Workgroup: Network Working Group
Internet-Draft:
draft-ietf-quic-applicability-10
Published: 22 February 2021
Intended Status: Informational
Expires: 26 August 2021
A M. Kuehlewind B. Trammell
uEricsson Google
t
h
o
r
s
:

Applicability of the QUIC Transport Protocol

Abstract

This document discusses the applicability of the QUIC transport protocol, focusing on caveats impacting application protocol development and deployment over QUIC. Its intended audience is designers of application protocol mappings to QUIC, and implementors of these application protocols.

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 26 August 2021.

Table of Contents

1. Introduction

QUIC [QUIC] is a new transport protocol providing a number of advanced features. While initially designed for the HTTP use case, it provides capabilities that can be used with a much wider variety of applications. QUIC is encapsulated in UDP. QUIC version 1 integrates TLS 1.3 [TLS13] to encrypt all payload data and most control information. The version of HTTP that uses QUIC is known as HTTP/3 [QUIC-HTTP].

This document provides guidance for application developers that want to use the QUIC protocol without implementing it on their own. This includes general guidance for applications operating over HTTP/3 or directly over QUIC.

In the following sections we discuss specific caveats to QUIC's applicability, and issues that application developers must consider when using QUIC as a transport for their application.

2. The Necessity of Fallback

QUIC uses UDP as a substrate. This enables userspace implementation and permits traversal of network middleboxes (including NAT) without requiring updates to existing network infrastructure.

While recent measurements have shown no evidence of a widespread, systematic disadvantage of UDP traffic compared to TCP in the Internet [Edeline16], somewhere between three [Trammell16] and five [Swett16] percent of networks block all UDP traffic. All applications running on top of QUIC must therefore either be prepared to accept connectivity failure on such networks or be engineered to fall back to some other transport protocol. In the case of HTTP, this fallback is TLS over TCP.

The IETF TAPS specifications [I-D.ietf-taps-arch] describe a system with a common API for multiple protocols and some of the implications of fallback between these different protocols, specifically precluding fallback to insecure protocols or to weaker versions of secure protocols.

An application that implements fallback needs to consider the security consequences. A fallback to TCP and TLS exposes control information to modification and manipulation in the network. Further downgrades to older TLS versions than used in QUIC, which is 1.3, might result in significantly weaker cryptographic protection. For example, the results of protocol negotiation [RFC7301] only have confidentiality protection if TLS 1.3 is used.

These applications must operate, perhaps with impaired functionality, in the absence of features provided by QUIC not present in the fallback protocol. For fallback to TLS over TCP, the most obvious difference is that TCP does not provide stream multiplexing and therefore stream multiplexing would need to be implemented in the application layer if needed. Further, TCP implementations and network paths often do not support the Fast Open option, which is analogous to 0-RTT session resumption. Note that there is some evidence of middleboxes blocking SYN data even if TFO was successfully negotiated (see [PaaschNanog]). And even if Fast Open successfully operates end-to-end, it is limited to a single packet of payload, unlike QUIC 0-RTT.

Moreover, while encryption (in this case TLS) is inseparably integrated with QUIC, TLS negotiation over TCP can be blocked. If TLS over TCP cannot be supported, the connection should be aborted instead, in order to enable the application to present a suitable prompt to the user that secure communication is unavailable.

In summary, any fallback mechanism is likely to impose a degradation of performance and can degrade security; however, fallback must not silently violate the application's expectation of confidentiality or integrity of its payload data.

3. Zero RTT

QUIC provides for 0-RTT connection establishment. Though the same facility exists in TLS 1.3 with TCP, 0-RTT presents opportunities and challenges for applications using QUIC.

A transport protocol that provides 0-RTT connection establishment is qualitatively different than one that does not from the point of view of the application using it. Relative trade-offs between the cost of closing and reopening a connection and trying to keep it open are different; see Section 3.2.

An application needs to deliberately choose to use 0-RTT, as 0-RTT carries a risk of replay attack. Application protocols that use 0-RTT require a profile that describes the types of information that can be safely sent. For HTTP, this profile is described in [HTTP-REPLAY].

3.1. Replay Attacks

Retransmission or (malicious) replay of data contained in 0-RTT packets could cause the server side to receive two copies of the same data.

Application data sent by the client in 0-RTT packets could be processed more than once if it is replayed. Applications need to be aware of what is safe to send in 0-RTT. Application protocols that seek to enable the use of 0-RTT need a careful analysis and a description of what can be sent in 0-RTT; see Section 5.6 of [QUIC-TLS].

In some cases, it might be sufficient to limit application data sent in 0-RTT to that which only causes actions at a server that are known to be free of lasting effect. Initiating data retrieval or establishing configuration are examples of actions that could be safe. Idempotent operations - those for which repetition has the same net effect as a single operation - might be safe. However, it is also possible to combine individually idempotent operations into a non-idempotent sequence of operations.

Once a server accepts 0-RTT data there is no means of selectively discarding data that is received. However, protocols can define ways to reject individual actions that might be unsafe if replayed.

Some TLS implementations and deployments might be able to provide partial or even complete replay protection, which could be used to manage replay risk.

3.2. Session resumption versus Keep-alive

Because QUIC is encapsulated in UDP, applications using QUIC must deal with short network idle timeouts. Deployed stateful middleboxes will generally establish state for UDP flows on the first packet sent, and keep state for much shorter idle periods than for TCP. [RFC5382] suggests a TCP idle period of at least 124 minutes, though there is not evidence of widespread implementation of this guideline in the literature. Short network timeout for UDP, however, is well-documented. According to a 2010 study ([Hatonen10]), UDP applications can assume that any NAT binding or other state entry can expire after just thirty seconds of inactivity. Section 3.5 of [RFC8085] further discusses keep-alive intervals for UDP: it requires a minimum value of 15 seconds, but recommends larger values, or omitting keep-alive entirely.

By using a connection ID, QUIC is designed to be robust to NAT address rebinding after a timeout. However, this only helps if one endpoint maintains availability at the address its peer uses, and the peer is the one to send after the timeout occurs.

Some QUIC connections might not be robust to NAT rebinding because the routing infrastructure (in particular, load balancers) uses the address/port four-tuple to direct traffic. Furthermore, middleboxes with functions other than address translation could still affect the path. In particular, some firewalls do not admit server traffic for which the firewall has no recent state for a corresponding packet sent from the client.

QUIC applications can adjust idle periods to manage the risk of timeout. Idle periods and the network idle timeout are distinct from the connection idle timeout, which is defined as the minimum of either endpoint's idle timeout parameter; see Section 10.1 of [QUIC]). There are three options:

- Ignore the issue, if the application-layer protocol consists only of interactions with no or very short idle periods, or the protocol's resistance to NAT rebinding is sufficient.

- Ensure there are no long idle periods.

- Resume the session after a long idle period, using 0-RTT resumption when appropriate.

The first strategy is the easiest, but it only applies to certain applications.

Either the server or the client in a QUIC application can send PING frames as keep-alives, to prevent the connection and any on-path state from timing out. Recommendations for the use of keep-alives are application-specific, mainly depending on the latency requirements and message frequency of the application. In this case, the application mapping must specify whether the client or server is responsible for keeping the application alive. While [Hatonen10] suggests that 30 seconds might be a suitable value for the public Internet when a NAT is on path, larger values are preferable if the deployment can consistently survive NAT rebinding or is known to be in a controlled environment (e.g. data centres) in order to lower network and computational load.

Sending PING frames more frequently than every 30 seconds over long idle periods may result in excessive unproductive traffic in some situations, and to unacceptable power usage for power-constrained (mobile) devices. Additionally, timeouts shorter than 30 seconds can make it harder to handle transient network interruptions, such as VM migration or coverage loss during mobilty. See [RFC8085], especially Section 3.5.

Alternatively, the client (but not the server) can use session resumption instead of sending keepalive traffic. In this case, a client that wants to send data to a server over a connection idle longer than the server's idle timeout (available from the idle_timeout transport parameter) can simply reconnect. When possible, this reconnection can use 0-RTT session resumption, reducing the latency involved with restarting the connection. Of course, this approach is only valid in cases in which 0-RTT data is safe, when the client is the restarting peer, and when the data to be sent is idempotent. It is also not applicable when the

application binds external state to the connection, as this state cannot reliably be transferred to a resumed connection.

The tradeoffs between resumption and keep-alives need to be evaluated on a per-application basis. In general, applications should use keep-alives only in circumstances where continued communication is highly likely; [QUIC-HTTP], for instance, recommends using keep-alives only when a request is outstanding.

4. Use of Streams

QUIC's stream multiplexing feature allows applications to run multiple streams over a single connection, without head-of-line blocking between streams, associated at a point in time with a single five-tuple. Stream data is carried within frames, where one QUIC packet on the wire can carry one or multiple stream frames.

Streams can be unidirectional or bidirectional, and a stream may be initiated either by client or server. Only the initiator of a unidirectional stream can send data on it.

Streams and connections can each carry a maximum of 2^{62}-1 bytes in each direction, due to encoding limitations on stream offsets and connection flow control limits. In the presently unlikely event that this limit is reached by an application, a new connection would need to be established.

Streams can be independently opened and closed, gracefully or abruptly. An application can gracefully close the egress direction of a stream by instructing QUIC to send a FIN bit in a STREAM frame. It cannot gracefully close the ingress direction without a peer-generated FIN, much like in TCP. However, an endpoint can abruptly close the egress direction or request that its peer abruptly close the ingress direction; these actions are fully independent of each other.

QUIC does not provide an interface for exceptional handling of any stream. If a stream that is critical for an application is closed, the application can generate error messages on the application layer to inform the other end and/or the higher layer, which can eventually reset the QUIC connection.

Mapping of application data to streams is application-specific and described for HTTP/3 in [QUIC-HTTP]. There are a few general principles to apply when designing an application's use of streams:

* A single stream provides ordering. If the application requires certain data to be received in order, that data should be sent on the same stream.

* Multiple streams provide concurrency. Data that can be processed independently, and therefore would suffer from head of line blocking if forced to be received in order, should be transmitted over separate streams.

* Streams can provide message orientation, and allow messages to be cancelled. If one message is mapped to a single stream, resetting

the stream to expire an unacknowledged message can be used to emulate partial reliability for that message.

If a QUIC receiver has opened the maximum allowed concurrent streams, and the sender indicates that more streams are needed, it does not automatically lead to an increase of the maximum number of streams by the receiver. Therefore it can be valuable to expose the maximum number of allowed, currently open, and currently used streams to the application to make the mapping of data to streams dependent on this information.

QUIC assigns a numerical identifier to each stream, called the Stream ID. While the relationship between these identifiers and stream types is clearly defined in version 1 of QUIC, future versions might change this relationship for various reasons. QUIC implementations should expose the properties of each stream (which endpoint initiated the stream, whether the stream is unidirectional or bidirectional, the Stream ID used for the stream); applications should query for these properties rather than attempting to infer them from the Stream ID.

The method of allocating stream identifiers to streams opened by the application might vary between transport implementations. Therefore, an application should not assume a particular stream ID will be assigned to a stream that has not yet been allocated. For example, HTTP/3 uses Stream IDs to refer to streams that have already been opened, but makes no assumptions about future Stream IDs or the way in which they are assigned Section 6 of [QUIC-HTTP]).

4.1. Stream versus Flow Multiplexing

Streams are meaningful only to the application; since stream information is carried inside QUIC's encryption boundary, no information about the stream(s) whose frames are carried by a given packet is visible to the network. Therefore stream multiplexing is not intended to be used for differentiating streams in terms of network treatment. Application traffic requiring different network treatment should therefore be carried over different five-tuples (i.e. multiple QUIC connections). Given QUIC's ability to send application data in the first RTT of a connection (if a previous connection to the same host has been successfully established to provide the necessary credentials), the cost of establishing another connection is extremely low.

4.2. Prioritization

Stream prioritization is not exposed to either the network or the receiver. Prioritization is managed by the sender, and the QUIC transport should provide an interface for applications to prioritize streams [QUIC]. Applications can implement their own prioritization scheme on top of QUIC: an application protocol that runs on top of QUIC can define explicit messages for signaling priority, such as those defined for HTTP/2; it can define rules that allow an endpoint to determine priority based on context; or it can provide a higher level interface and leave the determination to the application on top.

Priority handling of retransmissions can be implemented by the sender in the transport layer. [QUIC] recommends retransmitting lost data before new data, unless indicated differently by the application. Currently, QUIC only provides fully reliable stream transmission, which means that prioritization of retransmissions will be beneficial in most cases, by filling in gaps and freeing up the flow control window. For partially reliable or unreliable streams, priority scheduling of retransmissions over data of higher-priority streams might not be desirable. For such streams, QUIC could either provide an explicit interface to control prioritization, or derive the prioritization decision from the reliability level of the stream.

4.3. Ordered and Reliable Delivery

QUIC streams enable ordered and reliable delivery. Though it is possible for an implementation to provide options that use streams for partial reliability or out-of-order delivery, most implementations will assume that data is reliably delivered in order.

Under this assumption, an endpoint that receives stream data might not make forward progress until data that is contiguous with the start of a stream is available. In particular, a receiver might withhold flow control credit until contiguous data is delivered to the application; see Section 2.2 of [QUIC]. To support this receive logic, an endpoint will send stream data until it is acknowledged, ensuring that data at the start of the stream is sent and acknowledged first.

An endpoint that uses a different sending behavior and does not negotiate that change with its peer might encounter performance issues or deadlocks.

4.4. Flow Control Deadlocks

Flow control provides a means of managing access to the limited buffers endpoints have for incoming data. This mechanism limits the amount of data that can be in buffers in endpoints or in transit on the network. However, there are several ways in which limits can produce conditions that can cause a connection to either perform suboptimally or deadlock.

Deadlocks in flow control are possible for any protocol that uses QUIC, though whether they become a problem depends on how implementations consume data and provide flow control credit. Understanding what causes deadlocking might help implementations avoid deadlocks.

Large messages can produce deadlocking if the recipient does not process the message incrementally. If the message is larger than the flow control credit available and the recipient does not release additional flow control credit until the entire message is received and delivered, a deadlock can occur. This is possible even where stream flow control limits are not reached because connection flow control limits can be consumed by other streams.

A common flow control implementation technique is for a receiver to extend credit to the sender as a the data consumer reads data. In this setting, a length-prefixed message format makes it easier for the data consumer to leave data unread in the receiver's buffers and thereby withhold flow control credit. If flow control limits prevent the remainder of a message from being sent, a deadlock will result. A length prefix might also enable the detection of this sort of deadlock. Where protocols have messages that might be processed as a single unit, reserving flow control credit for the entire message atomically makes this style of deadlock less likely.

A data consumer can read all data as it becomes available to cause the receiver to extend flow control credit to the sender and reduce the chances of a deadlock. However, releasing flow control credit might mean that the data consumer might need other means for holding a peer accountable for the state it keeps for partially processed messages.

Deadlocking can also occur if data on different streams is interdependent. Suppose that data on one stream arrives before the data on a second stream on which it depends. A deadlock can occur if the first stream is left unread, preventing the receiver from extending flow control credit for the second stream. To reduce the likelihood of deadlock for interdependent data, the sender should ensure that dependent data is not sent until the data it depends on has been accounted for in both stream- and connection- level flow control credit.

Some deadlocking scenarios might be resolved by cancelling affected streams with STOP_SENDING or RESET_STREAM. Cancelling some streams results in the connection being terminated in some protocols.

5. Packetization and Latency

QUIC exposes an interface that provides multiple streams to the application; however, the application usually cannot control how data transmitted over those streams is mapped into frames or how those frames are bundled into packets.

By default, many implementations will try to maximally pack QUIC packets DATA frames from one or more streams to minimize bandwidth consumption and computational costs (see Section 13 of [QUIC]). If there is not enough data available to fill a packet, an implementation might wait for a short time, to optimize bandwidth efficiency instead of latency. This delay can either be pre-configured or dynamically adjusted based on the observed sending pattern of the application.

If the application requires low latency, with only small chunks of data to send, it may be valuable to indicate to QUIC that all data should be send out immediately. Alternatively, if the application expects to use a specific sending pattern, it can also provide a suggested delay to QUIC for how long to wait before bundle frames into a packet.

Similarly, an application has usually no control about the length of a QUIC packet on the wire. QUIC provides the ability to add a PADDING frame to arbitrarily increase the size of packets. Padding

is used by QUIC to ensure that the path is capable of transferring datagrams of at least a certain size, during the handshake (see Sections 8.1 and 14.1 of [QUIC]) and for path validation after connection migration (see Section 8.2 of [QUIC]) as well as for Datagram Packetization Layer PMTU Discovery (DPLMTUD) (see Section 14.3 of [QUIC]).

Padding can also be used by an application to reduce leakage of information about the data that is sent. A QUIC implementation can expose an interface that allows an application layer to specify how to apply padding.

6. Port Selection and Application Endpoint Discovery

In general, port numbers serve two purposes: "first, they provide a demultiplexing identifier to differentiate transport sessions between the same pair of endpoints, and second, they may also identify the application protocol and associated service to which processes connect" [RFC6335]. The assumption that an application can be identified in the network based on the port number is less true today due to encapsulation, mechanisms for dynamic port assignments, and NATs.

As QUIC is a general-purpose transport protocol, there are no requirements that servers use a particular UDP port for QUIC. For applications with a fallback to TCP that do not already have an alternate mapping to UDP, usually the registration (if necessary) and use of the UDP port number corresponding to the TCP port already registered for the application is appropriate. For example, the default port for HTTP/3 [QUIC-HTTP] is UDP port 443, analogous to HTTP/1.1 or HTTP/2 over TLS over TCP.

Applications could define an alternate endpoint discovery mechanism to allow the usage of ports other than the default. For example, HTTP/3 (Sections 3.2 and 3.3 of [QUIC-HTTP]) specifies the use of HTTP Alternative Services for an HTTP origin to advertise the availability of an equivalent HTTP/3 endpoint on a certain UDP port by using the "h3" ALPN token [RFC7301]. Note that HTTP/3's ALPN token ("h3") identifies not only the version of the application protocol, but also the version of QUIC itself; this approach allows unambiguous agreement between the endpoints on the protocol stack in use.

Given the prevalence of the assumption in network management practice that a port number maps unambiguously to an application, the use of ports that cannot easily be mapped to a registered service name might lead to blocking or other changes to the forwarding behavior by network elements such as firewalls that use the port number for application identification.

7. Connection Migration

QUIC supports connection migration by the client. If an IP address changes, a QUIC endpoint can still associate packets with an existing transport connection using the destination connection ID field (see also Section 9) in the QUIC header, unless a zero-length value is used. This supports cases where address information changes, such as NAT rebinding, intentional change of the local

interface, or based on an indication in the handshake of the server for a preferred address to be used.

Use of a non-zero-length connection ID for the server is strongly recommended if any clients are behind a NAT or could be. A non-zero-length connection ID is also strongly recommended when migration is supported.

Currently QUIC only supports failover cases. Only one "path" can be used at a time, and only when the new path is validated all traffic can be switched over to that new path. Path validation means that the remote endpoint is required to validate the new path before use in order to avoid address spoofing attacks. Path validation takes at least one RTT and congestion control will also be reset after path migration. Therefore migration usually has a performance impact.

QUIC probing packets, which cannot carry application data, can be sent on multiple paths at once. Probing packets can be used to perform address validation, measure path characteristics as input for the switching decision, or prime the congestion controller in preparation for switching to the new path.

Only the client can actively migrate in version 1 of QUIC. However, servers can indicate during the handshake that they prefer to transfer the connection to a different address after the handshake. For instance, this could be used to move from an address that is shared by multiple servers to an address that is unique to the server instance. The server can provide an IPv4 and an IPv6 address in a transport parameter during the TLS handshake and the client can select between the two if both are provided. See also Section 9.6 of [QUIC].

8. Connection Closure

QUIC connections are closed either by expiration of an idle timeout, as determined by transport parameters, or by an explicit indication of the application that a connection should be closed (immediate close). While data could still be received after the immediate close has been initiated by one endpoint (for a limited time period), the expectation is that an immediate close was negotiated at the application layer and therefore no additional data is expected from both sides.

An immediate close will emit an CONNECTION_CLOSE frame. This frame has two sets of types: one for QUIC internal problems that might lead to connection closure, and one for closures initiated by the application. An application using QUIC can define application-specific error codes (see, for example, Section 8.1 of [QUIC-HTTP]).

The CONNECTION_CLOSE frame provides an optional reason field, that can be used to append human-readable information to an error code. RESET_STREAM and STOP_SENDING frames also include an error code, but no reason string.

Alternatively, a QUIC connection can be silently closed by each endpoint separately after an idle timeout. If enabled as indicated by a transport parameter in the handshake, the idle timeout is announced for each endpoint during connection establishment and the

effective value for this connection is the minimum of the two values advertised by client and server. An application therefore should be able to configure its own maximum value as well as have access to the computed minimum value for this connection. An application may adjust the maximum idle timeout for new connections based on the number of open or expected connections, since shorter timeout values may free-up memory more quickly.

If an application desires to keep the connection open for longer than the announced timeout, it can send keep-alive messages; a QUIC implementation may provide an option to defer the time-out by sending keep-alive messages at the transport layer to avoid unnecessary load, as specified in Section 10.1.2 of [QUIC]. See Section 3.2 for further guidance on keep-alives.

9. Information Exposure and the Connection ID

QUIC exposes some information to the network in the unencrypted part of the header, either before the encryption context is established or because the information is intended to be used by the network. QUIC has a long header that exposes some additional information (the version and the source connection ID), while the short header exposes only the destination connection ID. In QUIC version 1, the long header is used during connection establishment, while the short header is used for data transmission in an established connection.

The connection ID can be zero length. Zero length connection IDs can be chosen on each endpoint individually, on any packet except the first packets sent by clients during connection establishment.

An endpoint that selects a zero-length connection ID will receive packets with a zero-length destination connection ID. The endpoint needs to use other information, such as the source and destination IP address and port number to identify which connection is referred to. This could mean that the endpoint is unable to match datagrams to connections successfully if these values change, making the connection effectively unable to survive NAT rebinding or migrate to a new path.

9.1. Server-Generated Connection ID

QUIC supports a server-generated connection ID, transmitted to the client during connection establishment (see Section 7.2 of [QUIC]). Servers behind load balancers may need to change the connection ID during the handshake, encoding the identity of the server or information about its load balancing pool, in order to support stateless load balancing.

Server deployments with load balancers and other routing infrastructure need to ensure that this infrastructure consistently routes packets to the correct server instance. This might require coordination between servers and infrastructure. One method of achieving this involves encoding routing information into the connection ID. This ensures that there is no need to for servers and infrastructure to coordinate routing information for each connection. See [QUIC-LB] for more information.

9.2. Mitigating Timing Linkability with Connection ID Migration

QUIC requires that endpoints generate fresh connection IDs for use on new network paths. Choosing values that are unlinkable to an outside observer ensures that activity on different paths cannot be trivially correlated using the connection ID.

While sufficiently robust connection ID generation schemes will mitigate linkability issues, they do not provide full protection. Analysis of the lifetimes of six-tuples (source and destination addresses as well as the migrated CID) may expose these links anyway.

In the limit where connection migration in a server pool is rare, it is trivial for an observer to associate two connection IDs. Conversely, in the opposite limit where every server handles multiple simultaneous migrations, even an exposed server mapping may be insufficient information.

The most efficient mitigations for these attacks are through network design and/or operational practice, by using a load balancing architecture that loads more flows onto a single server-side address, by coordinating the timing of migrations in an attempt to increase the number of simultaneous migrations at a given time, or through other means.

9.3. Using Server Retry for Redirection

QUIC provides a Server Retry packet that can be sent by a server in response to the Client Initial packet. The server may choose a new connection ID in that packet and the client will retry by sending another Client Initial packet with the server-selected connection ID. This mechanism can be used to redirect a connection to a different server, e.g. due to performance reasons or when servers in a server pool are upgraded gradually, and therefore may support different versions of QUIC. In this case, it is assumed that all servers belonging to a certain pool are served in cooperation with load balancers that forward the traffic based on the connection ID. A server can choose the connection ID in the Server Retry packet such that the load balancer will redirect the next Client Initial packet to a different server in that pool. Alternatively the load balancer can directly offer a Retry service as further described in [QUIC-LB].

Section 4 of [RFC5077] describes an example approach for constructing TLS resumption tickets that can be also applied for validation tokens, however, the use of more modern cryptographic algorithms is highly recommended.

10. Quality of Service (QoS) and DSCP

QUIC assumes that all packets of a QUIC connection, or at least with the same 5-tuple {dest addr, source addr, protocol, dest port, source port}, will receive similar network treatment since feedback about loss or delay of each packet is used as input to the congestion controller. Therefore it is not recommended to use different DiffServ Code Points (DSCPs) [RFC2475] for packets belonging to the same connection. If differential network treatment,

e.g. by the use of different DSCPs, is desired, multiple QUIC connections to the same server may be used. However, in general it is recommended to minimize the number of QUIC connections to the same server, to avoid increased overheads and, more importantly, competing congestion control.

11. Use of Versions and Cryptographic Handshake

Versioning in QUIC may change the protocol's behavior completely, except for the meaning of a few header fields that have been declared to be invariant [QUIC-INVARIANTS]. A version of QUIC with a higher version number will not necessarily provide a better service, but might simply provide a different feature set. As such, an application needs to be able to select which versions of QUIC it wants to use.

A new version could use an encryption scheme other than TLS 1.3 or higher. [QUIC] specifies requirements for the cryptographic handshake as currently realized by TLS 1.3 and described in a separate specification [QUIC-TLS]. This split is performed to enable light-weight versioning with different cryptographic handshakes.

12. Enabling New Versions

QUIC provides integrity protection for its version negotiation process. This process assumes that the set of versions that a server supports is fixed. This complicates the process for deploying new QUIC versions or disabling old versions when servers operate in clusters.

A server that rolls out a new version of QUIC can do so in three stages. Each stage is completed across all server instances before moving to the next stage.

In the first stage of deployment, all server instances start accepting new connections with the new version. The new version can be enabled progressively across a deployment, which allows for selective testing. This is especially useful when the new version is compatible with an old version, because the new version is more likely to be used.

While enabling the new version, servers do not advertise the new version in any Version Negotiation packets they send. This prevents clients that receive a Version Negotiation packet from attempting to connect to server instances that might not have the new version enabled.

During the initial deployment, some clients will have received Version Negotiation packets that indicate that the server does not support the new version. Other clients might have successfully connected with the new version and so will believe that the server supports the new version. Therefore, servers need to allow for this ambiguity when validating the negotiated version.

The second stage of deployment commences once all server instances are able accept new connections with the new version. At this point, all servers can start sending the new version in Version Negotiation packets.

During the second stage, the server still allows for the possibility that some clients believe the new version to be available and some do not. This state will persist only for as long as any Version Negotiation packets take to be transmitted and responded to. So the third stage can follow after a relatively short delay.

The third stage completes the process by enabling authentication of the negotiated version with the assumption that the new version is fully available.

The process for disabling an old version or rolling back the introduction of a new version uses the same process in reverse. Servers disable validation of the old version, stop sending the old version in Version Negotiation packets, then the old version is no longer accepted.

13. IANA Considerations

This document has no actions for IANA; however, note that Section 6 recommends that application bindings to QUIC for applications using TCP register UDP ports analogous to their existing TCP registrations.

14. Security Considerations

See the security considerations in [QUIC] and [QUIC-TLS]; the security considerations for the underlying transport protocol are relevant for applications using QUIC, as well. Considerations on linkability, replay attacks, and randomness discussed in [QUIC-TLS] should be taken into account when deploying and using QUIC.

Application developers should note that any fallback they use when QUIC cannot be used due to network blocking of UDP should guarantee the same security properties as QUIC; if this is not possible, the connection should fail to allow the application to explicitly handle fallback to a less-secure alternative. See Section 2.

Further, [QUIC-HTTP] provides security considerations specific to HTTP. However, discussions such as on cross-protocol attacks, traffic analysis and padding, or migration might be relevant for other applications using QUIC as well.

15. Contributors

Igor Lubashev contributed text to Section 9 on server-selected connection IDs.

16. Acknowledgments

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.

17. References

17.1. Normative References

[QUIC] 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://tools.ietf.org/html/draft-ietf-quic-transport-34>.

[QUIC-INVARIANTS] Thomson, M., "Version-Independent Properties of QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-invariants-13, 14 January 2021, <https://tools.ietf.org/html/draft-ietf-quic-invariants-13>.

[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://tools.ietf.org/html/draft-ietf-quic-tls-34>.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <https://www.rfc-editor.org/rfc/rfc6335>.

[TLS13] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/rfc/rfc8446>.

17.2. Informative References

[Edeline16] Edeline, K., Kuehlewind, M., Trammell, B., Aben, E., and B. Donnet, "Using UDP for Internet Transport Evolution (arXiv preprint 1612.07816)", 22 December 2016, <https://arxiv.org/abs/1612.07816>.

[Hatonen10] Hatonen, S., Nyrhinen, A., Eggert, L., Strowes, S., Sarolahti, P., and M. Kojo, "An experimental study of home gateway characteristics (Proc. ACM IMC 2010)", October 2010.

[HTTP-REPLAY] Thomson, M., Nottingham, M., and W. Tarreau, "Using Early Data in HTTP", RFC 8470, DOI 10.17487/RFC8470, September 2018, <https://www.rfc-editor.org/rfc/rfc8470>.

[I-D.ietf-taps-arch] Pauly, T., Trammell, B., Brunstrom, A., Fairhurst, G., Perkins, C., Tiesel, P. S., and C. A. Wood, "An Architecture for Transport Services", Work in Progress, Internet-Draft, draft-ietf-taps-arch-09, 2 November 2020, <https://tools.ietf.org/html/draft-ietf-taps-arch-09>.

[I-D.nottingham-httpbis-retry] Nottingham, M., "Retrying HTTP Requests", Work in Progress, Internet-Draft, draft-nottingham-httpbis-

retry-01, 1 February 2017, <https://tools.ietf.org/html/draft-nottingham-httpbis-retry-01>.

[PaaschNanog] Paasch, C., "Network Support for TCP Fast Open (NANOG 67 presentation)", 13 June 2016, <https://www.nanog.org/sites/default/files/Paasch_Network_Support.pdf>.

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

[QUIC-LB] Duke, M. and N. Banks, "QUIC-LB: Generating Routable QUIC Connection IDs", Work in Progress, Internet-Draft, draft-ietf-quic-load-balancers-06, 4 February 2021, <https://tools.ietf.org/html/draft-ietf-quic-load-balancers-06>.

[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>.

[RFC5077] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, DOI 10.17487/RFC5077, January 2008, <https://www.rfc-editor.org/rfc/rfc5077>.

[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>.

[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>.

[RFC8085] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <https://www.rfc-editor.org/rfc/rfc8085>.

[Swett16] Swett, I., "QUIC Deployment Experience at Google (IETF96 QUIC BoF presentation)", 20 July 2016, <https://www.ietf.org/proceedings/96/slides/slides-96-quic-3.pdf>.

[Trammell16] Trammell, B. and M. Kuehlewind, "Internet Path Transparency Measurements using RIPE Atlas (RIPE72 MAT presentation)", 25 May 2016, <https://ripe72.ripe.net/wp-content/uploads/presentations/86-atlas-udpdiff.pdf>.

Authors' Addresses

Mirja Kuehlewind
Ericsson

Email: mirja.kuehlewind@ericsson.com

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

Email: ietf@trammell.ch