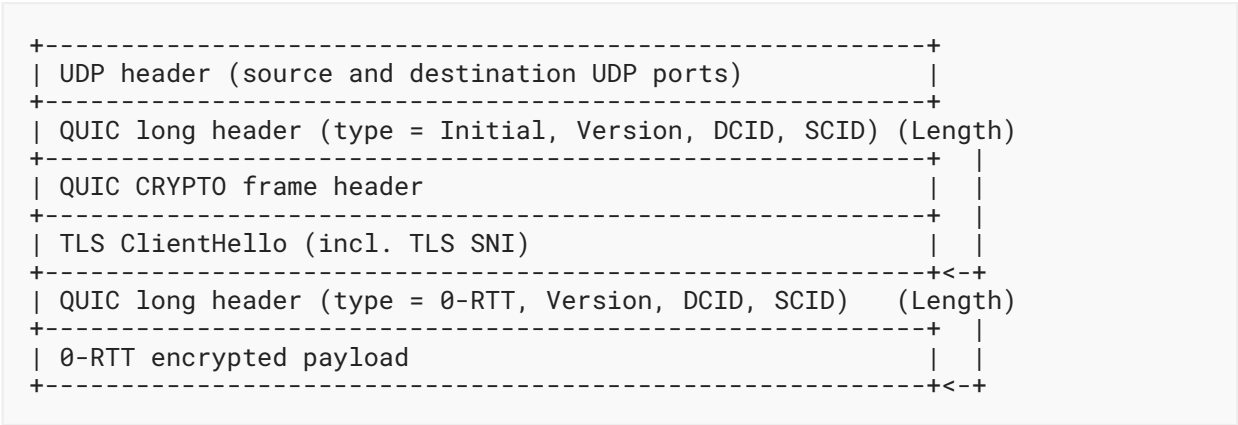

Figure 6: Coalesced 0-RTT Client Initial Datagram

When a 0-RTT packet is coalesced with an Initial packet, the datagram will be padded to 1200 bytes. Additional datagrams containing only 0-RTT packets with long headers can be sent after the client Initial packet, which contains more 0-RTT data. The amount of 0-RTT protected data that can be sent in the first flight is limited by the initial congestion window, typically to 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 reapplying 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 a connection establishment failure later.

2.6. Connection ID and Rebinding

The connection ID in the QUIC packet headers allows association of QUIC packets using information independent of the 5-tuple. This allows rebinding of a connection after one of the endpoints (usually the client) has experienced an address change. Further, it can be used by in-network devices to ensure that related 5-tuple flows are appropriately balanced together (see Section 4.4).

Client and server each choose a connection ID during the handshake; for example, a server might request that a client use a connection ID, whereas the client might choose a zero-length value. 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 necessarily 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 that 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 Initial packets (which are 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 could 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 of the selection mechanism. Further, a client may send an Initial packet with a reserved version number to trigger version negotiation. In the Version Negotiation packet, the connection IDs of the client's 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. Therefore, new versions (both experimental and IETF standard versions) will be deployed on the Internet more often than with other commonly deployed Internet and transport-layer protocols. Use of the Version field for traffic recognition will therefore behave differently than with these 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. Reliance on the Version field for the purpose of admission control is also likely to lead to unintended failure modes. Admission of QUIC traffic regardless of version avoids these failure modes, avoids unnecessary deployment delays, and supports continuous version-based evolution.

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, but typically without access to any keying information.

3.1. Identifying QUIC Traffic

The QUIC wire image is not specifically designed to be distinguishable from other UDP traffic by a passive observer in the network. While certain QUIC applications may be heuristically identifiable on a per-application basis, there is no general method for distinguishing QUIC traffic from otherwise unclassifiable UDP traffic on a given link. Therefore, any unrecognized UDP traffic may be QUIC traffic.

At the time of writing, two application bindings for QUIC have been published or adopted by the IETF: HTTP/3 [QUIC-HTTP] and DNS over Dedicated QUIC Connections [RFC9250]. These are both known to have active Internet deployments, so an assumption that all QUIC traffic is HTTP/3 is not valid. HTTP/3 uses UDP port 443 by convention but various methods can be used to specify alternate port numbers. Other applications (e.g., Microsoft's SMB over QUIC) also use UDP port 443 by default. Therefore, simple assumptions about whether a given flow is using QUIC (or indeed which application might be using QUIC) based solely upon a UDP port number may not hold; see 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 [QUIC-GREASE].

Even though transport parameters transmitted in the client's Initial packet are observable by the network, they cannot be modified by the network without causing a 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 might infer the version number in use by observing the handshake. If the version number in an Initial packet of the server response is subsequently seen in a packet from the client, that version has been accepted by both endpoints to be used for the rest of the connection (see Section 2 of [QUIC-VERSION-NEGOTIATION]).

The negotiated version cannot be identified for flows in 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 Connection Confirmation 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 ClientHello. Since packets may be reordered or lost 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 packets that carry only acknowledgment (ACK-only) information from packets carrying upper-layer data in order to attempt to enhance performance (for example, by queuing ACKs differently or manipulating ACK signaling [RFC3449]). Distinguishing ACK packets is possible in TCP, but is 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) extension [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. If present, SNI information is available to unidirectional observers on the client-to-server path if it.

The TLS ClientHello may also contain an Application-Layer Protocol Negotiation (ALPN) extension [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 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, SNI can be derived from the client's Initial packets by calculating the Initial secret to decrypt the packet payload and parsing the QUIC CRYPTO frames 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 (the second through fifth 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, which indicates 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. However, it is expected that these versions will gradually disappear over time and therefore do not require any special consideration or treatment.

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 from the packet. The Initial secret is

calculated 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.

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.

To determine the end of the packet 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 eighth bits of the header as described in Section 17.2 of [QUIC-TRANSPORT], but is protected 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 packets can be parsed to detect the CRYPTO frames that contain the TLS ClientHello, which then can be parsed similarly to TLS over TCP connections. Note that there can be multiple CRYPTO frames spread out over one or more Initial packets and they might not be in order, so reassembling the CRYPTO stream by parsing offsets and lengths is required. Further, the client's Initial packets may contain other frames, so the first bytes of each frame need to be checked to identify the frame type and determine whether the frame can be skipped over. Note that the length of the frames is dependent on the frame type; see Section 18 of [QUIC-TRANSPORT]. For example, PADDING frames (each consisting of a single zero byte) may occur before, after, or between CRYPTO frames. However, extensions might define additional frame types. If an unknown frame type is encountered, it is impossible to know the length of that frame, which prevents skipping over it; therefore, parsing fails.

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. 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; therefore, it is not possible for a passive observer to link intended changes of address using the connection ID.

When one endpoint's address unintentionally changes, 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 5-tuple but different connection IDs might or might not belong to the same connection. Likewise, packets with the same connection ID but different 5-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. Therefore, stateful devices on path such as NATs and firewalls must 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 it is 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. Note that QUIC packets containing only control frames (such as ACK-only packets) may be padded. Padding, though optional, may conceal connection roles or flow symmetry information.

3.8. Round-Trip Time (RTT) Measurement

The round-trip time (RTT) of QUIC flows can be inferred by observation once per flow during the handshake 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]. RTT measurement is available to unidirectional observers when the spin bit is enabled.

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 all transport- and application-layer delay at both endpoints.

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. 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, the latency spin bit in each direction changes value once per RTT any time that both endpoints are sending packets continuously. 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 all transport protocol delay (e.g., delayed sending of acknowledgments) 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. For example, if the sender only sends small amounts of application traffic periodically, where the periodicity is longer than the RTT, spin bit measurements provide information about the application period rather than 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 between 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 one-way packet delay variance as seen by an application end-point).

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. Limited observation of upstream congestion may be possible via the observation of Congestion Experienced (CE) markings in the IP header [RFC3168] on ECN-enabled QUIC traffic.

On-path devices can also make measurements of RTT, loss, and other performance metrics when information is carried in an additional network-layer packet header (Section 6 of [RFC9065] describes the use of Operations, Administration, and Management (OAM) information). Using network-layer approaches also has the advantage that common observation and analysis tools can be consistently used for multiple transport protocols; however, these techniques are often limited to measurements within one or multiple cooperating domains.

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 least-recently-used eviction depending on application requirements.

While QUIC has no clear network-visible end-of-flow 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 network state timeout that is not less than 2 minutes for most UDP traffic. However, in practice, a QUIC endpoint can experience lower timeouts in the range of 30 to 60 seconds [QUIC-TIMEOUT].

In contrast, [RFC5382] recommends a state 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 to 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.

Therefore, a state timeout of at least two minutes is recommended for QUIC traffic, even when lower state timeouts are used for other UDP 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 unnecessarily long delays in an otherwise working connection.

Furthermore, not all endpoints use routing architectures where connections will survive a port or address change. 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. In particular, the use of the connection ID for functions that require state to make a forwarding decision 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 TCP fallback. 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 a connection to survive client 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 8 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, which protects the new path from overload.

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 are often also used in other scenarios where packet filtering is desired. Though the guidance there holds, a particularly unwise behavior admits a handful of UDP packets and then makes a decision to whether or not filter later packets in the same connection. QUIC applications are encouraged to fall back to TCP if early packets do not arrive at their destination [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, Throttling, and NAT Binding

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; therefore, some networks 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; if this is 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 indiscriminately. When the handshake is blocked, QUIC-capable applications may fall back to TCP. However, blocking a random fraction of QUIC packets

across 4-tuples will allow many QUIC handshakes to complete, preventing TCP fallback, but these 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.

Note that some source ports are assumed to be reflection attack vectors by some servers; see Section 8.1 of [QUIC-APPLICABILITY]. As a result, NAT binding to these source ports can result in that traffic being blocked.

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 characterizing anomalous traffic. Other uses include support for security audits (e.g., verifying the compliance with cipher suites), client and application fingerprinting for inventory, and providing 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 mentioned in Section 4.2 above may be useful during classification.

Note that using 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 cannot 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 supported much easier, 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 0-RTT data 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 SYN cookies [RFC4987] are a well-established method of mitigating some kinds of TCP DDoS attacks. QUIC Retry packets are the functional analogue to SYN cookies, 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-RETRY] for standard ways for intermediaries to send Retry packets on behalf of consenting servers.

4.8. Quality of Service Handling and ECMP Routing

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 active QUIC connection get uniform treatment.

Using ECMP to distribute packets from a single flow across multiple network paths or any other nonuniform 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 on the loss recovery mechanism that is implemented, QUIC may be more tolerant of packet reordering than typical TCP traffic (see Section 2.7). However, the recovery mechanism used by a flow cannot be known by the network and therefore reordering tolerance should be considered as unknown.

Note that the 5-tuple of a QUIC connection can change due to migration. In this case different flows are observed by the path and may be treated differently, as congestion control is usually reset on migration (see also Section 3.5).

4.9. Handling ICMP Messages

Datagram Packetization Layer PMTU Discovery (DPLPMTUD) can be used by QUIC to probe for the supported PMTU. DPLPMTUD optionally uses ICMP messages (e.g., IPv6 Packet Too Big (PTB) messages). Given known attacks with the use of ICMP messages, the use of DPLPMTUD 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 network segments support 1500-byte packets, but can only do so by fragmenting at a lower layer before traversing a network segment with a smaller MTU, and then reassembling within the network segment. This is permissible even when the IP layer is IPv6 or IPv4 with the Don't Fragment (DF) bit set, because fragmentation 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 to avoid exceeding limited reassembly capacity.

For TCP, Maximum Segment Size (MSS) clamping (Section 3.2 of [RFC4459]) is often used to change the sender's TCP maximum segment size, but QUIC requires a different approach. Section 14 of [QUIC-TRANSPORT] advises senders to probe larger sizes using DPLPMTUD [DPLPMTUD] or Path Maximum Transmission Unit Discovery (PMTUD) [RFC1191] [RFC8201]. This mechanism encourages senders to approach the maximum packet size, which could then cause fragmentation within a network segment of which they may not be aware.

If path performance is limited when forwarding 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.

Networks with configurations that would lead to fragmentation of large packets within a network segment should drop such packets rather than fragmenting them. Network operators who plan to implement a more selective policy may start by focusing 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 networks supporting QUIC, it is recommended that a path drops any packet larger than the fragmentation size. When a QUIC endpoint uses DPLPMTUD, it will use a QUIC probe packet to discover the PMTU. If this probe is lost, it will 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 PTB in this case. PMTUD relies upon an endpoint receiving such PTB messages [RFC8201], whereas DPLPMTUD does not reply upon these messages, but can still optionally use these to improve performance Section 4.6 of [DPLPMTUD].

A network cannot know in advance which discovery method is used by a QUIC endpoint, so it should send a PTB message in addition to dropping an 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 for PMTU discovery. This provides a signal to the endpoint to prevent the packet size from growing too large, which can entirely avoid network segment fragmentation for that flow.

Endpoints can cache PMTU information in the IP-layer cache. This short-term consistency between the PMTU for flows can help avoid an endpoint using a PMTU that is inefficient. The IP cache can also influence the PMTU value of other IP flows that use the same path [RFC8201] [DPLPMTUD], including IP packets carrying protocols other than QUIC. The representation of an IP path is implementation specific [RFC8201].

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 observable as supporting manageability of QUIC traffic inherently involves trade-offs 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], which generally consider 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 required 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. References

7.1. Normative References

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

[QUIC-TRANSPORT] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/info/rfc9000>.

7.2. Informative References

[DOTS-ARCH] Mortensen, A., Ed., Reddy.K, T., Ed., Andreasen, F., Teague, N., and R. Compton, "DDoS Open Threat Signaling (DOTS) Architecture", RFC 8811, DOI 10.17487/RFC8811, August 2020, <https://www.rfc-editor.org/info/rfc8811>.

[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/info/rfc8899>.

[IPIM] Allman, M., Beverly, R., and B. Trammell, "Principles for Measurability in Protocol Design", 9 December 2016, <https://arxiv.org/abs/1612.02902>.

[QUIC-APPLICABILITY] Kühlewind, M. and B. Trammell, "Applicability of the QUIC Transport Protocol", RFC 9308, DOI 10.17487/RFC9308, September 2022, <https://www.rfc-editor.org/info/rfc9308>.

[QUIC-GREASE] Thomson, M., "Greasing the QUIC Bit", RFC 9287, DOI 10.17487/RFC9287, August 2022, <https://www.rfc-editor.org/info/rfc9287>.

[QUIC-HTTP] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <https://www.rfc-editor.org/info/rfc9114>.

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

[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-14, 11 July 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-load-balancers-14>.

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

[QUIC-RETRY] Duke, M. and N. Banks, "QUIC Retry Offload", Work in Progress, Internet-Draft, draft-ietf-quic-retry-offload-00, 25 May 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-retry-offload-00>.

[QUIC-TIMEOUT] Roskind, J., "QUIC", IETF-88 TSV Area Presentation, 7 November 2013, <https://www.ietf.org/proceedings/88/slides/slides-88-tsvarea-10.pdf>.

[QUIC-VERSION-NEGOTIATION] Schinazi, D. and E. Rescorla, "Compatible Version Negotiation for QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-version-negotiation-10, 27 September 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-version-negotiation-10>.

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

[RFC1812] Baker, F., Ed., "Requirements for IP Version 4 Routers", RFC 1812, DOI 10.17487/RFC1812, June 1995, <https://www.rfc-editor.org/info/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/info/rfc2475>.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <https://www.rfc-editor.org/info/rfc3168>.

[RFC3449] Balakrishnan, H., Padmanabhan, V., Fairhurst, G., and M. Sooriyabandara, "TCP Performance Implications of Network Path Asymmetry", BCP 69, RFC 3449, DOI 10.17487/RFC3449, December 2002, <https://www.rfc-editor.org/info/rfc3449>.

[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/info/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/info/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/info/rfc4787>.

[RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007, <https://www.rfc-editor.org/info/rfc4987>.

[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>.

[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>.

[RFC7801] Dolmatov, V., Ed., "GOST R 34.12-2015: Block Cipher "Kuznyechik"", RFC 7801, DOI 10.17487/RFC7801, March 2016, <https://www.rfc-editor.org/info/rfc7801>.

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

[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/info/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/info/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/info/rfc8504>.

[RFC9065] Fairhurst, G. and C. Perkins, "Considerations around Transport Header Confidentiality, Network Operations, and the Evolution of Internet Transport Protocols", RFC 9065, DOI 10.17487/RFC9065, July 2021, <https://www.rfc-editor.org/info/rfc9065>.

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

[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", Traffic Measurement and Analysis, TMA 2014, Lecture Notes in Computer Science, vol. 8406, pp. 15-25, DOI 10.1007/978-3-642-54999-1_2, April 2014, <https://link.springer.com/chapter/10.1007/978-3-642-54999-1_2>.

[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>.

Acknowledgments

Special thanks to last call reviewers Elwyn Davies, Barry Leiba, Al Morton, and Peter Saint-Andre.

This work was 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.

Contributors

The following people have contributed significant text to and/or feedback on this document:

Chris Box

Dan Druta

David Schinazi

Gorry Fairhurst

Ian Swett

Igor Lubashev

Jana Iyengar

Jared Mauch

Lars Eggert

Lucas Purdue

Marcus Ihlar

Mark Nottingham

Martin Duke

Martin Thomson

Matt Joras

Mike Bishop

Nick Banks

Thomas Fossati

Sean Turner

Authors' Addresses

Mirja Kühlewind
Ericsson
Email: mirja.kuehlewind@ericsson.com

Brian Trammell
Google Switzerland GmbH
Gustav-Gull-Platz 1
CH-8004 Zurich
Switzerland
Email: ietf@trammell.ch