QUIC Working Group
Internet-Draft
Intended status: Informational
Expires: December 1, 2020

M. Boucadair, Ed.
Orange
O. Bonaventure, Ed.
M. Piraux, Ed.
Q. De Coninck
UCLouvain
S. Dawkins, Ed.
Tencent America
M. Kuehlewind, Ed.
Ericsson
M. Amend
Deutsche Telekom
A. Kassler
Karlstad University
Q. An
Alibaba Group
N. Keukeleire
Tessares
S. Seo
Korea Telecom
May 30, 2020

3GPP Access Traffic Steering Switching and Splitting (ATSSS) - Overview for IETF Participants
draft-bonaventure-quic-atsss-overview-00

Abstract

This document briefly presents the Access Traffic Steering, Switching, and Splitting (ATSSS) service being specified within the 3rd Generation Partnership Project (3GPP). The ATSSS service provides network support for multihomed devices to select a path for transmission (steer), move traffic from one path to another (switch), or use multiple paths simultaneously (split). TS 23.501 specifies an ATSSS architecture for TCP traffic.

This document presents a snap-shot of the ongoing discussion in the 3GPP to enable ATSSS for non-TCP traffic, based on the use of QUIC, and assesses to what extent IETF specifications can be used to meet the ATSSS design goals. Apparent gaps are also documented.

Copyright Notice

Table of Contents

1. Introduction

The 3rd Generation Partnership Project (3GPP) has described the Access Traffic Steering, Switching, and Splitting (ATSSS) service, used to carry traffic over multiple available paths. In Release 16 [TS23501], ATSSS supports TCP traffic relying upon two IETF protocols: MPTCP [RFC6824] and Convert Protocol [I-D.ietf-tcpm-converters].

As part of preparation for Release 17 studies, 3GPP has expressed an interest in other IETF protocols and protocol extensions that would enable ATSSS service of traffic not supported by the Convert Protocol nor based on the use of MPTCP. To that aim, 3GPP has contacted the IETF through a formal liaison [atsssliaison], letting the IETF know about this interest. An excerpt of the liaison document is provided below:

"The work on the study has not yet started in 3GPP, and there are thus no agreed conclusions. The goal is to enable steering, switching and splitting of traffic (primarily UDP) across multiple accesses, including latency sensitive and real time traffic. Therefore 3GPP is interested to receive regular feedback on progress and prioritization on the multipath extensions to QUIC."

Because "3GPP SA kindly requests IETF to take the above information into account when discussing future work in prioritizing the multipath work for QUIC", but the complete specification of ATSSS in the relevant 3GPP architecture documents reflects the complexity of 3GPP networks, the authors of this document worked to provide a high level overview of the parts of ATSSS that would be impacted by IETF

protocol design, in terminology that is more familiar to IETF participants.

1.1. Notes for Readers

We provide a high-level overview of ATSSS in Section 5, describe the current ATSSS version in Section 7, describe our thoughts about a QUIC-based version of ATSSS in Section 7.1, and conclude with our understanding of the gaps between QUIC version 1 and what a QUIC-based version of ATSSS will require in Section 8.

This document is an informational Internet-Draft from individuals, and does not carry any special status within the IETF.

This document abstracts considerable architectural detail that is available in 3GPP specifications. The goal is to make this overview more accessible for IETF readers who are not familiar with 3GPP 5G architecture.

This document makes references to Internet-Drafts that are not as mature as the core QUIC Internet-Drafts, and in some cases, have not been adopted as Working Group drafts yet, although all are within existing and proposed IETF working group charters. The goal is to give 3GPP readers the most up-to-date understanding of what is possible.

2. Introduction to Access Traffic Steering, Switching, and Splitting (ATSSS)

Mobile devices such as laptops, smartphones, tablets support multiple network interfaces that may attach to different networks. Over the years, various techniques have been proposed to support such multi-interfaced devices (e.g., Shim6 [RFC5533], Mobile IPv6 [RFC6275], Proxy Mobile IPv6 [RFC5213], or Multipath TCP [RFC6824]).

Users of these devices have different expectations concerning the utilization of available network connectivity (and thus their different network interfaces). For simplicity, we consider a smartphone that is equipped with a Wireless LAN (WLAN) interface and a cellular interface, but the discussion below can be generalized to any device with multiple network interfaces that support IP.

Some users of these smartphones want to offload most or all of their traffic onto the WLAN when the WLAN is available while expecting seamless handovers when it is not available. For example, when they move out of the reach of their home WLAN, they expect that the established flows (e.g., TCP connections, UDP flows) will continue over the cellular interface without any interruption (called "session

continuity" in 3GPP). As the devices are assigned different IP addresses over WLAN and the cellular networks, this seamless handover requires some specific assistance from the network. The current utilization of Multipath TCP on Apple smartphones is an example of this use case [IETFJ16].

Other users want to load balance their flows over the different available networks, e.g., by sending a delay-sensitive flow over cellular and a long download over the WLAN network. Several smartphones enable applications to indicate their preferences when using available networks. This steering policy can be managed by the smartphone, but flows need to continue after a handover.

Still other users may want to combine the resources provided by the cellular and the WLAN networks to improve the up and download throughput performance of individual flows. The GiGA LTE and GiGA 5G services deployed using Multipath TCP in South Korea are examples of this use case [IETFJ16].

To support these different use cases in 5G networks, 3GPP is defining the Access Traffic Steering, Switching and Splitting (ATSSS) service [TS23501]. This work is further adopted by the Broadband Forum to provide similar capabilities to residential gateways equipped with multiple access interfaces, in the continuity of the Hybrid Access Networks [TR-348].

In this document, we abstract many of the technical details of future 5G networks to explain the capabilities ATSSS needs, which may impact decisions about future work on IETF protocols.

3. Contribution and Discussion Venues for this draft.

(Note to RFC Editor - if this document ever reaches you, please remove this section)

This document is under development in the Github repository at https://github.com/obonaventure/draft-quic-atsss-reqs. Readers are invited to open issues and send pull requests with contributed text for this document.

Substantial discussion of this document should take place on the QUIC working group mailing list (quic@ietf.org). Subscription and archive details are at https://www.ietf.org/mailman/listinfo/quic.

4. Conventions, Terminology, and Definitions

This document makes use of 3GPP specific terms defined in [RFC6459], mainly the following ones:

- Packet Data Network (PDN): is a packet-based network that either belongs to the operator or is external (such as the Internet or a corporate intranet). The user eventually accesses services in one or more PDNs. The operator's packet core networks are separated from packet data networks by User Plane Functions (UPFs).

- UE (User Equipment): refers to the devices that are hosts with the ability to obtain Internet connectivity via a 3GPP network.

- User Plane: refers to data traffic and the required sessions for the data traffic. In practice, IP is the only data traffic protocol used in the user plane.

Also, the document uses the following additional terms:

- Protocol Data Unit (PDU) Session: An association between the UE and the Data Network (DN) to carry the user data/traffic.

- PDU Connectivity Service: A service that provides exchange of PDUs between an UE and a Data Network.

- Multi-access PDU (MA-PDU) Session: A PDU session that has simultaneously user plane resources assigned on 3GPP and non-3GPP access networks.

- User Plane Function (UPF): A logical function in the 5G core network that provides the interconnect point between the mobile infrastructure and the Data Network (DN) and anchor point for Protocol Data Unit (PDU) Sessions to enable mobility.

- Data Network Name (DNN): is a Fully Qualified Domain Name (FQDN) and resolves to a set of gateways in an operator's network. DNN is used for the selection of the UPF(s) for a PDU Session.

- 5G Core (5GC) network: Refers to the part of the 5G System which is independent of the access technology used by an UE (e.g., cellular, WLAN) [TS23501]. A 5G Core network can be reached via one or more access networks.

- 3GPP access network: Refers to a radio access network used by an UE to reach a 5G Core network. In such case, the UE uses an access technology that is specified by 3GPP.

- Non-3GPP access network: Refers to an access network (e.g., WLAN) that is not a 3GPP access network and which is used by an UE to connect to a 5G Core network.

- 5G control plane: Denotes the 5G control management component of use plane resources (e.g., forwarding policies).

5. High Level ATSSS Overview

The 5G Core supports a service that provides exchange of data between a User Equipment (UE) and a data network (referred to as Packet Data Network (PDN)) identified by a Data Network Name (DNN). This connectivity service, called the Protocol Data Unit (PDU) Connectivity Service, is realized via 'PDU Sessions' that are established upon request from User Equipment (UE) when the UE first connects to that network. The type of PDU Session can be IPv4, IPv6, IPv4v6, Ethernet or Unstructured.

It is out of the scope of this document to provide a comprehensive overview of 5G System (5GS) architecture. In particular, this document does not describe how PDU Sessions are established, and thus how IP addresses/prefixes are assigned to requesting UEs.

An UE can be provided a multi-access PDU Connectivity Service. That is, an UE can exchange data with a PDN by using a "3GPP access network" and a "non-3GPP access network" (often a WLAN). This is realized using the ATSSS service that is provided in the 5G core network control plane and user plane. The user plane part of the ATSSS functionality is contained in the User Plane Function (UPF) that manages the UE's PDU session.

5.1. Reference Architecture

To understand the operation of the ATSSS service, it is useful to consider the reference environment shown in Figure 1. An UE is attached to two different access networks (Access Net A and Access Net B). Each of these two networks is potentially shared with other users, so the bandwidth available from each network varies over time. These fluctuations in bandwidth are managed by using congestion control schemes.

One of these access networks is managed by a 5G provider according to the 3GPP specifications. The second network is potentially managed by a different organization. It is important to note that in this second case, there is an IPSec tunnel between the UE and a dedicated device in the 5G network (not shown in Figure 1). A dedicated IP address is assigned by means of Internet Key Exchange version 2 (IKEv2) to the UE to access the 5G Core via this second network.

The UE interacts with a distant server through a User Plane Function hosting the ATSSS functionality. This UPF is called hereafter ATSSS UPF. The ATSSS UPF enables the UE to use a transport protocol that supports the different access networks even if the server does not support it (i.e., does not use a multipath-capable transport protocol).

The 5G control plane provides the UE and the ATSSS UPF with rules as discussed in Section 5.4. Note that, as per 3GPP definition, the rules provided to the UE are called ATSSS Rules, while the ATSSS information provided to the UPF is carried via Multi-Access Rules, but for simplicity this document uses the term "ATSSS rules" for the ATSSS information provided to both UE and UPF.

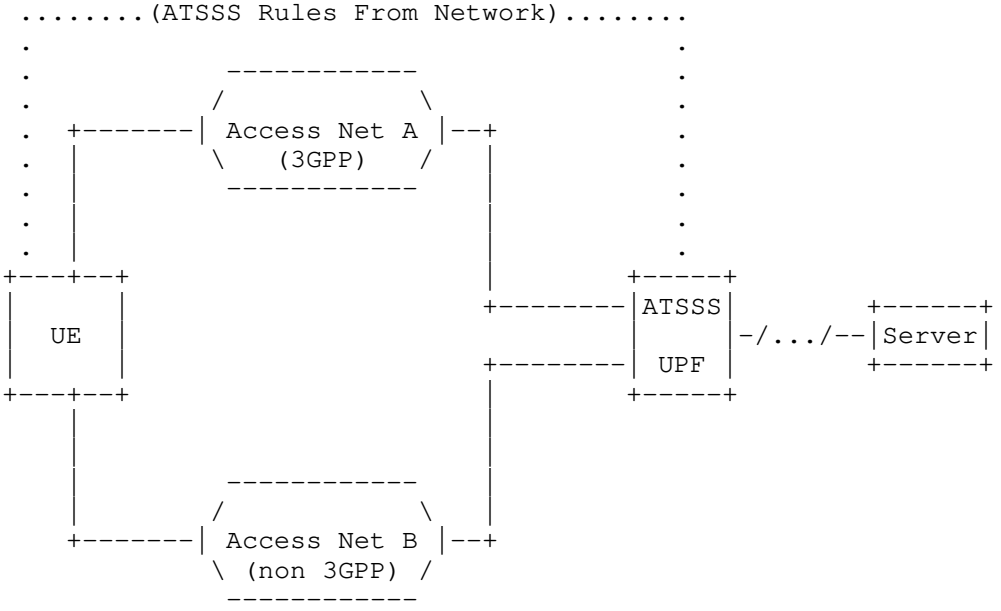


Figure 1: Simplified Reference Architecture for ATSSS

5.2. External IP Addresses Used by the ATSSS UPF

Depending on the ATSSS mode, the ATSSS UPF may translate the access network specific addresses into an address called the Multi-Access (MA) IP address and vice versa. Such an address is also directly assigned to the UE (that is, packets that are not eligible for the ATSSS service are sourced by the UE with that IP address).

Absent any coordination between the UE and the ATSSS UPF (e.g., assignment of port ranges), the same IP address and port number could

be used simultaneously by the ATSSS UPF and the UE; which is problematic.

To avoid such issue, the ATSSS UPF may be configured with a pool of IP addresses that it can use in the Internet-facing interfaces instead of preserving the MA IP address assigned to the UE. How that pool is used is deployment- and implementation-specific.

5.3. ATSSS Modes

3GPP defines the following procedures [TS23501] that are applicable between "3GPP access" and "non-3GPP access" networks:

- Access Traffic Steering: selection of an access network for a new data flow and the transfer of the traffic of that data flow over the selected access network.

- Access Traffic Switching: migration of all packets of an ongoing data flow from one access network to another access network. Only one access network is in use at a time, but this still ensures session continuity.

- Access Traffic Splitting: forwarding the packets of a data flow across multiple access networks simultaneously.

Techniques to provide ATSSS are classified by the 3GPP into two flavors: (1) higher-layer techniques which operate above the IP layer (e.g., MPTCP), and (2) lower-layer techniques which operate below the IP layer.

5.4. ATSSS Rules

The 5G control plane provides the UE and the ATSSS UPF with rules that specify which flows are eligible to the ATSSS service (i.e., by mapping them to a Multi-Access PDU Session). Once a Multi-Access PDU Session has been established, a set of rules are then delivered to both the UE and the ATSSS UPF in order to enable consistent treatment of the flows by both the UE and the ATSSS UPF within the Session.

The traffic that matches an ATSSS rule can be distributed among the available access networks following one of these modes:

- "Active-Standby": The traffic associated with the matching flow will be forwarded via a specific access (called 'active access') and switched to another access (called 'standby access') when the active access is unavailable.

- "Smallest Delay": The traffic associated with the matching flow will be forwarded via the access that presents the smallest RTT. To that aim, specific measurements are conducted by the UE and a dedicated function co-located with the ATSSS UPF.

- "Load-Balancing": The traffic associated with the matching flow will be distributed among the available access networks following a distribution ratio (e.g., 30% via a first access, 70% via a second access).

- "Priority-based": For this mode, accesses are assigned priority levels that indicate which access to be used first. Concretely, the traffic associated with the matching flow will be steered on the access with a high priority till congestion is detected, then the overflow will be forwarded over a low priority access.

In order to provide the above-mentioned steering modes, measurement information about the current network state of each path is needed. Often if a multipath capable protocol is used, the measurements are available as part of the protocol itself. For ATSSS approaches where this is not the case, a dedicated protocol called Performance Measurement Function (PMF) protocol can be used. This protocol is enabled between the UE and the UPF in order to provide RTT measurements and report access availability/unavailability by the UE to the UPF.

These modes of operations can be met by using multipath protocols such as MPTCP [RFC6824] to select the different paths between the UE and the ATSSS. Such protocols usually include two types of mechanisms to control the utilization of the paths: (i) a path manager and (ii) a packet scheduler [RFC8041]. The path manager decides when a new subflow needs to be established over a path while the packet scheduler selects the subflow over which the next packet will be sent. A more detailed description of several packet schedulers may be found in [I-D.bonaventure-iccrg-schedulers].

The "Active-Standby" mode can be implemented using a path manager that tries to use the active access and switches to the standby one after a certain number of retransmissions. This mode of operation is similar to the utilization of Multipath TCP on iOS smartphones [RFC8041].

The "Smallest Delay" mode can be implemented using a path manager that establishes a subflow over both paths and a packet scheduler that measures their RTTs and prefers the one having the lowest RTT. These path manager and scheduler are similar to those used by Multipath TCP on Linux [RFC8041].

The "Load-Balancing" mode would use the same path manager with a weighted round-robin scheduler.

The "Priority-based" mode can be implemented using a path manager that is similar to the one used by the "Active-Standby" one, but which reacts faster and a packet scheduler that prefers the high priority path. These path manager and scheduler are similar to the ones used in deployed Hybrid Access networks [Hybrid].

6. ATSSS Phases

We first describe in Section 7 ATSSS as specified in Release 16 (called hereafter, ATSSS Phase 1) that uses Multipath TCP [RFC6824] and the 0-RTT Convert Protocol [I-D.ietf-tcpm-converters] to handle TCP traffic. We then discuss in Section 7.1 the data plane requirements for Phase 2 of the ATSSS specification that 3GPP plans for Release 17. More details about 3GPP releases can be found at: https://www.3gpp.org/specifications/Releases.

7. ATSSS Phase 1: Support for TCP

For ATSSS with Multipath TCP functionality, a client with two interfaces connected to two disjoint access networks (in this case, Access Net A and Access Net B) uses MPTCP to reach an MPTCP proxy over either, or both, of the access networks. This allows the client to communicate with a server which does not support MPTCP.

During the attachment of an ATSSS-capable UE to the network, the UE may retrieve the MPTCP proxy information: an IP address, a port number, and the type of proxy. In the current release, the mandatory MPTCP proxy type is the "Transport Converter" [I-D.ietf-tcpm-converters].

Also, both the MPTCP Client and MPTCP proxy are configured with ATSSS rules from the network that govern how the multiple network paths between the MPTCP Client and MPTCP proxy are used. This relationship is shown using "." between the MPTCP Client and MPTCP Proxy in Figure 2.

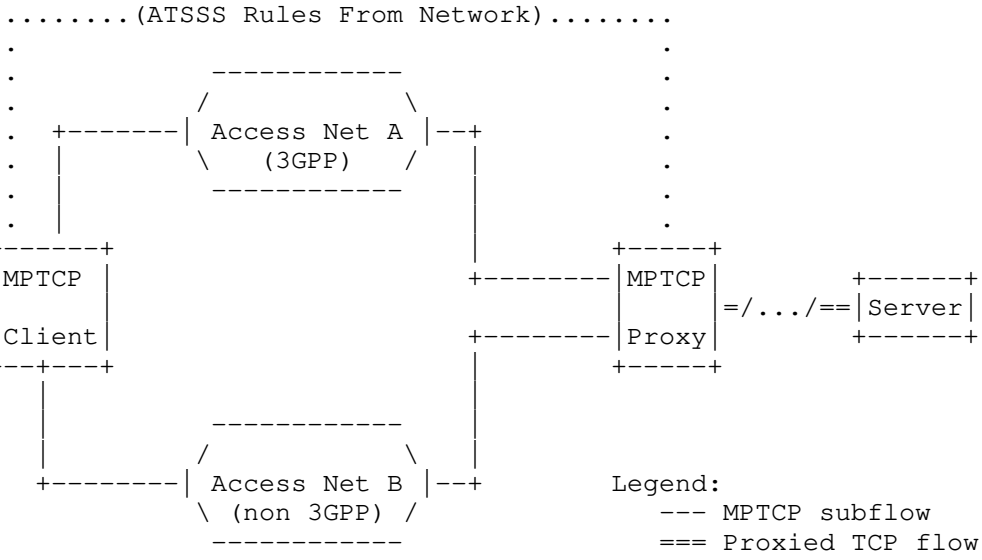
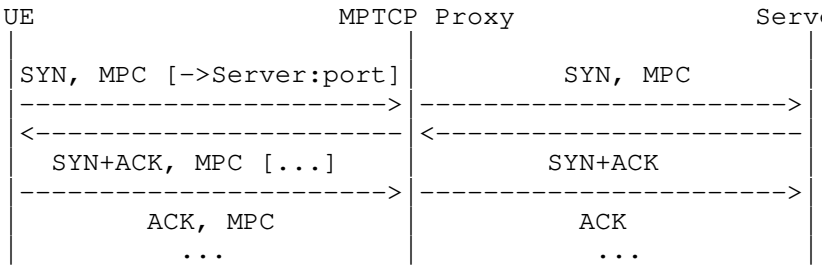


Figure 2: Simplified Reference Architecture for ATSSS with Multipath TCP

An ATSSS-capable UE can make use of the MPTCP functionality by establishing MPTCP-assisted connections via the MPTCP proxy relying upon the Convert Protocol [I-D.ietf-tcpm-converters]. The UE behaves as a "Client", while the MPTCP proxy behaves as a Transport Converter [I-D.ietf-tcpm-converters].

The UE then sends packets bound to connections matching an ATSSS rule to the provisioned Transport Converter and destination port number. Concretely, the UE initiates the MPTCP connection towards the Transport Converter and indicates the IP address and port number of the Server within the connection establishment packet (i.e., in the payload of the SYN sent to the Transport Converter). Doing so enables the Transport Converter to immediately initiate a connection towards that Server, without experiencing an extra delay. The Transport Converter waits until the receipt of the confirmation that the Server agrees to establish the connection before confirming it to the Client.

A flow example of an MPTCP-proxied connection is shown in Figure 3. This example assumes that the Server is not MPTCP-aware. The instructions included in the matching ATSSS rule will be followed for the management of the MPTCP connection (including the selection of the access network to establish the first subflow).



Legend:
[]: Convert Protocol TLVs
MPC: MP_CAPABLE option [RFC6824]

Figure 3: An Example of MPTCP-proxied Connection Matching an ATSSS Rule.

This approach provides 0-RTT (Zero Round-Trip Time) conversion service since no extra delay is induced by the Convert protocol compared to connections that are not proxied. Also, the Convert Protocol does not require any encapsulation (so, no tunnels). The UE and the MPTCP proxy track the performance of the access networks by leveraging MPTCP's internal mechanisms including congestion control and round-trip-time measurements. MPTCP uses this performance information to support splitting and switching.

If the server supports MPTCP, the Convert Protocol provides an option for clients to "opt-out". In such case, an MPTCP connection is directly established between the client and the server. Given that few servers are MPTCP capable, relying on the ATSSS service is the only option for UEs to make use of available multiple paths to most servers simultaneously.

7.1. ATSSS Phase 2: Adding Non-TCP Support

The MPTCP-based ATSSS approach discussed in the previous section is specific to TCP, obviously. Therefore it does not support non-TCP traffic, such as UDP, QUIC [QUIC-Deployment] or IPsec and Datagram Transport Layer Security (DTLS) Virtual Private Network (VPN) services.

As the share of these protocols grows, mainly driven by QUIC deployment, a future ATSSS needs to extend beyond supporting TCP only. The seamless handover provided by ATSSS is particularly useful for real-time traffic (e.g., voice or video calls),

Several proposals to carry non-TCP traffic have been discussed, including using TCP [I-D.boucadair-mptcp-plain-mode] or defining

multipath extensions to DCCP [I-D.amend-tsvwg-multipath-dccp]. The work within 3GPP now focuses on using QUIC as the baseline for ATSSS Phase 2.

7.1.1. QUIC and Multihoming

For non-TCP traffic, QUIC is already the dominant part of UDP traffic. A solution as realized with the Convert Protocol for (MP)TCP is not possible for QUIC as a QUIC connection cannot be intercepted and converted as in the ATSSS architecture for (MP)TCP (Figure 2). For (MP)TCP only the transport protocol (TCP) is intercepted, while transport security provided by Transport Layer Security (TLS) on top of TCP stays in tact. For QUIC transport, security is integrated into the transport protocol and thus cannot be intercepted.

Some ATSSS modes can be natively supported by the base QUIC specification for QUIC flows. For example, the "Active-Standby" and "Smallest Delay" steering modes can be supported directly between an UE and a QUIC server without any assistance from the network other than the performance measurement information.

Further, QUIC provides a feature called connection migration (Section 9 of [I-D.ietf-quic-transport]) that makes it possible to move a QUIC connection from one path/IP address to another without terminating and reestablishing the connection. Connection migration can further enable traffic switching but does not support traffic splitting as only one path can be used simultaneously.

An extension to QUIC to support simultaneous use of multiple paths is proposed in [I-D.deconinck-quic-multipath]. However, similar to a native MPTCP connection, a (MP)QUIC connection initiated between the UE and a server without the ATSSS UPF assistance cannot benefit from any direct application of the ATSSS steering methods based on network input given that the steering policy as currently defined in ATSSS is local to the UE and the ATSSS UPF and there are no means to signal that policy to a remote server.

Network input can be especially beneficial for cases such as:

- avoiding unnecessary use of user quota if one of the access networks is subject to volume-based quota.

- avoiding frequent connection migration if both access networks could be used to forward packets (each with a distinct source IP address).

7.1.2. QUIC as an ATSSS Data Plane Protocol

This section elaborates how non-TCP traffic (UDP or a subset like QUIC) can be encapsulated into a tunnel between the UE and the ATSSS UPF to enable ATSSS for all traffic, not only TCP or QUIC. This section discusses to what extent QUIC can be used as a tunneling protocol for the ATSSS service and whether gaps are found.

When tunneling non-TCP (e.g., UDP, IP) over QUIC the Unreliable Datagram Extension [I-D.ietf-quic-datagram] can be used. Each data packet would be transported unreliably as a datagram over a QUIC connection. Transporting these datagrams unreliably as would be done in IPsec or DTLS-based tunnels is especially important for flows that do not require reliable delivery and would suffer from unnecessary delays caused by the retransmissions used to support reliability. QUIC datagrams are congestion-controlled, but since the latency between the UE and the ATSSS UPF is small compared to the end-to-end latency, having second, local, congestion control loops should not impact the end-to-end congestion control negatively.

This document discusses three approaches:

- Use of QUIC version 1 with the Unreliable Datagram Extension Section 7.1.3
- Use of QUIC version 1 with the Unreliable Datagram Extension but with one QUIC connection over each access network Section 7.1.4
- Use of a single Multipath QUIC connection [I-D.deconinck-quic-multipath], with the Unreliable Datagram Extension, over all access networks Section 7.1.5.

7.1.3. Single QUICv1 Tunnel with Unreliable Datagram Extension and Connection Migration

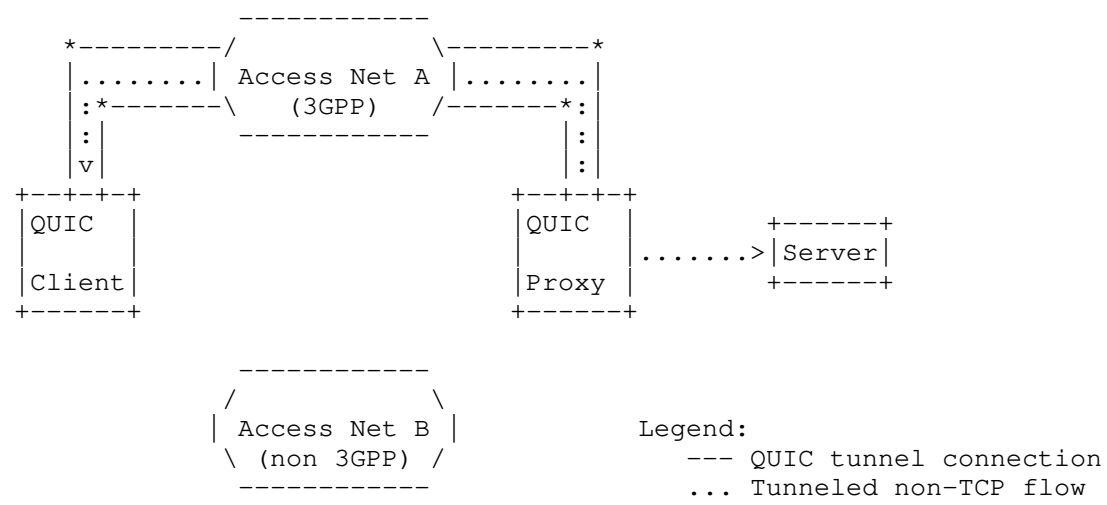


Figure 4: Single QUICv1 tunnel

QUIC can be used as a tunneling protocol between the UE and the ATSSS UPF. Use of the Unreliable Datagram Extension avoids unnecessary delays due to local retransmissions and, more important, subsequent head-of-line blocking.

In this case, there is (only) one QUIC connection between the UE and the ATSSS UPF for a given flow. And as such, that QUIC connection uses only one access network at a time. However, given the connection migration capability of QUIC, the QUIC connection could be moved to another access network, e.g., when the network indicates that the currently used access network would go away or that its capacity becomes limited. The UE can then initiate the connection migration to start the path validation process as specified in Section 9 of [I-D.ietf-quic-transport]. When, and how, to switch over depends on the rules provided by the network (Section 5.4) and the performance measurements that are accessible to both the UE and the ATSSS UPF. Note that connection migration can only be at the initiative of the UE as per Section 9 of [I-D.ietf-quic-transport]. This means that the UPF cannot make use of a second access network upon failure or degradation observed on a first access network.

It is thus possible to support the switching and steering functions of ATSSS, but splitting cannot be supported.

Path validation induces a delay when switching as packets are buffered. Further, congestion control state is reset on a new path and needs to ramp up. When frequent handovers happen, splitting traffic over multiple paths simulaneously can be beneficial for a

smooth user experience. Further, of course, the ability to use multiple paths simultaneously also increases the maximum capacity available, which can also be beneficial in some cases.

This option is not considered a valid approach for the ATSSS Phase 2 discussed within 3GPP.

7.1.4. Multiple QUICv1 Tunnels with Unreliable Datagram Extension and Connection Migration

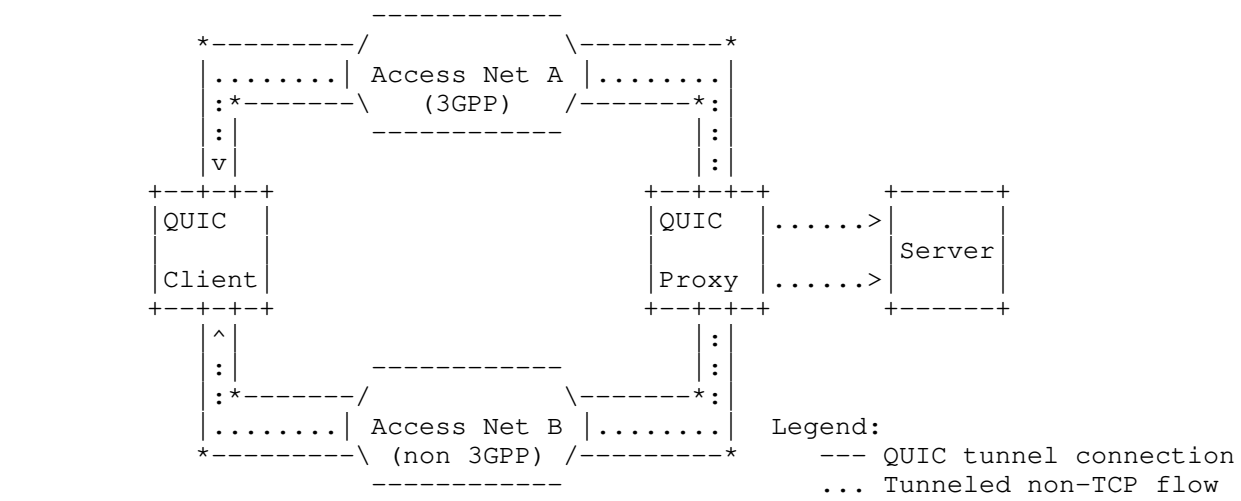


Figure 5: Traffic steering of two end-to-end flows using multiple QUICv1 tunnels

Another approach is to use one QUIC connection over each access network. In this case, there are multiple QUIC connections that are used as tunnels from the UE to the ATSSS UPF to transport traffic that belongs to one or multiple flows. The UE and ATSSS UPF need to select one QUIC connection to forward each application data packet, based on the rules provided by the network.

This approach can support steering (by selecting just one QUIC connection for each flow), switching (by moving flows from one QUIC connection to another) as well as splitting when both tunnels are used simultaneously for different packets of the same flow.

However, this approach for splitting is challenging since data from the same flow are sent over different QUIC connections, again using unreliable datagram to minimize head-of-line blocking. Because both these QUIC connections are completely independent of each other and the paths on the different access networks may have different

latency, this approach would likely result in reordering of packets that belong to a split flow. If reordering should be avoided, this would require additional signaling between the UE and the ATSSS UPF, e.g., adding sequence numbers, and adding a reordering buffer and logic to both the UE and ATSSS UPF.

Furthermore, since the bandwidth available for each QUIC connection varies as a function of the congestion experienced over each access network, data sent over a congested connection could delay the delivery of subsequent data over another connection.

Experience with MPTCP on smartphones shows that its integrated mechanisms (e.g., congestion control, round-trip-time estimation, packet scheduler) are well suited to support splitting and switching. Providing a similar service over independent QUIC connections would require a complex application that would need to track the congestion window, round-trip-time, and loss characteristics of the underlying QUIC connections as well as some specific application layer signalling to glue the various QUIC connections together.

7.1.5. MP-QUIC Tunneling

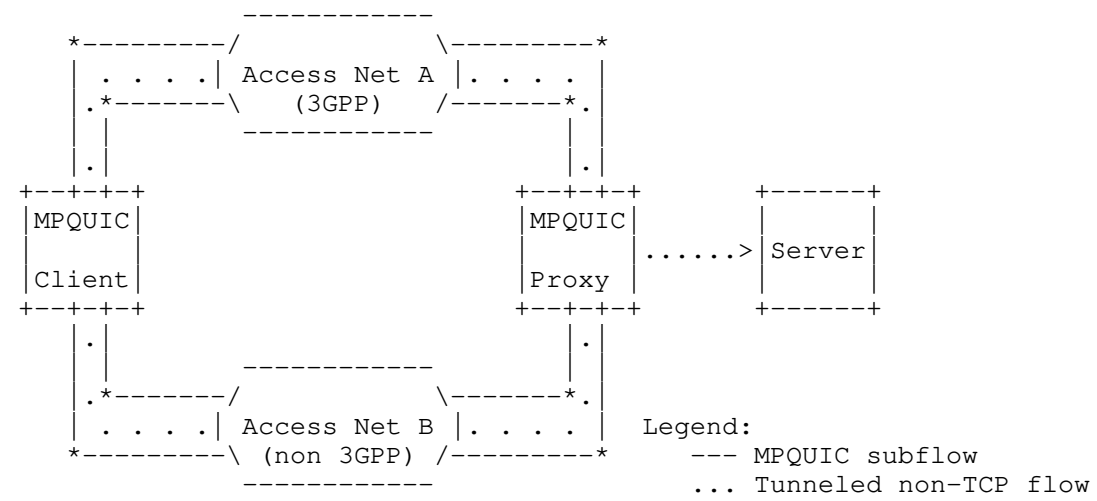


Figure 6: Traffic splitting using a Multipath QUIC tunnel

A third candidate solution is to leverage the ability of QUIC to support multiple streams and the Unreliable Datagram Extension, and to extend it with Multipath capabilities as described in [I-D.deconinck-quic-multipath].

In this case, there is a single (Multipath) QUIC connection between the UE and the ATSSS UPF. With a multipath transport, splitting is naturally supported.

Data sent over one access network can be retransmitted over the other if the first becomes congested. Some measurements and simulations have shown that Multipath QUIC provides similar performance as Multipath TCP when combining different access networks [MPQUIC-Conext].

Steering can be also supported, similarly to MPTCP. The path scheduler can map datagrams carrying an entire flow to one or another access network based on the provided rules.

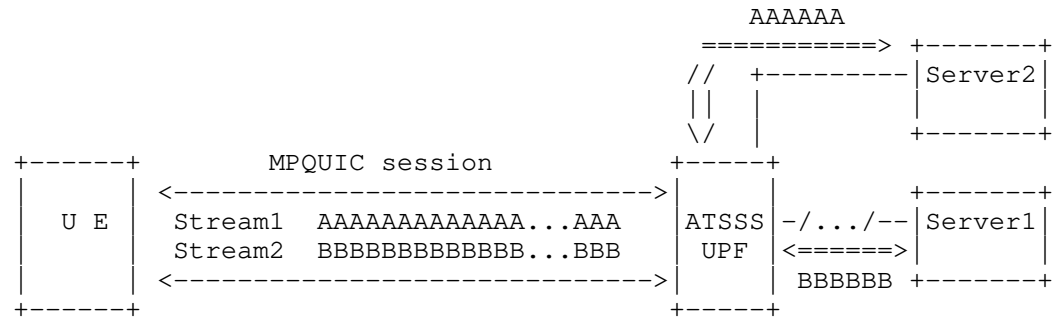
Switching can be improved by splitting traffic simultaneously over both links such that the congestion window of the new path can be open before the old path goes away. This makes handovers smoother. Experience with Multipath TCP on smartphones has shown that handovers are not a binary process. When a smartphone performs handovers, there are frequently short periods of time during which both networks are imperfect [MPTester]. Using use both networks simultaneously during these periods, improves user experience.

Furthermore, traffic can be better distributed among available paths based on available resources if one of the access networks fails or begins to become congested.

7.2. Mapping of Both TCP and Non-TCP to QUIC Streams and Datagrams

In ATSSS Phase 1, each TCP connection originated by the UE corresponds to one MPTCP connection that is terminated on the ATSSS UPF. With ATSSS Phase 2 using MPQUIC as a tunneling protocol, one can leverage the multi-stream capability of QUIC to carry the bytestreams of multiple TCP connections over a single MPQUIC session.

Focusing on the case where the UE maintains one QUIC connection with its ATSSS UPF, every TCP connection that it creates results in the creation of a new stream of the MPQUIC connection with the ATSSS UPF. Similarly the ATSSS UPF can map each incoming TCP connection from a remote host onto a stream of the MPQUIC connection. This is illustrated in Figure 7. Stream mapping is most beneficial for TCP, as it avoids reordering and TCP anyway requires in-order delivery. It is more appropriate to use QUIC with the Unreliable Datagram Extension for all other traffic.



<----> MPQUIC session
<====> TCP connection

Figure 7: ATSSS Phase 2 Maps Each TCP Connection on a Stream of the MPQUIC Connection Between the UE and the ATSSS UPF

Two application layer protocols are proposed to manage the mapping of TCP connections to QUIC streams as well as the transmission of UDP datagrams over QUIC datagrams:

(1) The proposed masque working group (wg) extends the HTTP/3 CONNECT method with a UDPCONNECT method to use QUIC as a tunnel for UDP traffic [I-D.schinazi-masque-connect-udp]. Another use case in scope for masque wg is IP proxying which can be used for tunneling and forwarding of TCP connections [I-D.schinazi-masque-protocol].

(2) QUIC Tunnel is a close proposal providing similar functionalities to MASQUE based on a binary protocol [I-D.piraux-quic-tunnel][I-D.piraux-quic-tunnel-tcp], and focusing on the ATSSS use case.

Both proposals rely upon single QUIC connections and inherit thus the same issues discussed in Section 7.1.3 and Section 7.1.4.

7.3. Encapsulation Overhead

In 3GPP architectures, a variety of encapsulations are already used to carry user data. The use of QUIC as a tunneling method for ATSSS will add some additional overhead. When the user data is forwarded over a non-3GPP network access, this overhead comes further in addition to an IPsec tunnel between the UE to the 5G Core Network which is already at least 142 octets for an IPv6 packet forwarded over a non-3GPP network access. In contrast, the MPTCP based mechanism in ATSSS Phase 1 does only add a minor overhead during

connection establishment, but no additional overhead during the rest of the connection.

More investigation is required to assess whether the ATSSS Phase 2 overhead is an issue. Solutions to optimize that overhead could be considered, if needed.

7.4. Multiple Encryptions

The use of QUIC as a tunneling protocol in ATSSS will add an additional layer of encryption. As there is already encryptions on other layers (e.g., IP Encapsulating Security Payload (ESP)) this leads to multiple layers of encryption, which might be redundant.

Such multiple encryptions will have performance implications on the UE, in particular. Means to optimize the various redundant encryptions are for further investigation in 3GPP.

7.5. Congestion Control in Congestion Control and Coexistence

Many applications that are sending unreliable traffic use various congestion control algorithms to detect congestion and adjust their sending rate based on inferred end-to-end latency and loss characteristics. Examples include Adaptive Video Streaming using e.g. SCREAM [RFC8298], or other media streaming applications that use QUIC as transport layer. When using QUIC version 1 with Unreliable Datagram Extension or Multipath QUIC as ATSSS solution, this leads to nested congestion control, where both inner and outer congestion control coexist. When using Multiple QUICv1 tunnels with Unreliable Datagram Extension or MP-QUIC tunneling, the packet scheduler could have an additional impact on the perceived end-to-end latency and loss due to the potential difference of the individual path characteristics.

When deploying ATSSS Phase 1 and Phase 2 in parallel, with the UE serving both reliable and unreliable flows, different congestion control algorithms may coexist on individual paths, which may lead to fairness issues or even meltdown effects.

We recognize that nested congestion control mechanisms (such as QUIC encapsulated in QUIC or SCREAM encapsulated in QUIC) as well as the coexistence of ATSSS Phase 1 and Phase 2 have implications that require further study.

7.6. Packet Order Reconstruction for (MP-)QUIC Splitting Mode

Both tunnel solutions in Section 7.1.4 and Section 7.1.5 allow the splitting mode for simultaneously sending data over multiple paths. In this case packet reordering can occur when a QUIC tunnel communication is split across paths with very different latencies. Generally, applications have to deal with packet reordering, since the best effort for Internet traffic has no guarantee to prevent it. However, in practice, packet reordering in the network is assumed to be very limited. Applications that require in-order delivery and e.g. rely on TCP or implement a similar mechanism itself can be sensitive to high amounts of reordering and experience decreased performance.

As such it is desirable that the respective receiver side of the tunnel termination points has the ability to reconstruct the original packet ordering. While in the case when using the MPTCP converter, losses are retransmitted quickly on the local segment, when the Unreliable Datagram Extension for QUIC is used, the reconstruction mechanism has to further account for packet loss which may occur for any of the paths within the QUIC tunnel. This can cause delays in the reordering logic, which in turn can have a negative impact on applications that do not require in-order delivery, such as real-time transmissions.

It is assumed that this is a solvable task similar to [MPDCCP-paper], but is probably left to the implementer, to take care. Even if the reconstruction of the packet order does not become a standardized part of the MP-QUIC in Section 7.1.5, it possibly requires path sequencing and end-to-end sequencing.

8. QUICv1 Gap Analysis for ATSSS Phase 2

This section summarizes QUIC protocol capabilities that would be beneficial for ATSSS Phase 2, as described in Section 7.1.

- ATSSS Phase 2 is focused on transport services for Ethernet frames and IP packets (with the intention of supporting TCP, UDP, and UDP-encapsulated transport protocols such as QUIC). The discussed approaches are based on tunnelling Ethernet or IP directly over QUIC. The masque working group that is currently in the chartering validation process is scoped to cover UDP and IP proxying. While UDP proxying does cover the most important use case to support ATSSS for UDP/QUIC, a more generic solution based on IP proxying would simplify the ATSSS design. However, IP proxying is only considered at a later stage by the masque working group. Also, multipath mechanisms of QUIC are not covered in the proposed charter.

- We envision the ability to select paths (steering), detect path failures and reroute traffic (switching), and forward packets on multiple active paths simultaneously (splitting), based on external policies, including active-standby, smallest delay, weighted load-balancing, and path selection based on assigned priorities, for the full range of encapsulated protocols in ATSSS Phase 2, similar to the abilities provided by Multipath TCP for ATSSS Phase 1. Splitting cannot be supported easily in the discussed QUICv1-based approaches. Multipath transport capability similar to Multipath TCP, as used in ATSSS Phase 1, would support splitting well. The QUIC working group is originally chartered to produce a multipath extension document by December 2021. Proposals exist, however, this work has been postponed to QUICv2 and discussion is still on-going if support will be kept in the charter.

- While the base protocol for QUICv1 does not provide support for unreliable datagrams, an QUIC extension for datagram support has been adopted by the group and the QUIC working group is chartered to produce this capability by March 2021. This can be used to support for the additional user-plane protocols as envisioned in ATSSS Phase 2.

- When QUIC is used as a tunneling protocol, nested congestion control mechanisms (such as QUIC encapsulated in QUIC) have implications that require further study.

- When QUIC is used as a tunneling protocol, the complete ATSSS Phase 2 protocol stack would include encrypted headers at multiple layers. It needs further investigation if this is a problem for ATSSS, however, it is likely a problem that can be solved by the 3GPP. Likewise, the implications of the various encapsulation overhead is to be further assessed within 3GPP.

9. IANA Considerations

This document does not make any request to IANA.

10. Security Considerations

Section 9 of [RFC6459] provides an overview of security considerations in 3GPP networks. ATSSS Phase 1 data plane security considerations are documented in Section 9 of [I-D.ietf-tcpm-converters].

This document discusses the use of QUIC (including Multipath QUIC) as an additional ATSSS steering method. QUIC-specific security

considerations are discussed in Section 21 of [I-D.ietf-quic-transport].

This document does not specify specific mechanisms to use QUIC as a tunneling protocol towards an ATSSS proxy, as the intention of this document is to provide an informational overview of the ongoing work in 3GPP on ATSSS to support non-TCP, rather than discussing a detailed solution. Nevertheless, this document cites candidate solutions to provide such tunneling service. Security considerations specific to these solutions are provided below.

Multipath QUIC-specific security considerations can be found in Section 8 of [I-D.deconinck-quic-multipath].

Section 6 of {I-D.ietf-quic-datagram} discusses security considerations specific to the use of the Unreliable Datagram Extension to QUIC.

11. Acknowledgements

Many thanks to Anna Brunstrom, Dieter Gludovacz, Dirk von-Hugo, Tim Costello, Stephen Johnson, and Florin Baboescu for the review, comments, and suggestions.

12. Document History

(Note to RFC Editor - if this document ever reaches you, please remove this section)

Version -00: initial submission

13. References

13.1. Normative References

[TS23501] 3GPP (3rd Generation Partnership Project), ., "Technical Specification Group Services and System Aspects; System Architecture for the 5G System; Stage 2 (Release 16)", 2019, <https://www.3gpp.org/ftp/Specs/archive/23_series/23.501/>.

13.2. Informative References

[atsssliaison] 3GPP (3rd Generation Partnership Project), ., "LS on need for Multi-Path QUIC for ATSSS", April 2020, <https://datatracker.ietf.org/liaison/1678/>.

[Hybrid] Keukeleire, N., Hesmans, B., and O. Bonaventure, "Increasing broadband reach with Hybrid Access Networks", IEEE Communications and Standards Magazine, Vol. 4, Issue 1 , March 2020, <https://doi.org/10.1109/MCOMSTD.001.1900036>.

[I-D.amend-tsvwg-multipath-dccp] Amend, M., Bogenfeld, E., Brunstrom, A., Kassler, A., and V. Rakocevic, "DCCP Extensions for Multipath Operation with Multiple Addresses", draft-amend-tsvwg-multipath-dccp-03 (work in progress), November 2019.

[I-D.bonaventure-iccrg-schedulers] Bonaventure, O., Piraux, M., Coninck, Q., Baerts, M., Paasch, C., and M. Amend, "Multipath schedulers", draft-bonaventure-iccrg-schedulers-00 (work in progress), March 2020.

[I-D.boucadair-mptcp-plain-mode] Boucadair, M., Jacquenet, C., Bonaventure, O., Behaghel, D., stefano.secci@lip6.fr, s., Henderickx, W., Skog, R., Vinapamula, S., Seo, S., Cloetens, W., Meyer, U., Contreras, L., and B. Peirens, "Extensions for Network-Assisted MPTCP Deployment Models", draft-boucadair-mptcp-plain-mode-10 (work in progress), March 2017.

[I-D.deconinck-quic-multipath] Coninck, Q. and O. Bonaventure, "Multipath Extensions for QUIC (MP-QUIC)", draft-deconinck-quic-multipath-04 (work in progress), March 2020.

[I-D.ietf-quic-datagram] Pauly, T., Kinnear, E., and D. Schinazi, "An Unreliable Datagram Extension to QUIC", draft-ietf-quic-datagram-00 (work in progress), February 2020.

[I-D.ietf-quic-transport] Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-28 (work in progress), May 2020.

[I-D.ietf-tcpm-converters] Bonaventure, O., Boucadair, M., Gundavelli, S., Seo, S., and B. Hesmans, "0-RTT TCP Convert Protocol", draft-ietf-tcpm-converters-19 (work in progress), March 2020.

[I-D.piraux-quic-tunnel]
Piraux, M. and O. Bonaventure, "Tunneling Internet protocols inside QUIC", draft-piraux-quic-tunnel-01 (work in progress), March 2020.

[I-D.piraux-quic-tunnel-tcp]
Piraux, M. and O. Bonaventure, "Tunneling TCP inside QUIC", draft-piraux-quic-tunnel-tcp-00 (work in progress), March 2020.

[I-D.schinazi-masque-connect-udp]
Schinazi, D., "The CONNECT-UDP HTTP Method", draft-schinazi-masque-connect-udp-00 (work in progress), April 2020.

[I-D.schinazi-masque-protocol]
Schinazi, D., "The MASQUE Protocol", draft-schinazi-masque-protocol-01 (work in progress), March 2020.

[IETFJ16] Bonaventure, O. and S. Seo, "Multipath TCP Deployment", IETF Journal, Fall 2016 , 2016, <https://www.ietfjournal.org/multipath-tcp-deployments/>.

[MPDCCP-paper]
Amend, M., Bogenfeld, E., Cvjetkovic, M., Rakocevic, V., Pieska, M., Kassler, A., and A. Brunstrom, "A Framework for Multiaccess Support for Unreliable Traffic using Multipath DCCP", 2019 IEEE 44th Conference on Local Computer Networks (LCN) , 2019, <https://doi.org/10.1109/LCN44214.2019.8990746>.

[MPQUIC-Conext]
De Coninck, Q. and O. Bonaventure, "Multipath QUIC - Design and evaluation", Proceedings of the 13th international conference on emerging networking experiments and technologies 2017 Nov 28 (pp. 160-166). , 2017, <https://multipath-quic.org/>.

[MPTester]
De Coninck, Q. and O. Bonaventure, "MultipathTester - Comparing MPTCP and MPQUIC in Mobile Environments. In 2019 Network Traffic Measurement and Analysis Conference (TMA)", 2019, <https://multipath-quic.org/multipathtester/2019/06/18/mnm-paper.html>.

[QUIC-Deployment]
Kuehlewind, M., "Some updates on QUIC deployment numbers", 2019, <https://datatracker.ietf.org/meeting/106/materials/slides-106-maprg-quic-deployment-update-00>.

[RFC5213] Gundavelli, S., Ed., Leung, K., Devarapalli, V., Chowdhury, K., and B. Patil, "Proxy Mobile IPv6", RFC 5213, DOI 10.17487/RFC5213, August 2008, <https://www.rfc-editor.org/info/rfc5213>.

[RFC5533] Nordmark, E. and M. Bagnulo, "Shim6: Level 3 Multihoming Shim Protocol for IPv6", RFC 5533, DOI 10.17487/RFC5533, June 2009, <https://www.rfc-editor.org/info/rfc5533>.

[RFC6275] Perkins, C., Ed., Johnson, D., and J. Arkko, "Mobility Support in IPv6", RFC 6275, DOI 10.17487/RFC6275, July 2011, <https://www.rfc-editor.org/info/rfc6275>.

[RFC6459] Korhonen, J., Ed., Soininen, J., Patil, B., Savolainen, T., Bajko, G., and K. Iisakkila, "IPv6 in 3rd Generation Partnership Project (3GPP) Evolved Packet System (EPS)", RFC 6459, DOI 10.17487/RFC6459, January 2012, <https://www.rfc-editor.org/info/rfc6459>.

[RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <https://www.rfc-editor.org/info/rfc6824>.

[RFC8041] Bonaventure, O., Paasch, C., and G. Detal, "Use Cases and Operational Experience with Multipath TCP", RFC 8041, DOI 10.17487/RFC8041, January 2017, <https://www.rfc-editor.org/info/rfc8041>.

[RFC8298] Johansson, I. and Z. Sarker, "Self-Clocked Rate Adaptation for Multimedia", RFC 8298, DOI 10.17487/RFC8298, December 2017, <https://www.rfc-editor.org/info/rfc8298>.

[TR-348] Broadband Forum, ., "Hybrid Access Broadband Network Architecture", 2016, <https://www.broadband-forum.org/download/TR-348.pdf>.

Authors' Addresses

Mohamed Boucadair (editor)
Orange
Clos Courtel
Rennes 35000
France

Email: mohamed.boucadair@orange.com

Olivier Bonaventure (editor)
UCLouvain

Email: Olivier.Bonaventure@uclouvain.be

Maxime Piraux (editor)
UCLouvain

Email: Maxime.Piraux@uclouvain.be

Quentin De Coninck
UCLouvain

Email: quentin.deconinck@uclouvain.be

Spencer Dawkins (editor)
Tencent America

Email: spencerdawkins.ietf@gmail.com

Mirja Kuehlewind (editor)
Ericsson

Email: mirja.kuehlewind@ericsson.com

Markus Amend
Deutsche Telekom

Email: markus.amend@telekom.de

Andreas Kassler
Karlstad University

Email: andreas.kassler@kau.se

Qing An
Alibaba Group

Email: anqing.aq@alibaba-inc.com

Nicolas Keukeleire
Tessares

Email: nicolas.keukeleire@tessares.net

SungHoon Seo
Korea Telecom

Email: sh.seo@kt.com

QUIC
Internet-Draft
Intended status: Informational
Expires: 30 January 2021

M. Duke
F5 Networks, Inc.

29 July 2020

Network Address Translation Support for QUIC
draft-duke-quic-natsupp-03

Abstract

Network Address Translators (NATs) are widely deployed to share scarce public IPv4 addresses among multiple end hosts. They overwrite IP addresses and ports in IP packets to do so. QUIC is a protocol on top of UDP that provides transport-like services. QUIC is better-behaved in the presence of NATs than older protocols, and existing UDP NATs should operate without incident if unmodified. QUIC offers additional features that may tempt NAT implementers as potential optimizations. However, in practice, leveraging these features will lead to new connection failure modes and security vulnerabilities.

Table of Contents

1. Introduction

Network Address Translators (NATs) are a widely deployed means of multiplexing multiple private IP addresses over scarce IPv4 public address space by replacing those addresses and using ports to distinguish those connections. The new address can also guarantee that packets move through a proxy throughout the life of a connection, so that the connection can continue with the required state at that proxy.

This document uses the colloquial term NAT to mean NAPT (section 2.2 of [RFC3022]), which overloads several IP addresses to one IP address or to an IP address pool, as commonly deployed in carrier-grade NATs or residential NATs.

QUIC [QUIC-TRANSPORT] is a protocol, operating over UDP, that provides many transport-like services to the application layer. Among these services is the mapping of multiple endpoint IP addresses to a single connection through use of a Connection ID (CID). Connection IDs are opaque byte fields that are expressed consistently across all QUIC versions [QUIC-INVARIANTS]. This feature may appear to present opportunities to optimize NAT port usage and simplify the work of the QUIC server. In fact, NAT behavior that relies on CID may instead cause connection failure when endpoints change Connection ID, and disable important protocol security features.

The remainder of this document explains how QUIC supports NATs better than other connection-oriented protocols, why NAT use of Connection ID might appear attractive, and how NAT use of CID can create serious problems for the endpoints. The conclusion of this document is that NATs should retain their existing 4-tuple-based operation and refrain from parsing or otherwise using QUIC connection IDs.

[RFC4787] contains some guidance on building NATs to interact constructively with a wide range of applications. This document extends the discussion to QUIC.

2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. QUIC and NAT Rebinding

An explicit goal of QUIC is to be robust to NAT rebinding. When a connection is idle for a long time, the NAT may guess it has terminated and assign the client port to a new connection. As TCP defines a connection by its address and port 4-tuple, a TCP packet will not appear to belong to any existing connection at the receiver.

QUIC endpoints identify their connections using a CID that is encoded in every packet. If the client attempts to resume communication, the packet will be assigned a new source IP and/or port. Incoming packets from the server will be misrouted and dropped until the client sends a packet from its new address.

Therefore, QUIC connections can survive NAT rebindings as long as no routing function in the path is dependent on client IP address and port to deliver packets between server and NAT. Reducing the timeout on UDP NATs might be tempting in light of this property, but not all QUIC server deployments will be robust to rebinding.

4. The Lure of the Connection ID

There are a few reasons that CID-aware NATs could seemingly appear attractive.

4.1. Resource Conservation

NATs sometimes hit an operational limit where they exhaust available public IP addresses and ports, and must evict flows from their address/port mapping. CIDs offer a way to multiplex many connections over a single address and port.

However, QUIC endpoints may negotiate new connection IDs inside cryptographically protected packets, and begin using them at will. Imagine two clients behind a NAT that are sharing the same public IP address and port. The NAT is differentiating them using the incoming Connection ID. If one client secretly changes its connection ID, there will be no mapping for the NAT, and the connection will suddenly break.

While mid-connection failure in some cases may seem superior to rejecting QUIC outright, HTTP/3 over QUIC falls back to TCP. This is preferable to a connection suddenly black holing and timing out. Furthermore, wide deployment of NATs with this behavior would make it risky to change Connection IDs in the internet, which would thwart various important protocol properties.

It is possible, in principle, to encode the client's identity in a connection ID using [QUIC-LB] and explicit coordination with the NAT. However, QUIC-LB makes assumptions about endpoint mobility and common configuration in server infrastructure that are almost never valid in client/NAT architectures. Deploying such a system would include the administrative overhead while not solving the problem described in this section if the client changes networks.

Note that using connection IDs in this manner would anyway violate the best common practice to avoid "port overloading" as described in [RFC4787].

4.2. "Helping" with routing infrastructure issues

One problem in QUIC deployment is router and switch server infrastructures that direct traffic based on address-port 4-tuple rather than connection ID. The use of source IP address means that a NAT rebinding or address migration will deliver packets to the wrong server. For the reasons described above, routers and switches will not have access to negotiated CIDs. This is a particular problem for low-state load balancers, and a QUIC extension exists [QUIC-LB] to allow some server-load balancer coordination for routable CIDs.

A NAT at the front of this infrastructure might save the effort of converting all these devices by decoding routable connection IDs and rewriting the packet IP addresses to allow consistent routing by legacy devices.

Unfortunately, the change of IP address or port is an important signal to QUIC endpoints. It requires a review of path-dependent variables like congestion control parameters. It can also signify various attacks that mislead one endpoint about the best peer address for the connection (see section 9 of [QUIC-TRANSPORT]). The QUIC PATH_CHALLENGE and PATH_RESPONSE frames are intended to detect and mitigate these attacks and verify connectivity to the new address. This mechanism cannot work if the NAT is bleaching peer address changes.

For example, an attacker might copy a legitimate QUIC packet and change the source address to match its own. In the absence of a bleaching NAT, the receiving endpoint would interpret this as a potential NAT rebinding and use a PATH_CHALLENGE frame to prove that the peer endpoint is not truly at the new address, thus thwarting the attack. A bleaching NAT has no means of sending an encrypted PATH_CHALLENGE frame, so it might start redirecting all QUIC traffic to the attacker address and thus allow an observer to break the connection.

5. Filtering behavior

[RFC4787] describes possible packet filtering behaviors that relate to NATs. Though thes guidance there holds, a particularly unwise behavior is to admit a handful of UDP packets and then make a decision as to whether or not to filter it. QUIC applications are encouraged to fail over to TCP if early packets do not arrive at their destination. 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.

6. QUIC Detection

Beyond the above difficulties, merely identifying that a UDP packet is part of a QUIC connection is not straightforward. Due to address migration, NATs cannot assume that QUIC version 1 application traffic is preceeded by a handshake on the path. The short header prepended to version 1 application traffic has few consistent codepoints that reliably identify it as QUIC. Moreover, the protocol is designed to be extensible. [QUIC-INVARIANTS] describes the small set of QUIC protocol properties that will remain stable across versions.

For these reasons, applying generalized UDP policies will prevent accidental breakage of QUIC features and mishandled non-QUIC UDP packets.

7. Security Considerations

This document proposes no change in behavior in the internet, so there are no new security implications. However, ignoring the recommendations here could prevent existing security mechanisms in QUIC from working properly.

8. IANA Considerations

There are no IANA requirements.

9. Informative References

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

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

[QUIC-TRANSPORT]
Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", Work in Progress, Internet-Draft, draft-ietf-quic-transport-latest, <https://tools.ietf.org/html/draft-ietf-quic-transport-latest>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3022] Srisuresh, P. and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)", RFC 3022, DOI 10.17487/RFC3022, January 2001, <https://www.rfc-editor.org/info/rfc3022>.

[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>.

Appendix A. Acknowledgments

Thanks to Dmitri Tikhonov, who first recognized that certain NAT behaviors could create problems for QUIC.

Appendix B. Change Log

RFC Editor's Note: Please remove this section prior to$ publication of a final version of this document.$

B.1. since draft-duke-quic-natsupp-02

* Added discussion of QUIC identification

B.2. since draft-duke-quic-natsupp-01

* Added brief discussion of impact of filtering.

* Added references to RFC 4787.

* Corrected normative reference to be informative.

B.3. since draft-duke-quic-natsupp-00

* Tightened NAT terminology

* Added additional clarfying examples

* Added warning against using QUIC-LB for NATs that front clients.

Author's Address

Martin Duke
F5 Networks, Inc.

Email: martin.h.duke@gmail.com

QUIC
Internet-Draft
Intended status: Experimental
Expires: 25 March 2021

M. Duke
F5 Networks, Inc.

21 September 2020

QUIC Version Aliasing
draft-duke-quic-version-aliasing-02

Abstract

The QUIC transport protocol [QUIC-TRANSPORT] preserves its future extensibility partly by specifying its version number. There will be a relatively small number of published version numbers for the foreseeable future. This document provides a method for clients and servers to negotiate the use of other version numbers in subsequent connections and encrypts Initial Packets using secret keys instead of standard ones. If a sizeable subset of QUIC connections use this mechanism, this should prevent middlebox ossification around the current set of published version numbers and the contents of QUIC Initial packets, as well as improving the protocol's privacy properties.

Table of Contents

1. Introduction

The QUIC version number is critical to future extensibility of the protocol. Past experience with other protocols, such as TLS1.3 [RFC8446], shows that middleboxes might attempt to enforce that QUIC packets use versions known at the time the middlebox was implemented. This has a chilling effect on deploying experimental and standard versions on the internet.

Each version of QUIC has a "salt" [QUIC-TLS] that is used to derive the keys used to encrypt Initial packets. As each salt is published in a standards document, any observer can decrypt these packets and inspect the contents, including a TLS Client Hello. A subsidiary mechanism like Encrypted SNI [ENCRYPTED-SNI] might protect some of the TLS fields inside a TLS Client Hello.

This document proposes "QUIC Version Aliasing," a standard way for servers to advertise the availability of other versions inside the cryptographic protection of a QUIC handshake. These versions are syntactically identical to the QUIC version in which the communication takes place, but use a different salt. In subsequent communications, the client uses the new version number and encrypts its Initial packets with a key derived from the provided salt. These version numbers and salts are unique to the client.

If a large subset of QUIC traffic adopts his technique, middleboxes will be unable to enforce particular version numbers or policy based on Client Hello contents without incurring unacceptable penalties on users. This would simultaneously protect the protocol against ossification and improve its privacy properties.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying significance described in RFC 2119.

A "standard version" is a QUIC version that would be advertised in a QUIC version negotiation and conforms to a specification. Any aliased version corresponds to a standard version in all its formats and behaviors, except for the version number field in long headers.

An "aliased version" is a version with a number generated in accordance with this document. Except for the version field in long headers, it conforms entirely to the specification of the standard version.

2. Protocol Overview

When they instantiate a connection, servers select an alternate 32-bit version number, and optionally an initial token extension, for the next connection at random and securely derive a salt and Packet Length Offset from those values using a repeatable process. They communicate this using a transport parameter extension including the version, initial token extension, salt, Packet Length Offset, and an expiration time for that value.

If a client next connects to that server within the indicated expiration time, it MAY use the provided version number and encrypt its Initial Packets using a key derived from the provided salt. It adds the Packet Length Offset to the true packet length when encoding it in the long header. If the server provided an Initial Token Extension, the client puts it in the Initial Packet token field. If there is another token the client wishes to include, it appends the Initial Token Extension to that token. The server can reconstruct the salt and Packet Length Offset from the requested version and token, and proceed with the connection normally.

The Packet Length Offset is provides a low-cost way for the server to verify it can derive a valid salt from the inputs without trial decryption. This has important security implications, as described in Section 7.2.

When generating a salt and Packet Length Offset, servers can choose between doing so randomly and storing the mapping, or using a cryptographic process to transform the aliased version number and token extension into the salt. The two options provide a simple tradeoff between computational complexity and storage requirements.

Note that clients and servers MUST implement [QUIC-VERSION-NEGOTIATION] to use this specification. Therefore, servers list supported versions in Version Negotiation Packets. Both clients and servers list supported versions in Version Negotiation Transport Parameters.

3. The Version Alias Transport Parameter

3.1. Version Number Generation

Servers MUST use a random process to generate version numbers. This version number MUST NOT correspond to a QUIC version the server advertises in QUIC Version Negotiation packets or transport parameters. Servers SHOULD also exclude version numbers used in known specifications or experiments to avoid confusion at clients, whether or not they have plans to support those specifications.

Servers MUST NOT use client-controlled information (e.g. the client IP address) in the random process, see Section 7.4.

Servers MUST NOT advertise these versions in QUIC Version Negotiation packets.

3.2. Initial Token Extension (ITE) Generation

Servers SHOULD generate an Initial Token Extension (ITE) to provide additional entropy in salt generation. Two clients that receive the same version number but different extensions will not be able to decode each other's Initial Packets.

Servers MAY choose any length that will allow client Initial Packets to fit within the minimum QUIC packet size of 1200 octets. A four-octet extension is RECOMMENDED. The ITE MUST appear to be random to observers.

If a server supports multiple standard versions, it MUST either encode the standard version of the current connection in the ITE or store it in a lookup table.

If the server chooses to encode the standard version, it MUST be cryptographically protected.

Encoded standard versions MUST be robust to false positives. That is, if decoded with a new key, the version encoding must return as invalid, rather than an incorrect value.

Alternatively, servers MAY store a mapping of unexpired aliased versions and ITEs to standard versions. This mapping SHOULD NOT create observable patterns, e.g. one plaintext bit indicates if the standard version is 1 or 2.

The server MUST be able to distinguish ITEs from Resumption and Retry tokens in incoming Initial Packets that contain an aliased version number. As the server controls the lengths and encoding of each, there are many ways to guarantee this.

3.3. Salt and Packet Length Offset Generation

The salt is an opaque 20-octet field. It is used to generate Initial connection keys using the process described in [QUIC-TLS].

The Packet Length Offset is a 64-bit unsigned integer with a maximum value of 2^62 - 1. Clients MUST ignore a transport parameter with a value that exceeds this limit.

To reduce header overhead, servers MAY consistently use a Packet Length Offset of zero if and only if it either (1) never sends Retry packets, or (2) can guarantee, through the use of persistent storage or other means, that it will never lose the cryptographic state required to generate the salt before the promised expiration time. Section 7.2 describes the implications if it uses zero without meeting these conditions.

Servers MUST either generate a random salt and Packet Length Offset and store a mapping of aliased version and ITE to salt and offset, or generate the salt and offset using a cryptographic method that uses the version number, ITE, and only server state that is persistent across connections.

If the latter, servers MUST implement a method that it can repeat deterministically at a later time to derive the salt and offset from the incoming version number and ITE. It MUST NOT use client controlled information other than the version number and ITE; for example, the client's IP address and port.

3.4. Expiration Time

Servers should select an expiration time in seconds, measured from the instant the transport parameter is first sent. This time SHOULD be less than the time until the server expects to support new QUIC versions, rotate the keys used to encode information in the version number, or rotate the keys used in salt generation.

Furthermore, the expiration time SHOULD be short enough to frustrate a salt polling attack ({salt-polling}})

Conversely, an extremely short expiration time will often force the client to use standard QUIC version numbers and salts.

3.5. Format

This document defines a new transport parameter extension for QUIC with identifier 0x5641. The contents of the value field are indicated below.

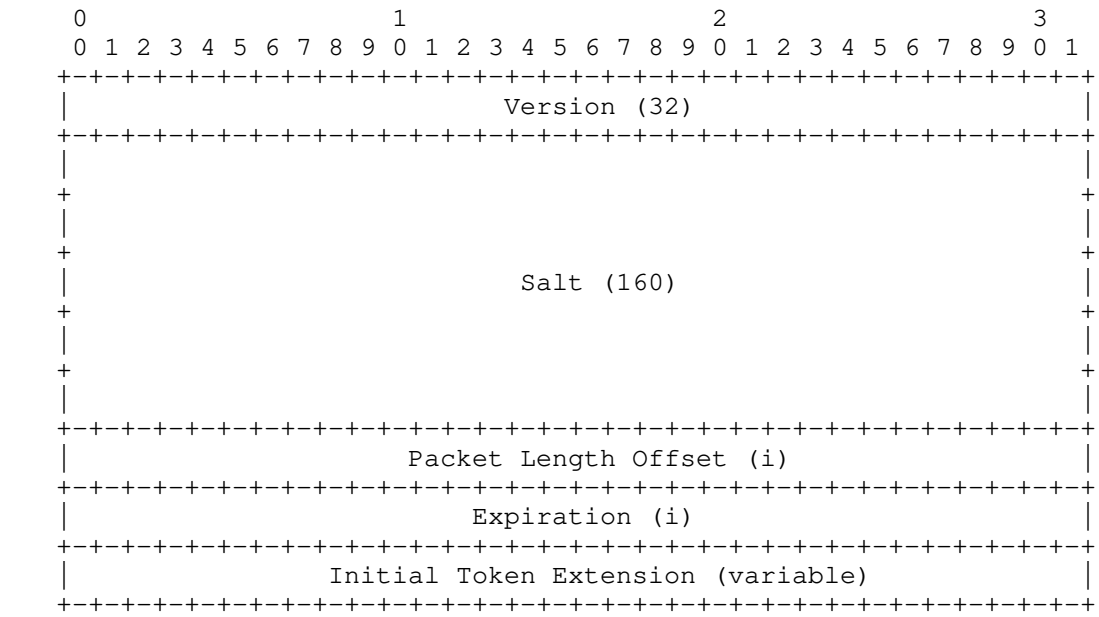


Figure 1: Version Alias Transport Parameter value

The definition of the fields is described above. Note that the "Expiration" field is in seconds, and its length is encoded using the Variable Length Integer encoding from Section 16 of [QUIC-TRANSPORT].

The Packet Length Offset is also encoded as a Variable Length Integer.

Clients can determine the length of the Initial Token Extension by subtracting known and encoded field lengths from the overall transport parameter length.

3.6. Multiple Servers for One Domain

If multiple servers serve the same entity behind a load balancer, all such servers SHOULD either have a common configuration for encoding standard versions and computing salts, or share a common database of mappings. They MUST NOT generate version numbers that any of them would advertise in a Version Negotiation Packet or Transport Parameter.

4. Client Behavior

When a client receives the Version Alias Transport Parameter, it MAY cache the version number, ITE, salt, Packet Length Offset, and the expiration of these values. It MAY use the version number and ITE in a subsequent connection and compute the initial keys using the provided salt.

Clients MUST NOT advertise aliased versions in the Version Negotiation Transport Parameter unless they support a standard version with the same number. Including that number signals support for the standard version, not the aliased version.

Clients SHOULD NOT attempt to use the provided version number and salt after the provided Expiration time has elapsed.

Clients MAY decline to use the provided version number or salt in more than one connection. It SHOULD do so if its IP address has changed between two connection attempts. Using a consistent version number can link the client across connection attempts.

Clients MUST use the same standard version to format the Initial Packet as the standard version used in the connection that provided the aliased version.

If the server provided an ITE, the client MUST append it to any Initial Packet token it is including from a Retry packet or NEW_TOKEN frame, if it is using the associated aliased version. If there is no such token, it simply includes the ITE as the entire token.

The QUIC Token Length field MUST include the length of both any Retry or NEW_TOKEN token and the ITE.

The Length fields of all Initial, Handshake, and 0-RTT packets in the connection are set to the value described in [QUIC-TRANSPORT] plus the provided Packet Length Offset, modulo 2^62.

If the response to an Initial packet using the provided version is a Version Negotiation Packet, the client SHOULD cease attempting to use that version and salt to the server unless it later determines that the packet was the result of a version downgrade, see Section 7.1.

If a client receives an aliased version number that matches a standard version that the client supports, it SHOULD assume the server does not support the standard version and MUST use aliased version behaviors in any connection with the server using that version number.

If a client receives a Version Negotiation packet or Version Negotiation transport parameter advertising a version number the server previously sent as an aliased version, and the client verifies any Version Negotiation Packet is not a Version Downgrade attack (Section 7.1), it MUST discard the aliased version number, ITE, packet length offset, and salt and not use it in future connections.

5. Server Actions on Aliased Version Numbers

When a server receives an Initial Packet with an unsupported version number, it SHOULD send a Version Negotiation Packet if it is specifically configured not to generate that version number at random.

Otherwise, it extracts the ITE, if any, and either looks up the corresponding salt in its database or computes it using the technique originally used to derive the salt from the version number and ITE.

The server similarly obtains the Packet Length Offset and subtracts it from the provided Length field, modulo 2^62. If the resulting value is larger than the entire UDP datagram, the server discards the packet and SHOULD send a Version Negotiation Packet.

If the server supports multiple standard versions, it uses the standard version extracted by the ITE or stored in the mapping to parse the decrypted packet.

In all packets with long headers, the server uses the aliased version number and adds the Packet Length Offset to the length field.

In the extremely unlikely event that the Packet Length Offset resulted in a legal value but the salt is incorrect, the packet may fail authentication. If so, or the encoded standard version is not supported at the server, the server SHOULD send a Version Negotiation Packet.

To reduce linkability for the client, servers SHOULD provide a new Version Alias transport parameter, with a new version number, ITE, salt, and Packet Length Offset, each time a client connects. However, issuing version numbers to a client SHOULD be rate-limited to mitigate the salt polling attack Section 7.4.

6. Considerations for Retry Packets

QUIC Retry packets reduce the load on servers during periods of stress by forcing the client to prove it possesses the IP address before the server decrypts any Initial Packets or establishes any connection state. Version aliasing substantially complicates the process.

If a server has to send a Retry packet, the required format is ambiguous without understanding which standard version to use. If all supported standard versions use the same Retry format, it simply uses that format with the client-provided version number.

If the supported standard versions use different Retry formats, the server obtains the standard version via lookup or decoding and formats a Retry containing the aliased version number accordingly.

Servers generate the Retry Integrity Tag of a Retry Packet using the procedure in Section 5.8 of [QUIC-TLS]. However, for aliased versions, the secret key K uses the first 16 octets of the aliased salt instead of the key provided in the specification.

Clients MUST ignore Retry packets that contain a QUIC version other than the version it used in its Initial Packet.

Servers MUST NOT reply to a packet with an incorrect Length field in its long header with a Retry packet; it SHOULD reply with Version Negotiation as described above.

7. Security and Privacy Considerations

This document intends to improve the existing security and privacy properties of QUIC by dramatically improving the secrecy of QUIC Initial Packets. However, there are new attacks against this mechanism.

7.1. Version Downgrade

A countermeasure against version aliasing is the downgrade attack. Middleboxes may drop a packet containing a random version and imitate the server's failure to correctly process it. Clients and servers are required to implement [QUIC-VERSION-NEGOTIATION] to detect downgrades.

Note that downgrade detection only works after receiving a response from the server. If a client immediately responds to a Version Negotiation Packet with an Initial Packet with a standard version number, it will have exposed its request in a format readable to

observers before it discovers if the Version Negotiation Packet is authentic. A client SHOULD wait for an interval to see if a valid response comes from the server before assuming the version negotiation is valid. The client MAY also alter its Initial Packet (e.g., its ALPN field) to sanitize sensitive information and obtain another aliased version before proceeding with its true request.

Servers that support version aliasing SHOULD be liberal about the Initial Packet content they receive, keeping the connection open long enough to deliver their transport parameters, to support this mechanism.

7.2. Retry Injection

QUIC Version 1 Retry packets are spoofable, as they follow a fixed format, are sent in plaintext, and the integrity protection uses a widely known key. As a result, QUIC Version 1 has verification mechanisms in subsequent packets of the connection to validate the origin of the Retry.

Version aliasing largely frustrates this attack. As the integrity check key is derived from the secret salt, packets from attackers will fail their integrity check and the client will ignore them.

The Packet Length Offset is important in this framework. Without this mechanism, servers would have to perform trial decryption to verify the client was using the correct salt. As this does not occur before sending Retry Packets, servers would not detect disagreement on the salt beforehand and would send a Retry packet signed with a different salt than the client expects. Therefore, a client that received a Retry packet with an invalid integrity check would not be able to distinguish between the following possibilities:

* a Retry packet corrupted in the network, which should be ignored;

* a Retry packet generated by an attacker, which should be ignored; or

* a Retry packet from a server that lost its cryptographic state, meaning that further communication with aliased versions is impossible and the client should revert to using a standard version.

The Packet Length Offset introduces sufficient entropy to make the third possibility exceedingly unlikely.

7.3. Increased Linkability

As each version number and ITE is unique to each client, if a client uses one twice, those two connections are extremely likely to be from the same host. If the client has changed IP address, this is a significant increase in linkability relative to QUIC with a standard version numbers.

7.4. Salt Polling Attack

Observers that wish to decode Initial Packets might open a large number of connections to the server in an effort to obtain part of the mapping of version numbers and ITEs to salts for a server. While storage-intensive, this attack could increase the probability that at least some version-aliased connections are observable. There are three mitigations servers can execute against this attack:

* use a longer ITE to increase the entropy of the salt,

* rate-limit transport parameters sent to a particular client, and/or

* set a low expiration time to reduce the lifetime of the attacker's database.

Segmenting the version number space based on client information, i.e. using only a subset of version numbers for a certain IP address range, would significantly amplify an attack. Observers will generally be on the path to the client and be able to mimic having an identical IP address. Segmentation in this way would dramatically reduce the search space for attackers. Thus, servers are prohibited from using this mechanism.

7.5. Increased Processing of Garbage UDP Packets

As QUIC shares the UDP protocol number with other UDP applications, in some deployments it may be possible for traffic intended for other UDP applications to arrive at a QUIC server endpoint. When servers support a finite set of version numbers, a valid version number field is a strong indicator the packet is, in fact, QUIC. If the version number is invalid, a QUIC Version Negotiation is a low-cost response that triggers very early in packet processing.

However, a server that provides version aliasing is prepared to accept almost any version number. As a result, many more sufficiently sized UDP payloads with the first bit set to '1' are potential QUIC Initial Packets that require generation of a salt and Packet Length Offset.

Note that a nonzero Packet Length Offset will allow the server to drop all but approximately 1 in every 2^49 packets, so trial decryption is unnecessary.

While not a more potent attack then simply sending valid Initial Packets, servers may have to provision additional resources to address this possibility.

7.6. Increased Retry Overhead

This document requires two small cryptographic operations to build a Retry packet instead of one, placing more load on servers when already under load.

8. IANA Considerations

This draft chooses a transport parameter (0x5641) to minimize the risk of collision. IANA should assign a permanent value from the QUIC Transport Parameter Registry.

9. References

9.1. Normative References

[QUIC-TLS] Thomson, M., Ed. and S. Turner, Ed., "Using Transport Layer Security (TLS) to Secure QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-tls-latest, <https://tools.ietf.org/html/draft-ietf-quic-tls-latest>.

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

[QUIC-VERSION-NEGOTIATION] Schinazi, D., Ed. and E. Rescorla, Ed., "Compatible Version Negotiation for QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-version-negotiation-latest, <https://tools.ietf.org/html/draft-ietf-quic-version-negotiation-latest>.

9.2. Informative References

[ENCRYPTED-SNI]
Rescorla, E., Ed., Oku, K., Ed., Sullivan, N., Ed., and C.A. Wood, Ed., "Encrypted Server Name Indication for TLS 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-esni-latest, <https://tools.ietf.org/html/draft-ietf-tls-esni-latest>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

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

Appendix A. Acknowledgments

Marten Seemann was the original progenitor of the version aliasing approach.

Appendix B. Change Log

RFC Editor's Note: Please remove this section prior to publication of a final version of this document.

B.1. since draft-duke-quic-version-aliasing-01

* Fixed all references to "seed" where I meant "salt."

* Added the Packet Length Offset, which eliminates Retry Injection Attacks

B.2. since draft-duke-quic-version-aliasing-00

* Added "Initial Token Extensions" to increase salt entropy and make salt polling attacks impractical.

* Allowed servers to store a mapping of version number and ITE to salt instead.

* Made standard version encoding mandatory. This dramatically simplifies the new Retry logic and changes the security model.

* Added references to Version Negotiation Transport Parameters.

* Extensive readability edit.

Author's Address

Martin Duke
F5 Networks, Inc.

Email: martin.h.duke@gmail.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 9 January 2021

M. Kuehlewind
Ericsson
B. Trammell
Google

8 July 2020

Applicability of the QUIC Transport Protocol
draft-ietf-quic-applicability-07

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.

Table of Contents

1. Introduction

QUIC [QUIC] is a new transport protocol currently under development in the IETF quic working group, focusing on support of semantics as needed for HTTP/2 [QUIC-HTTP] such as stream-multiplexing to avoid head-of-line blocking. Based on current deployment practices, QUIC is encapsulated in UDP. The version of QUIC that is currently under development will integrate TLS 1.3 [TLS13] to encrypt all payload data and most control information.

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 application use of HTTP/2 over QUIC as well as the use of other application layer protocols over QUIC. For specific guidance on how to integrate HTTP/2 with QUIC, see [QUIC-HTTP].

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.

1.1. Notational Conventions

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

2. The Necessity of Fallback

QUIC uses UDP as a substrate for userspace implementation and port numbers for NAT and middlebox traversal. While there is no evidence of widespread, systematic disadvantage of UDP traffic compared to TCP in the Internet [Edeline16], somewhere between three [Trammell16] and five [Swett16] percent of networks simply block 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. This fallback SHOULD provide TLS 1.3 or equivalent cryptographic protection, if available, in order to keep fallback from being exploited as a downgrade attack. In the case of HTTP, this fallback is TLS 1.3 over TCP.

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 without the TCP Fast Open extension does not support 0-RTT session resumption. TCP Fast Open can be requested by the connection initiator but might no be supported by the far end or could be blocked on the network path. Note that there is some evidence of middleboxes blocking SYN data even if TFO was successfully negotiated (see [PaaschNanog]).

Any fallback mechanism is likely to impose a degradation of performance; however, fallback MUST not silently violate the application's expectation of confidentiality or integrity of its payload data.

Moreover, while encryption (in this case TLS) is inseparably integrated with QUIC, TLS negotiation over TCP can be blocked. In case it is RECOMMENDED to abort the connection, allowing the application to present a suitable prompt to the user that secure communication is unavailable.

3. Zero RTT

QUIC provides for 0-RTT connection establishment. This presents opportunities and challenges for applications using QUIC.

3.1. Thinking in Zero RTT

A transport protocol that provides 0-RTT connection establishment to recently contacted servers 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 3.3.

Applications must be slightly rethought in order to make best use of 0-RTT resumption. Most importantly, application operations must be divided into idempotent and non-idempotent operations, as only idempotent operations may appear in 0-RTT packets. This implies that the interface between the application and transport layer exposes idempotence either explicitly or implicitly.

3.2. Here There Be Dragons

Retransmission or (malicious) replay of data contained in 0-RTT resumption packets could cause the server side to receive two copies of the same data. This is further described in [HTTP-RETRY]. Data sent during 0-RTT resumption also cannot benefit from perfect forward secrecy (PFS).

Data in the first flight sent by the client in a connection established with 0-RTT MUST be idempotent (as specified in section 2.1 in [QUIC-TLS]). Applications MUST be designed, and their data MUST be framed, such that multiple reception of idempotent data is recognized as such by the receiverApplications that cannot treat data that may appear in a 0-RTT connection establishment as idempotent MUST NOT use 0-RTT establishment. For this reason the QUIC transport SHOULD provide an interface for the application to indicate if 0-RTT support is in general desired or a way to indicate whether data is idempotent, whether PFS is a hard requirement for the application, and/or whether rejected 0-RTT dgitata should be retransmitted or withdrawn.

3.3. Session resumption versus Keep-alive

Because QUIC is encapsulated in UDP, applications using QUIC must deal with short idle timeouts. Deployed stateful middleboxes will generally establish state for UDP flows on the first packet state, and keep state for much shorter idle periods than for TCP. According to a 2010 study ([Hatonen10]), UDP applications can assume that any NAT binding or other state entry will be expired after just thirty seconds of inactivity.

A QUIC application has three strategies to deal with this issue:

* Ignore it, if the application-layer protocol consists only of interactions with no or very short idle periods.

* 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. Note that sending PING frames more frequently than every 30 seconds over long idle periods may result in a too much unproductive traffic and power usage for some situations.

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. This of course only applies 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.

The tradeoffs between resumption and keepalive need to be evaluated on a per-application basis. However, in general applications should use keepalives only in circumstances where continued communication is highly likely; [QUIC-HTTP], for instance, recommends using PING frames for keepalive 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 (UDP) packet on the wire can carry one of multiple stream frames.

Stream can be independently open and closed, gracefully or by error. If a critical stream for the application is closed, the application can generate respective error messages on the application layer to inform the other end or the higher layer and eventually indicate QUIC to reset the connection. QUIC, however, does not need to know which streams are critical, and does not provide an interface to exceptional handling of any stream. There are special streams in QUIC that are used for control on the QUIC connection, however, these streams are not exposed to the application.

Mapping of application data to streams is application-specific and described for HTTP/s in [QUIC-HTTP]. In general 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 different streams. If the application requires certain data to be received in order, the same stream should be used for that data. If there is a logical grouping of data chunks or messages, streams can be reused, or a new stream can be opened for each chunk/message. If one message is mapped to a single stream, resetting the stream to expire an unacknowledged message can be used to emulate partial reliability on a message basis. If a QUIC receiver has maximum allowed concurrent streams open and the sender on the other end indicates that more streams are needed, it doesn't automatically lead to an increase of the maximum number of streams by the receiver. Therefore it can be valuable to expose 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.

Further, streams have a maximum number of bytes that can be sent on one stream. This number is high enough (2^64) that this will usually not be reached with current applications. Applications that send chunks of data over a very long period of time (such as days, months, or years), should rather utilize the 0-RTT session resumption ability provided by QUIC, than trying to maintain one connection open.

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 respective 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]. Further 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 to retransmit 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. 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 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 ensures that this style of deadlock is 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 RST_STREAM. Cancelling some streams results in the connection being terminated in some protocols.

5. Packetization and Latency

QUIC provides an interface that provides multiple streams to the application; however, the application usually cannot control how data transmitted over one stream is mapped into frames or how those frames are bundled into packets. By default, QUIC will try to maximally pack packets with one or more stream data frames to minimize bandwidth consumption and computational costs (see section 8 of [QUIC]). If there is not enough data available to fill a packet, QUIC may even 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 appliaction has usually no control about the length of a QUIC packet on the wire. However, QUIC provides the ability to add a padding frame to impact the packet size. This is mainly used by QUIC itself in the first packet in order to ensure that the path is capable of transferring packets of at least a certain size. Additionally, a QUIC implementation can expose an application layer interface to specify a certain packet size. This can either be used by the application to force certian packet sizes in specific use cases/networks, or ensure that all packets are equally sized to conceal potential leakage of application layer information when the data sent by the application are not greedy. Note the initial packet must have a minimum size of 1200 bytes according to the QUIC specification. A receiver of a smaller initial packet may reject this packet in order to avoid amplification attacks.

6. Port Selection and Application Endpoint Discovery

In general, port numbers serves 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]. Note that 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 as well as NATs.

As QUIC is a general purpose transport protocol, there are no requirements that servers use a particular UDP port for QUIC in general. For applications with a fallback to TCP which do not

already have an alternate mapping to UDP, the registration (if necessary) and use of the UDP port number corresponding to the TCP port already registered for the application is RECOMMENDED. 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 SHOULD define an alternate endpoint discovery mechanism to allow the usage of ports other than the default. For example, HTTP/3 ([QUIC-HTTP] sections 3.2 and 3.3) specifies the use of ALPN [RFC7301] for service discovery which allows the server to use and announce a different port number. Note that HTTP/3's ALPN token ("h3") identifies not only the version of the application protocol, but also the binding to QUIC as well as the version of QUIC itself; this approach allows unambiguous agreement between the endpoints on the protocol stack in use.

Note that 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 may lead to blocking or other interference by network elements such as firewalls that rely on the port number for application identification.

7. Connection Migration

QUIC supports connection migration. Even if lower-layer addresses (usually the 4-tuple of IP addresses and ports) changes, QUIC packets can still be associated with an existing connection based on the Connection ID (see also section Section 9) in the QUIC header, if present. This supports cases where address information changes due to e.g. NAT rebinding or change of the local interface. Currently QUIC only supports failover cases. Only one "path" can be used at a time, and as soon as the new path is validated all traffic will be switched over to the next path. Of course if an endpoint decided to not use the Connection ID in short packets (Zero-length Conn ID) for a certain connection, migration is not supported for that direction of the connection.

8. Connection closure

QUIC connections are closed either by expiration of an idle timeout 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 immidate close will emit an CONNECTION_CLOSE frame. This frames 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, e.g. see [QUIC-HTTP] section 8.1. In the case of a grateful shut-down initiated by the application after application layer negotiation, a NO_ERROR code is expected. Further, the CONNECTION_CLOSE frame provides an optional reason field, that can be used to append human-readable information to an error code. Note that QUIC RESET_STREAM and STOP_SENDING frames provide similar capablities. Usually application error codes are defined to be applicabile to all three frames.

Alternatively, a QUIC connection will be silently closed by each endpoint separately after an idle timeout. The idle timeout is announce for each endpoint during connection established and should be accessible by the application. If an application desires to keep the connection open for longer than the announced timeout, it can send keep-alives messages. See {#resumption-v-keepalive} for further guidance.

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, because the information is intended to be used by the network. QUIC has a long header that is used during connection establishment and for other control processes, and a short header that may be used for data transmission in an established connection. While the long header always exposes some information (such as the version and Connection IDs), the short header exposes at most only a single Connection ID.

Note that the Connection ID in the short header may be omitted. This is a per-connection configuration option; if the Connection ID is not present, then the peer omitting the connection ID will use the same local address for the lifetime of the connection.

9.1. Server-Generated Connection ID

QUIC supports a server-generated Connection ID, transmitted to the client during connection establishment (see Section 6.1 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. Once the server generates a Connection ID that encodes its identity, every CDN load balancer would be able to forward the packets to that server without retaining connection state.

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

The best way to obscure an encoding is to appear random to observers, which is most rigorously achieved with encryption.

9.2. Mitigating Timing Linkability with Connection ID Migration

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 mitigation for these attacks is operational, either by using a load balancing architecture that loads more flows onto a single server-side address, by coordinating the timing of migrations to 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.

10. 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.

11. 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 validation of the negotiation version as though the new version were disabled.

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.

12. 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.

13. 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.

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.

14. Contributors

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

15. 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.

16. References

16.1. 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-29, 9 June 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-29.txt>.

[QUIC-INVARIANTS] Thomson, M., "Version-Independent Properties of QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-invariants-09, 9 June 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-invariants-09.txt>.

[QUIC-TLS] Thomson, M. and S. Turner, "Using TLS to Secure QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-tls-29, 9 June 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-tls-29.txt>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[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/info/rfc6335>.

[TLS13] Thomson, M. and S. Turner, "Using TLS to Secure QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-tls-29, 9 June 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-tls-29.txt>.

16.2. 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-RETRY]
Nottingham, M., "Retrying HTTP Requests", Work in Progress, Internet-Draft, draft-nottingham-httpbis-retry-01, 1 February 2017, <http://www.ietf.org/internet-drafts/draft-nottingham-httpbis-retry-01.txt>.

[I-D.nottingham-httpbis-retry]
Nottingham, M., "Retrying HTTP Requests", Work in Progress, Internet-Draft, draft-nottingham-httpbis-retry-01, 1 February 2017, <http://www.ietf.org/internet-drafts/draft-nottingham-httpbis-retry-01.txt>.

[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-29, 9 June 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-http-29.txt>.

[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>.

[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

QUIC
Internet-Draft
Intended status: Standards Track
Expires: 15 February 2021

M. Duke
F5 Networks, Inc.
N. Banks
Microsoft

14 August 2020

QUIC-LB: Generating Routable QUIC Connection IDs
draft-ietf-quic-load-balancers-04

Abstract

QUIC connection IDs allow continuation of connections across address/port 4-tuple changes, and can store routing information for stateless or low-state load balancers. They also can prevent linkability of connections across deliberate address migration through the use of protected communications between client and server. This creates issues for load-balancing intermediaries. This specification standardizes methods for encoding routing information given a small set of configuration parameters. This framework also enables offload of other QUIC functions to trusted intermediaries, given the explicit cooperation of the QUIC server.

Table of Contents

1. Introduction

QUIC packets [QUIC-TRANSPORT] usually contain a connection ID to allow endpoints to associate packets with different address/port 4-tuples to the same connection context. This feature makes connections robust in the event of NAT rebinding. QUIC endpoints usually designate the connection ID which peers use to address packets. Server-generated connection IDs create a potential need for out-of-band communication to support QUIC.

QUIC allows servers (or load balancers) to designate an initial connection ID to encode useful routing information for load balancers. It also encourages servers, in packets protected by cryptography, to provide additional connection IDs to the client. This allows clients that know they are going to change IP address or port to use a separate connection ID on the new path, thus reducing linkability as clients move through the world.

There is a tension between the requirements to provide routing information and mitigate linkability. Ultimately, because new connection IDs are in protected packets, they must be generated at the server if the load balancer does not have access to the connection keys. However, it is the load balancer that has the context necessary to generate a connection ID that encodes useful routing information. In the absence of any shared state between load balancer and server, the load balancer must maintain a relatively expensive table of server-generated connection IDs, and will not route packets correctly if they use a connection ID that was originally communicated in a protected NEW_CONNECTION_ID frame.

This specification provides common algorithms for encoding the server mapping in a connection ID given some shared parameters. The mapping is generally only discoverable by observers that have the parameters, preserving unlinkability as much as possible.

Aside from load balancing, a QUIC server may also desire to offload other protocol functions to trusted intermediaries. These intermediaries might include hardware assist on the server host itself, without access to fully decrypted QUIC packets. For example, this document specifies a means of offloading stateless retry to counter Denial of Service attacks. It also proposes a system for self-encoding connection ID length in all packets, so that crypto offload can consistently look up key information.

While this document describes a small set of configuration parameters to make the server mapping intelligible, the means of distributing these parameters between load balancers, servers, and other trusted intermediaries is out of its scope. There are numerous well-known infrastructures for distribution of configuration.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying significance described in RFC 2119.

In this document, "client" and "server" refer to the endpoints of a QUIC connection unless otherwise indicated. A "load balancer" is an intermediary for that connection that does not possess QUIC connection keys, but it may rewrite IP addresses or conduct other IP or UDP processing. A "configuration agent" is the entity that determines the QUIC-LB configuration parameters for the network and leverages some system to distribute that configuration.

Note that stateful load balancers that act as proxies, by terminating a QUIC connection with the client and then retrieving data from the server using QUIC or another protocol, are treated as a server with respect to this specification.

For brevity, "Connection ID" will often be abbreviated as "CID".

2. Protocol Objectives

2.1. Simplicity

QUIC is intended to provide unlinkability across connection migration, but servers are not required to provide additional connection IDs that effectively prevent linkability. If the coordination scheme is too difficult to implement, servers behind load balancers using connection IDs for routing will use trivially linkable connection IDs. Clients will therefore be forced to choose between terminating the connection during migration or remaining linkable, subverting a design objective of QUIC.

The solution should be both simple to implement and require little additional infrastructure for cryptographic keys, etc.

2.2. Security

In the limit where there are very few connections to a pool of servers, no scheme can prevent the linking of two connection IDs with high probability. In the opposite limit, where all servers have many connections that start and end frequently, it will be difficult to associate two connection IDs even if they are known to map to the same server.

QUIC-LB is relevant in the region between these extremes: when the information that two connection IDs map to the same server is helpful to linking two connection IDs. Obviously, any scheme that transparently communicates this mapping to outside observers compromises QUIC’s defenses against linkability.

Though not an explicit goal of the QUIC-LB design, concealing the server mapping also complicates attempts to focus attacks on a specific server in the pool.

3. First CID octet

The first octet of a Connection ID is reserved for two special purposes, one mandatory (config rotation) and one optional (length self-description).

Subsequent sections of this document refer to the contents of this octet as the "first octet."

3.1. Config Rotation

The first two bits of any connection ID MUST encode an identifier for the configuration that the connection ID uses. This enables incremental deployment of new QUIC-LB settings (e.g., keys).

When new configuration is distributed to servers, there will be a transition period when connection IDs reflecting old and new configuration coexist in the network. The rotation bits allow load balancers to apply the correct routing algorithm and parameters to incoming packets.

Configuration Agents SHOULD deliver new configurations to load balancers before doing so to servers, so that load balancers are ready to process CIDs using the new parameters when they arrive.

A Configuration Agent SHOULD NOT use a codepoint to represent a new configuration until it takes precautions to make sure that all connections using CIDs with an old configuration at that codepoint have closed or transitioned.

Servers MUST NOT generate new connection IDs using an old configuration after receiving a new one from the configuration agent. Servers MUST send NEW_CONNECTION_ID frames that provide CIDs using the new configuration, and retire CIDs using the old configuration using the "Retire Prior To" field of that frame.

It also possible to use these bits for more long-lived distinction of different configurations, but this has privacy implications (see Section 10.3).

3.2. Configuration Failover

If a server has not received a valid QUIC-LB configuration, and believes that low-state, Connection-ID aware load balancers are in the path, it SHOULD generate connection IDs with the config rotation bits set to '11' and SHOULD use the "disable_active_migration" transport parameter in all new QUIC connections. It SHOULD NOT send NEW_CONNECTION_ID frames with new values.

A load balancer that sees a connection ID with config rotation bits set to '11' MUST revert to 5-tuple routing.

3.3. Length Self-Description

Local hardware cryptographic offload devices may accelerate QUIC servers by receiving keys from the QUIC implementation indexed to the connection ID. However, on physical devices operating multiple QUIC servers, it is impractical to efficiently lookup these keys if the connection ID does not self-encode its own length.

Note that this is a function of particular server devices and is irrelevant to load balancers. As such, load balancers MAY omit this from their configuration. However, the remaining 6 bits in the first octet of the Connection ID are reserved to express the length of the following connection ID, not including the first octet.

A server not using this functionality SHOULD make the six bits appear to be random.

3.4. Format

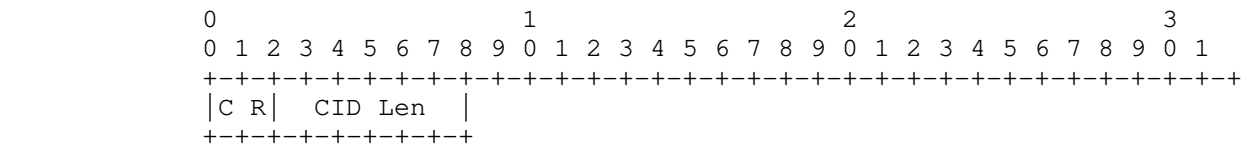


Figure 1: First Octet Format

The first octet has the following fields:

CR: Config Rotation bits.

CID Len: Length Self-Description (if applicable). Encodes the length of the Connection ID following the First Octet.

4. Routing Algorithms

In QUIC-LB, load balancers do not generate individual connection IDs to servers. Instead, they communicate the parameters of an algorithm to generate routable connection IDs.

The algorithms differ in the complexity of configuration at both load balancer and server. Increasing complexity improves obfuscation of the server mapping.

As clients sometimes generate the DCIDs in long headers, these might not conform to the expectations of the routing algorithm. These are called "non-compliant DCIDs":

* The DCID might not be long enough for the routing algorithm to process.

* The extracted server mapping might not correspond to an active server.

Load balancers MUST forward packets with long headers with non-compliant DCIDs to an active server using an algorithm of its own choosing. It need not coordinate this algorithm with the servers. The algorithm SHOULD be deterministic over short time scales so that related packets go to the same server. The design of this algorithm SHOULD consider the version-invariant properties of QUIC described in [QUIC-INVARIANTS] to maximize its robustness to future versions of QUIC. For example, a non-compliant DCID might be converted to an integer and divided by the number of servers, with the modulus used to forward the packet. The number of servers is usually consistent on the time scale of a QUIC connection handshake. See also Section 9.

As a partial exception to the above, load balancers MAY drop packets with long headers and non-compliant DCIDs if and only if it knows that the encoded QUIC version does not allow a non-compliant DCID in a packet with that signature. For example, a load balancer can safely drop a QUIC version 1 Handshake packet with a non-compliant DCIDs. The prohibition against dropping packets with long headers remains for unknown QUIC versions.

Load balancers SHOULD drop packets with non-compliant DCIDs in a short header.

A QUIC-LB configuration MAY significantly over-provision the server ID space (i.e., provide far more codepoints than there are servers) to increase the probability that a randomly generated Destination Connection ID is non- compliant.

Load balancers MUST forward packets with compliant DCIDs to a server in accordance with the chosen routing algorithm.

The load balancer MUST NOT make the routing behavior dependent on any bits in the first octet of the QUIC packet header, except the first bit, which indicates a long header. All other bits are QUIC version-dependent and intermediaries would cannot build their design on version-specific templates.

There are situations where a server pool might be operating two or more routing algorithms or parameter sets simultaneously. The load balancer uses the first two bits of the connection ID to multiplex incoming DCIDs over these schemes.

This section describes three participants: the configuration agent, the load balancer, and the server.

4.1. Plaintext CID Algorithm

The Plaintext CID Algorithm makes no attempt to obscure the mapping of connections to servers, significantly increasing linkability. The format is depicted in the figure below.

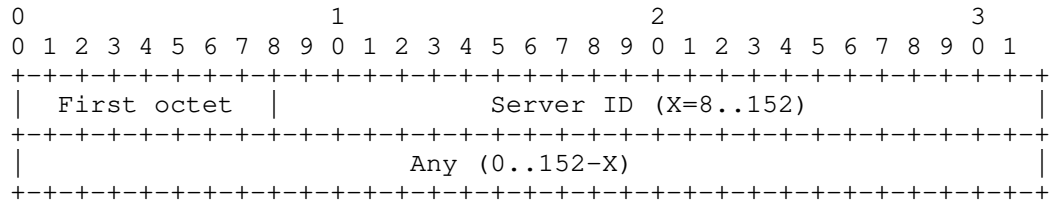


Figure 2: Plaintext CID Format

4.1.1. Configuration Agent Actions

The configuration agent selects a number of bytes of the server connection ID to encode individual server IDs, called the "routing bytes". The number of bytes MUST have enough entropy to have a different code point for each server.

It also assigns a server ID to each server.

4.1.2. Load Balancer Actions

On each incoming packet, the load balancer extracts consecutive octets, beginning with the second octet. These bytes represent the server ID.

4.1.3. Server Actions

The server chooses a connection ID length. This MUST be at least one byte longer than the routing bytes.

When a server needs a new connection ID, it encodes its assigned server ID in consecutive octets beginning with the second. All other bits in the connection ID, except for the first octet, MAY be set to any other value. These other bits SHOULD appear random to observers.

4.2. Stream Cipher CID Algorithm

The Stream Cipher CID algorithm provides cryptographic protection at the cost of additional per-packet processing at the load balancer to decrypt every incoming connection ID. The CID format is depicted below.

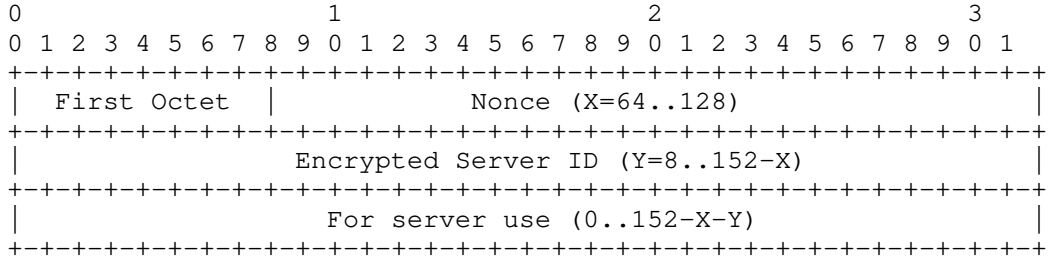


Figure 3: Stream Cipher CID Format

4.2.1. Configuration Agent Actions

The configuration agent assigns a server ID to every server in its pool, and determines a server ID length (in octets) sufficiently large to encode all server IDs, including potential future servers.

The configuration agent also selects a nonce length and an 16-octet AES-ECB key to use for connection ID decryption. The nonce length MUST be at least 8 octets and no more than 16 octets. The nonce length and server ID length MUST sum to 19 or fewer octets.

4.2.2. Load Balancer Actions

Upon receipt of a QUIC packet, the load balancer extracts as many of the earliest octets from the destination connection ID as necessary to match the nonce length. The server ID immediately follows.

The load balancer decrypts the nonce and the server ID using the following three pass algorithm:

* Pass 1: The load balancer decrypts the server ID using 128-bit AES Electronic Codebook (ECB) mode, much like QUIC header protection. The encrypted nonce octets are zero-padded to 16 octets. AES-ECB encrypts this encrypted nonce using its key to generate a mask which it applies to the encrypted server id. This provides an intermediate value of the server ID, referred to as server-id intermediate.

server_id_intermediate = encrypted_server_id ^ AES-ECB(key, padded-encrypted-nonce)

* Pass 2: The load balancer decrypts the nonce octets using 128-bit AES ECB mode, using the server-id intermediate as "nonce" for this pass. The server-id intermediate octets are zero-padded to 16 octets. AES-ECB encrypts this padded server-id intermediate using its key to generate a mask which it applies to the encrypted nonce. This provides the decrypted nonce value.

nonce = encrypted_nonce ^ AES-ECB(key, padded-server_id_intermediate)

* Pass 3: The load balancer decrypts the server ID using 128-bit AES ECB mode. The nonce octets are zero-padded to 16 octets. AES-ECB encrypts this nonce using its key to generate a mask which it applies to the intermediate server id. This provides the decrypted server ID.

server_id = server_id_intermediate ^ AES-ECB(key, padded-nonce)

For example, if the nonce length is 10 octets and the server ID length is 2 octets, the connection ID can be as small as 13 octets. The load balancer uses the the second through eleventh octets of the connection ID for the nonce, zero-pads it to 16 octets, uses xors the result with the twelfth and thirteenth octet. The result is padded with 14 octets of zeros and encrypted to obtain a mask that is xored with the nonce octets. Finally, the nonce octets are padded with six octets of zeros, encrypted, and the first two octets xored with the server ID octets to obtain the actual server ID.

This three-pass algorithm is a simplified version of the FFX algorithm, with the property that each encrypted nonce value depends on all server ID bits, and each encrypted server ID bit depends on all nonce bits and all server ID bits. This mitigates attacks against stream ciphers in which attackers simply flip encrypted server-ID bits.

The output of the decryption is the server ID that the load balancer uses for routing.

4.2.3. Server Actions

When generating a routable connection ID, the server writes arbitrary bits into its nonce octets, and its provided server ID into the server ID octets. Servers MAY opt to have a longer connection ID beyond the nonce and server ID. The additional bits MAY encode additional information, but SHOULD appear essentially random to observers.

If the decrypted nonce bits increase monotonically, that guarantees that nonces are not reused between connection IDs from the same server.

The server encrypts the server ID using exactly the algorithm as described in Section 4.2.2, performing the three passes in reverse order.

4.3. Block Cipher CID Algorithm

The Block Cipher CID Algorithm, by using a full 16 octets of plaintext and a 128-bit cipher, provides higher cryptographic protection and detection of non-compliant connection IDs. However, it also requires connection IDs of at least 17 octets, increasing overhead of client-to-server packets.

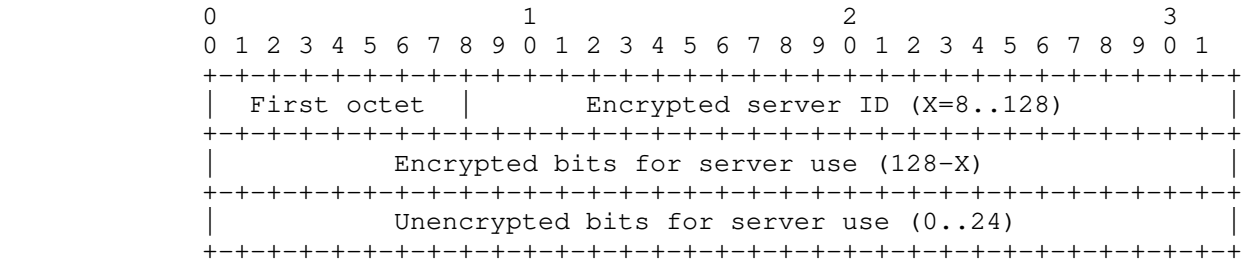


Figure 4: Block Cipher CID Format

4.3.1. Configuration Agent Actions

The configuration agent assigns a server ID to every server in its pool, and determines a server ID length (in octets) sufficiently large to encode all server IDs, including potential future servers. The server ID will start in the second octet of the decrypted connection ID and occupy continuous octets beyond that.

They server ID length MUST be no more than 16 octets and SHOULD sum to no more than 12 octets, to provide servers adequate space to encode their own opaque data.

The configuration agent also selects an 16-octet AES-ECB key to use for connection ID decryption.

4.3.2. Load Balancer Actions

Upon receipt of a QUIC packet, the load balancer reads the first octet to obtain the config rotation bits. It then decrypts the subsequent 16 octets using AES-ECB decryption and the chosen key.

The decrypted plaintext contains the server id and opaque server data in that order. The load balancer uses the server ID octets for routing.

4.3.3. Server Actions

When generating a routable connection ID, the server MUST choose a connection ID length between 17 and 20 octets. The server writes its provided server ID into the server ID octets and arbitrary bits into the remaining bits. These arbitrary bits MAY encode additional information. Bits in the eighteenth, nineteenth, and twentieth octets SHOULD appear essentially random to observers. The first octet is reserved as described in Section 3.

The server then encrypts the second through seventeenth octets using the 128-bit AES-ECB cipher.

5. ICMP Processing

For protocols where 4-tuple load balancing is sufficient, it is straightforward to deliver ICMP packets from the network to the correct server, by reading the IP and transport-layer headers to obtain the 4-tuple. When routing is based on connection ID, further measures are required, as most QUIC packets that trigger ICMP responses will only contain a client-generated connection ID that contains no routing information.

To solve this problem, load balancers MAY maintain a mapping of Client IP and port to server ID based on recently observed packets.

Alternatively, servers MAY implement the technique described in Section 14.4.1 of [QUIC-TRANSPORT] to increase the likelihood a Source Connection ID is included in ICMP responses to Path Maximum Transmission Unit (PMTU) probes. Load balancers MAY parse the echoed packet to extract the Source Connection ID, if it contains a QUIC long header, and extract the Server ID as if it were in a Destination CID.

6. Retry Service

When a server is under load, QUICv1 allows it to defer storage of connection state until the client proves it can receive packets at its advertised IP address. Through the use of a Retry packet, a token in subsequent client Initial packets, and the original_destination_connection_id transport parameter, servers verify address ownership and clients verify that there is no "man in the middle" generating Retry packets.

As a trusted Retry Service is literally a "man in the middle," the service must communicate the original_destination_connection_id back to the server so that it can pass client verification. It also must either verify the address itself (with the server trusting this verification) or make sure there is common context for the server to verify the address using a service-generated token.

The service must also communicate the source connection ID of the Retry packet to the server so that it can include it in a transport parameter for client verification.

There are two different mechanisms to allow offload of DoS mitigation to a trusted network service. One requires no shared state; the server need only be configured to trust a retry service, though this imposes other operational constraints. The other requires shared key, but has no such constraints.

Retry services MUST forward all QUIC packets that are not of type Initial or 0-RTT. Other packets types might involve changed IP addresses or connection IDs, so it is not practical for Retry Services to identify such packets as valid or invalid.

6.1. Common Requirements

Regardless of mechanism, a retry service has an active mode, where it is generating Retry packets, and an inactive mode, where it is not, based on its assessment of server load and the likelihood an attack is underway. The choice of mode MAY be made on a per-packet or per-connection basis, through a stochastic process or based on client address.

A retry service MUST forward all packets for a QUIC version it does not understand. Note that if servers support versions the retry service does not, this may increase load on the servers. However, dropping these packets would introduce chokepoints to block deployment of new QUIC versions. Note that future versions of QUIC might not have Retry packets, require different information in Retry, or use different packet type indicators.

6.2. No-Shared-State Retry Service

The no-shared-state retry service requires no coordination, except that the server must be configured to accept this service and know which QUIC versions the retry service supports. The scheme uses the first bit of the token to distinguish between tokens from Retry packets (codepoint '0') and tokens from NEW_TOKEN frames (codepoint '1').

6.2.1. Configuration Agent Actions

The configuration agent distributes a list of QUIC versions to be served by the Retry Service.

6.2.2. Service Requirements

A no-shared-state retry service MUST be present on all paths from potential clients to the server. These paths MUST fail to pass QUIC traffic should the service fail for any reason. That is, if the service is not operational, the server MUST NOT be exposed to client traffic. Otherwise, servers that have already disabled their Retry capability would be vulnerable to attack.

The path between service and server MUST be free of any potential attackers. Note that this and other requirements above severely restrict the operational conditions in which a no-shared-state retry service can safely operate.

Retry tokens generated by the service MUST have the format below.

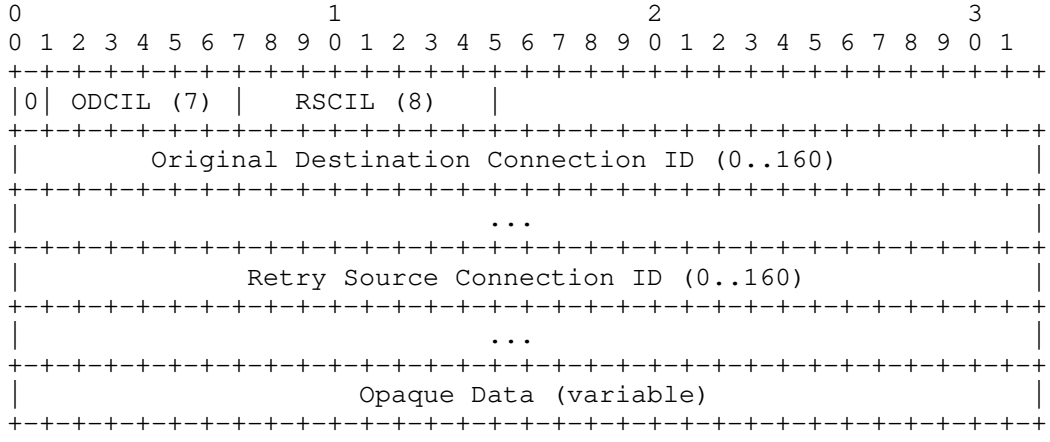


Figure 5: Format of non-shared-state retry service tokens

The first bit of retry tokens generated by the service MUST be zero. The token has the following additional fields:

ODCIL: The length of the original destination connection ID from the triggering Initial packet. This is in cleartext to be readable for the server, but authenticated later in the token.

RSCIL: The retry source connection ID length.

Original Destination Connection ID: This also in cleartext and authenticated later.

Retry Source Connection ID: This also in cleartext and authenticated later.

Opaque Data: This data MUST contain encrypted information that allows the retry service to validate the client's IP address, in accordance with the QUIC specification. It MUST also provide a cryptographically secure means to validate the integrity of the entire token.

Upon receipt of an Initial packet with a token that begins with '0', the retry service MUST validate the token in accordance with the QUIC specification.

In active mode, the service MUST issue Retry packets for all Client initial packets that contain no token, or a token that has the first bit set to '1'. It MUST NOT forward the packet to the server. The service MUST validate all tokens with the first bit set to '0'. If successful, the service MUST forward the packet with the token intact. If unsuccessful, it MUST drop the packet. The Retry Service MAY send an Initial Packet containing a CONNECTION_CLOSE frame with the INVALID_TOKEN error code when dropping the packet.

Note that this scheme has a performance drawback. When the retry service is in active mode, clients with a token from a NEW_TOKEN frame will suffer a 1-RTT penalty even though it has proof of address with its token.

In inactive mode, the service MUST forward all packets that have no token or a token with the first bit set to '1'. It MUST validate all tokens with the first bit set to '0'. If successful, the service MUST forward the packet with the token intact. If unsuccessful, it MUST either drop the packet or forward it with the token removed. The latter requires decryption and re-encryption of the entire Initial packet to avoid authentication failure. Forwarding the packet causes the server to respond without the original_destination_connection_id transport parameter, which preserves the normal QUIC signal to the client that there is an unauthorized man in the middle.

6.2.3. Server Requirements

A server behind a non-shared-state retry service MUST NOT send Retry packets for a QUIC version the retry service understands. It MAY send Retry for QUIC versions the Retry Service does not understand.

Tokens sent in NEW_TOKEN frames MUST have the first bit be set to '1'.

If a server receives an Initial Packet with the first bit set to '1', it could be from a server-generated NEW_TOKEN frame and should be processed in accordance with the QUIC specification. If a server receives an Initial Packet with the first bit to '0', it is a Retry token and the server MUST NOT attempt to validate it. Instead, it MUST assume the address is validated and MUST extract the Original Destination Connection ID and Retry Source Connection ID, assuming the format described in Section 6.2.2.

6.3. Shared-State Retry Service

A shared-state retry service uses a shared key, so that the server can decode the service's retry tokens. It does not require that all traffic pass through the Retry service, so servers MAY send Retry packets in response to Initial packets that don't include a valid token.

Both server and service must have access to Universal time, though tight synchronization is not necessary.

All tokens, generated by either the server or retry service, MUST use the following format. This format is the cleartext version. On the wire, these fields are encrypted using an AES-ECB cipher and the token key. If the token is not a multiple of 16 octets, the last block is padded with zeroes.

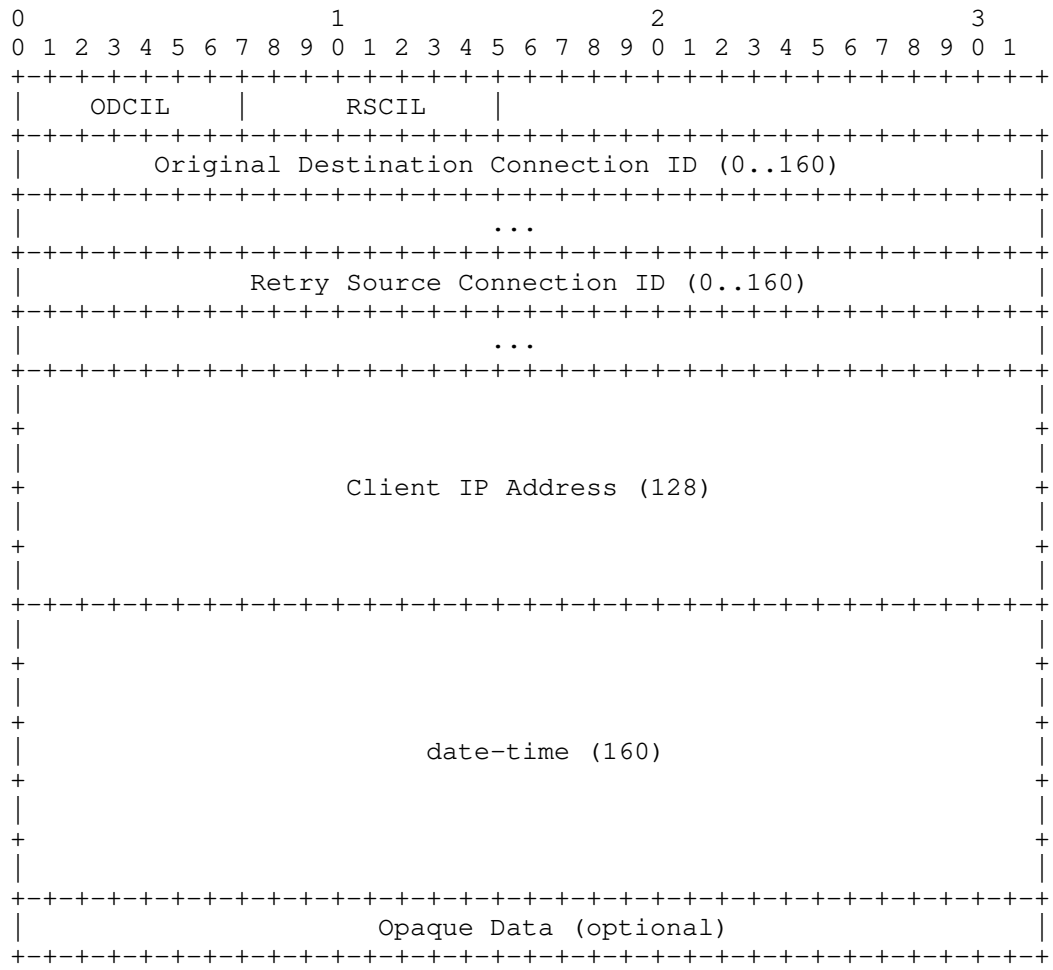


Figure 6: Cleartext format of shared-state retry tokens

The tokens have the following fields:

ODCIL: The original destination connection ID length. Tokens in NEW_TOKEN frames MUST set this field to zero.

RSCIL: The retry source connection ID length. Tokens in NEW_TOKEN frames MUST set this field to zero.

Original Destination Connection ID: The server or Retry Service copies this from the field in the client Initial packet.

Retry Source Connection ID: The server or Retry service copies this from the Source Connection ID of the Retry packet.

Client IP Address: The source IP address from the triggering Initial packet. The client IP address is 16 octets. If an IPv4 address, the last 12 octets are zeroes.

date-time: The date-time string is a total of 20 octets and encodes the time the token was generated. The format of date-time is described in Section 5.6 of [RFC3339]. This ASCII field MUST use the "Z" character for time-offset.

Opaque Data: The server may use this field to encode additional information, such as congestion window, RTT, or MTU. Opaque data SHOULD also allow servers to distinguish between retry tokens (which trigger use of the original_destination_connection_id transport parameter) and NEW_TOKEN frame tokens.

6.3.1. Configuration Agent Actions

The configuration agent generates and distributes a "token key."

6.3.2. Service Requirements

When in active mode, the service MUST generate Retry tokens with the format described above when it receives a client Initial packet with no token.

In active mode, the service SHOULD decrypt incoming tokens. The service SHOULD drop packets with an IP address that does not match, and SHOULD forward packets that do, regardless of the other fields.

In inactive mode, the service SHOULD forward all packets to the server so that the server can issue an up-to-date token to the client.

6.3.3. Server Requirements

The server MUST validate all tokens that arrive in Initial packets, as they may have bypassed the Retry service. It SHOULD use the date-time field to apply its expiration limits for tokens. This need not be synchronized with the retry service. However, servers MAY allow retry tokens marked as being a few seconds in the future, due to possible clock synchronization issues.

After decrypting the token, the server uses the corresponding fields to populate the original_destination_connection_id transport parameter, with a length equal to ODCIL, and the retry_source_connection_id transport parameter, with length equal to RSCIL.

As discussed in [QUIC-TRANSPORT], a server MUST NOT send a Retry packet in response to an Initial packet that contains a retry token.

7. Configuration Requirements

QUIC-LB requires common configuration to synchronize understanding of encodings and guarantee explicit consent of the server.

The load balancer and server MUST agree on a routing algorithm and the relevant parameters for that algorithm.

For Plaintext CID Routing, this consists of the Server ID and the routing bytes. The Server ID is unique to each server, and the routing bytes are global.

For Stream Cipher CID Routing, this consists of the Server ID, Server ID Length, Key, and Nonce Length. The Server ID is unique to each server, but the others MUST be global. The authentication token MUST be distributed out of band for this algorithm to operate.

For Block Cipher CID Routing, this consists of the Server ID, Server ID Length, Key, and Zero-Padding Length. The Server ID is unique to each server, but the others MUST be global.

A full QUIC-LB configuration MUST also specify the information content of the first CID octet and the presence and mode of any Retry Service.

The following pseudocode describes the data items necessary to store a full QUIC-LB configuration at the server. It is meant to describe the conceptual range and not specify the presentation of such configuration in an internet packet. The comments signify the range of acceptable values where applicable.

```
    uint2    config_rotation_bits;
    boolean  first_octet_encodes_cid_length;
    enum     { none, non_shared_state, shared_state } retry_service;
    select (retry_service) {
        case none: null;
        case non_shared_state: uint32 list_of_quic_versions[];
        case shared_state:     uint8 key[16];
    } retry_service_config;
    enum     { none, plaintext, stream_cipher, block_cipher }
                     routing_algorithm;
    select (routing_algorithm) {
        case none: null;
        case plaintext: struct {
            uint8 server_id_length; /* 1..19 */
            uint8 server_id[server_id_length];
        } plaintext_config;
        case stream_cipher: struct {
            uint8  nonce_length; /* 8..16 */
            uint8  server_id_length; /* 1..(19 - nonce_length) */
            uint8  server_id[server_id_length];
            uint8  key[16];
        } stream_cipher_config;
        case block_cipher: struct {
            uint8  server_id_length;
            uint8  server_id[server_id_length];
            uint8  key[16];
        } block_cipher_config;
   } routing_algorithm_config;
```

8. Additional Use Cases

This section discusses considerations for some deployment scenarios not implied by the specification above.

8.1. Load balancer chains

Some network architectures may have multiple tiers of low-state load balancers, where a first tier of devices makes a routing decision to the next tier, and so on until packets reach the server. Although QUIC-LB is not explicitly designed for this use case, it is possible to support it.

If each load balancer is assigned a range of server IDs that is a subset of the range of IDs assigned to devices that are closer to the client, then the first devices to process an incoming packet can extract the server ID and then map it to the correct forwrading address. Note that this solution is extensible to arbitrarily large numbers of load-balancing tiers, as the maximum server ID space is quite large.

8.2. Moving connections between servers

Some deployments may transparently move a connection from one server to another. The means of transferring connection state between servers is out of scope of this document.

To support a handover, a server involved in the transition could issue CIDs that map to the new server via a NEW_CONNECTION_ID frame, and retire CIDs associated with the new server using the "Retire Prior To" field in that frame.

Alternately, if the old server is going offline, the load balancer could simply map its server ID to the new server's address.

9. Version Invariance of QUIC-LB

Retry Services are inherently dependent on the format (and existence) of Retry Packets in each version of QUIC, and so Retry Service configuration explicitly includes the supported QUIC versions.

The server ID encodings, and requirements for their handling, are designed to be QUIC version independent (see [QUIC-INVARIANTS]). A QUIC-LB load balancer will generally not require changes as servers deploy new versions of QUIC. However, there are several unlikely future design decisions that could impact the operation of QUIC-LB.

The maximum Connection ID length could be below the minimum necessary for one or more encoding algorithms.

Section 4 provides guidance about how load balancers should handle non-compliant DCIDs. This guidance, and the implementation of an algorithm to handle these DCIDs, rests on some assumptions:

* Incoming short headers do not contain DCIDs that are client-generated.

* The use of client-generated incoming DCIDs does not persist beyond a few round trips in the connection.

* While the client is using DCIDs it generated, some exposed fields (IP address, UDP port, client-generated destination Connection ID) remain constant for all packets sent on the same connection.

While this document does not update the commitments in [QUIC-INVARIANTS], the additional assumptions are minimal and narrowly scoped, and provide a likely set of constants that load balancers can use with minimal risk of version- dependence.

If these assumptions are invalid, this specification is likely to lead to loss of packets that contain non-compliant DCIDs, and in extreme cases connection failure.

10. Security Considerations

QUIC-LB is intended to prevent linkability. Attacks would therefore attempt to subvert this purpose.

Note that the Plaintext CID algorithm makes no attempt to obscure the server mapping, and therefore does not address these concerns. It exists to allow consistent CID encoding for compatibility across a network infrastructure, which makes QUIC robust to NAT rebinding. Servers that are running the Plaintext CID algorithm SHOULD only use it to generate new CIDs for the Server Initial Packet and SHOULD NOT send CIDs in QUIC NEW_CONNECTION_ID frames, except that it sends one new Connection ID in the event of config rotation Section 3.1. Doing so might falsely suggest to the client that said CIDs were generated in a secure fashion.

A linkability attack would find some means of determining that two connection IDs route to the same server. As described above, there is no scheme that strictly prevents linkability for all traffic patterns, and therefore efforts to frustrate any analysis of server ID encoding have diminishing returns.

10.1. Attackers not between the load balancer and server

Any attacker might open a connection to the server infrastructure and aggressively simulate migration to obtain a large sample of IDs that map to the same server. It could then apply analytical techniques to try to obtain the server encoding.

The Stream and Block Cipher CID algorithms provide robust protection against any sort of linkage. The Plaintext CID algorithm makes no attempt to protect this encoding.

Were this analysis to obtain the server encoding, then on-path observers might apply this analysis to correlating different client IP addresses.

10.2. Attackers between the load balancer and server

Attackers in this privileged position are intrinsically able to map two connection IDs to the same server. The QUIC-LB algorithms do prevent the linkage of two connection IDs to the same individual connection if servers make reasonable selections when generating new IDs for that connection.

10.3. Multiple Configuration IDs

During the period in which there are multiple deployed configuration IDs (see Section 3.1), there is a slight increase in linkability. The server space is effectively divided into segments with CIDs that have different config rotation bits. Entities that manage servers SHOULD strive to minimize these periods by quickly deploying new configurations across the server pool.

10.4. Limited configuration scope

A simple deployment of QUIC-LB in a cloud provider might use the same global QUIC-LB configuration across all its load balancers that route to customer servers. An attacker could then simply become a customer, obtain the configuration, and then extract server IDs of other customers' connections at will.

To avoid this, the configuration agent SHOULD issue QUIC-LB configurations to mutually distrustful servers that have different keys for encryption algorithms. The load balancers can distinguish these configurations by external IP address, or by assigning different values to the config rotation bits (Section 3.1). Note that either solution has a privacy impact; see Section 10.3.

These techniques are not necessary for the plaintext algorithm, as it does not attempt to conceal the server ID.

10.5. Stateless Reset Oracle

Section 21.9 of [QUIC-TRANSPORT] discusses the Stateless Reset Oracle attack. For a server deployment to be vulnerable, an attacking client must be able to cause two packets with the same Destination CID to arrive at two different servers that share the same cryptographic context for Stateless Reset tokens. As QUIC-LB requires deterministic routing of DCIDs over the life of a connection, it is a sufficient means of avoiding an Oracle without additional measures.

11. IANA Considerations

There are no IANA requirements.

12. References

12.1. Normative References

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

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

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://www.rfc-editor.org/info/rfc3339>.

12.2. Informative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

Appendix A. Load Balancer Test Vectors

Because any connection ID encoding in this specification includes many bits for server use without affecting extraction of the server ID, there are many possible connection IDs for any given set of parameters. However, every connection ID should result in a unique server ID. The following connection IDs can be used to verify that a load balancer implementation extracts the correct server ID.

A.1. Plaintext Connection ID Algorithm

TBD

A.2. Stream Cipher Connection ID Algorithm

cr_bits 0x0 length_self_encoding: y nonce_len 10 sid_len 1 key 9c46142f1597511357cf437841721d4b

cid 0b05be7bf896ed26cb4cc59a sid ab cid 0b43909398577dd7df1597d4 sid 37 cid 0bf85fa27034785803747464 sid 0e cid 0bc630c588fdecbfbdb62e61 sid 44 cid 0b8788901684f5d4e4dc6aeb sid 83

cr_bits 0x0 length_self_encoding: n nonce_len 9 sid_len 2 key 434ae6fbf36aca0773a6a75f10e3f747

cid 08644a29067622f363d4c83e sid 846a cid 234b2899f9b213a70abfe193 sid 4417 cid 3ff4ef53bbaad327c1e18fa5 sid 7554 cid 08a0eaf4cc08f184e6cf7743 sid b78a cid 3fb2f5cf1b3e08bf97709c42 sid ed7e

cr_bits 0x0 length_self_encoding: y nonce_len 12 sid_len 3 key 02e895bf84f6a80c3c7156da88a96755

cid 0f7405813570b8f9a6a10564d7b92834 sid 49023c cid 0f3bb656319c6af210239dcaef77d3b9 sid b0a8ce cid 0f3ae6d54ee97fc6907b5e2d60436caf sid 21f035 cid 0f4774918a6576c88f85829306f6450f sid 9e46ea cid 0f7467db6ca1eb4c185e642b0c9f8f44 sid c33db0

cr_bits 0x0 length_self_encoding: n nonce_len 11 sid_len 4 key ccb612da03f5dc205faf9b0b1d5429cb

cid 0c4b23e27639aef72f861ad2dce39d96 sid 125fdba1 cid 063ed9a173d22be11818b77a3bd5ec37 sid 0f3f82bc cid 1a14e39b0f6ca6a3a48f6fdd2083fa09 sid 05950af2 cid 36cb4df5a7776edb21ec87c35c24e988 sid 3cb80d59 cid 05749809112a91327fef4b3152335298 sid 4746cb79

cr_bits 0x0 length_self_encoding: y nonce_len 8 sid_len 5 key 625696d413ea1a352401afce6eec2432

cid 0d2a7b43eeaac8b36fce2c14ac96 sid 4b00da143a cid 0ddd6cdb6685e75b91f4a1bb0dde sid f9aa795663 cid 0d870ea4d173d29484e41ea4a189 sid e430dcfb3f cid 0df12abe175241b5ab035d23910f sid 8bc66a2596 cid 0d390df5de76903ca94b2e9daa49 sid 7637d0c172

A.3. Block Cipher Connection ID Algorithm

Like the previous section, the text below lists a set of load balancer configuration and 5 CIDs generated with that configuration.

cr_bits 0x0 length_self_encoding: y sid_len 1 zp_len 11 key 8c24cb9b9c3289b4ee63c3f3d7f93a9a

cid: 1378e44f874642624fa69e7b4aec15a2a678b8b5 sid: 48
cid: 13772c82fe8ce6a00813f76a211b730eb4b20363 sid: 66
cid: 135ccf507b1c209457f80df0217b9a1df439c4b2 sid: 30
cid: 13898459900426c073c66b1001c867f9098a7aab sid: fe
cid: 1397a18da00bf912f20049d9f0a007444f8b6699 sid: 30

cr_bits 0x0 length_self_encoding: n sid_len 2 zp_len 10 key cc7ec42794664a8428250c12a7fb16fa

cid: 0cb28bfc1f65c3de14752bc0fc734ef824ce8f78 sid: 33fa
cid: 2345e9fc7a7be55b4ba1ff6ffa04f3f5f8c67009 sid: ee47
cid: 0d32102be441600f608c95841fd40ce978aa7a02 sid: 0c8b
cid: 2e6bfc53c91c275019cd809200fa8e23836565ab sid: feca
cid: 29b87a902ed129c26f7e4e918a68703dc71a6e0a sid: 8941

cr_bits 0x1 length_self_encoding: y sid_len 3 zp_len 9 key 42e657946b96b7052ab8e6eeb863ee24

cid: 53c48f7884d73fd9016f63e50453bfd9bcfc637d sid: b46b68
cid: 53f45532f6a4f0e1757fa15c35f9a2ab0fcce621 sid: 2147b4
cid: 5361fd4bbcee881a637210f4fffc02134772cc76 sid: e4bf4b
cid: 53881ffde14e613ef151e50ba875769d6392809b sid: c2afee
cid: 53ad0d60204d88343492334e6c4c4be88d4a3add sid: ae0331

cr_bits 0x0 length_self_encoding: n sid_len 4 zp_len 8 key ee2dc6a3359a94b0043ca0c82715ce71

cid: 058b9da37f436868cca3cef40c7f98001797c611 sid: eaf846c7
cid: 1259fc97439adaf87f61250afea059e5ddf66e44 sid: 4cc5e84a
cid: 202f424376f234d5f014f41cebc38de2619c6c71 sid: f94ff800
cid: 146ac3e4bbb750d3bfb617ef4b0cb51a1cae5868 sid: c2071b1b
cid: 36dfe886538af7eb16a196935b3705c9d741479f sid: 26359dbb

cr_bits 0x2 length_self_encoding: y sid_len 5 zp_len 7 key 700837da8834840afe7720186ec610c9

cid: 931ef3cc07e2eaf08d4c1902cd564d907cc3377c sid: 759b1d419a
cid: 9398c3d0203ab15f1dfeb5aa8f81e52888c32008 sid: 77cc0d3310
cid: 93f4ba09ab08a9ef997db4fa37a97dbf2b4c5481 sid: f7db9dce32
cid: 93744f4bedf95e04dd6607592ecf775825403093 sid: e264d714d2
cid: 93256308e3d349f8839dec840b0a90c7e7a1fc20 sid: 618b07791f

Appendix B. Acknowledgments

Appendix C. Change Log

RFC Editor's Note: Please remove this section prior to publication of a final version of this document.

C.1. since-draft-ietf-quic-load-balancers-03

* Improved Config Rotation text
* Added stream cipher test vectors
* Deleted the Obfuscated CID algorithm

C.2. since-draft-ietf-quic-load-balancers-02

* Replaced stream cipher algorithm with three-pass version
* Updated Retry format to encode info for required TPs
* Added discussion of version invariance
* Cleaned up text about config rotation
* Added Reset Oracle and limited configuration considerations
* Allow dropped long-header packets for known QUIC versions

C.3. since-draft-ietf-quic-load-balancers-01

* Test vectors for load balancer decoding

* Deleted remnants of in-band protocol
* Light edit of Retry Services section
* Discussed load balancer chains

C.4. since-draft-ietf-quic-load-balancers-00

* Removed in-band protocol from the document

C.5. Since draft-duke-quic-load-balancers-06

* Switch to IETF WG draft.

C.6. Since draft-duke-quic-load-balancers-05

* Editorial changes
* Made load balancer behavior independent of QUIC version
* Got rid of token in stream cipher encoding, because server might not have it
* Defined "non-compliant DCID" and specified rules for handling them.
* Added psuedocode for config schema

C.7. Since draft-duke-quic-load-balancers-04

* Added standard for retry services

C.8. Since draft-duke-quic-load-balancers-03

* Renamed Plaintext CID algorithm as Obfuscated CID
* Added new Plaintext CID algorithm
* Updated to allow 20B CIDs
* Added self-encoding of CID length

C.9. Since draft-duke-quic-load-balancers-02

* Added Config Rotation
* Added failover mode

* Tweaks to existing CID algorithms
* Added Block Cipher CID algorithm
* Reformatted QUIC-LB packets

C.10. Since draft-duke-quic-load-balancers-01

* Complete rewrite
* Supports multiple security levels
* Lightweight messages

C.11. Since draft-duke-quic-load-balancers-00

* Converted to markdown
* Added variable length connection IDs

Authors' Addresses

Martin Duke
F5 Networks, Inc.

Email: martin.h.duke@gmail.com

Nick Banks
Microsoft

Email: nibanks@microsoft.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 9 January 2021

M. Kuehlewind
Ericsson
B. Trammell
Google

8 July 2020

Manageability of the QUIC Transport Protocol
draft-ietf-quic-manageability-07

Abstract

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

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 9 January 2021.

Table of Contents

1. Introduction

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

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

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

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

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

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

1.1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Features of the QUIC Wire Image

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

This document is focused on the protocol as presently defined in [QUIC-TRANSPORT] and [QUIC-TLS]. Non-invariant parts of the wire image as described herein and in those documents cannot be used to identify QUIC as a protocol, and cannot be relied upon to infer 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 us the Header Form bit, and indicates which type of header is present.

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

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

The following information is exposed in QUIC packet headers:

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

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

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

* version number: the version number present in the long header, and identifies the version used for that packet. Note that during Version Negotiation (see Section 2.8, and Section 17.2.1 of [QUIC-TRANSPORT], the version number field has a special value (0x00000000) that identifies the packet as a Version Negotiation packet.

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

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

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

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

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

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

2.2. Coalesced Packets

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

2.3. Use of Port Numbers

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

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

2.4. The QUIC handshake

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

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

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

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

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

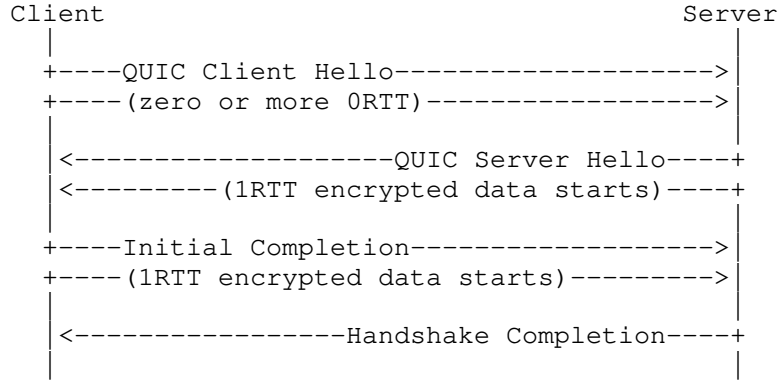


Figure 1: General communication pattern visible in the QUIC handshake

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

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

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

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

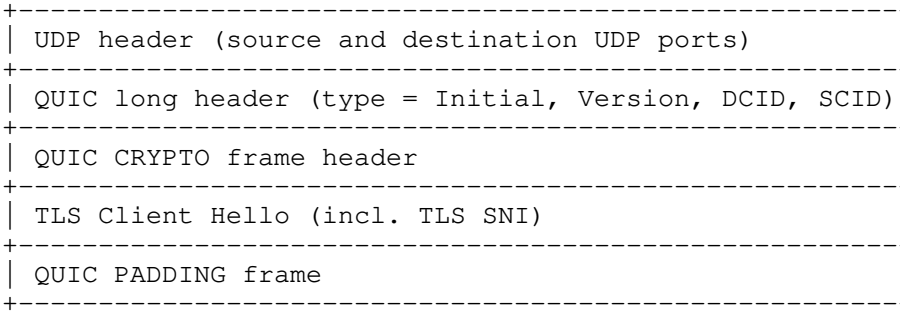


Figure 2: Typical 1-RTT QUIC Client Hello datagram pattern

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

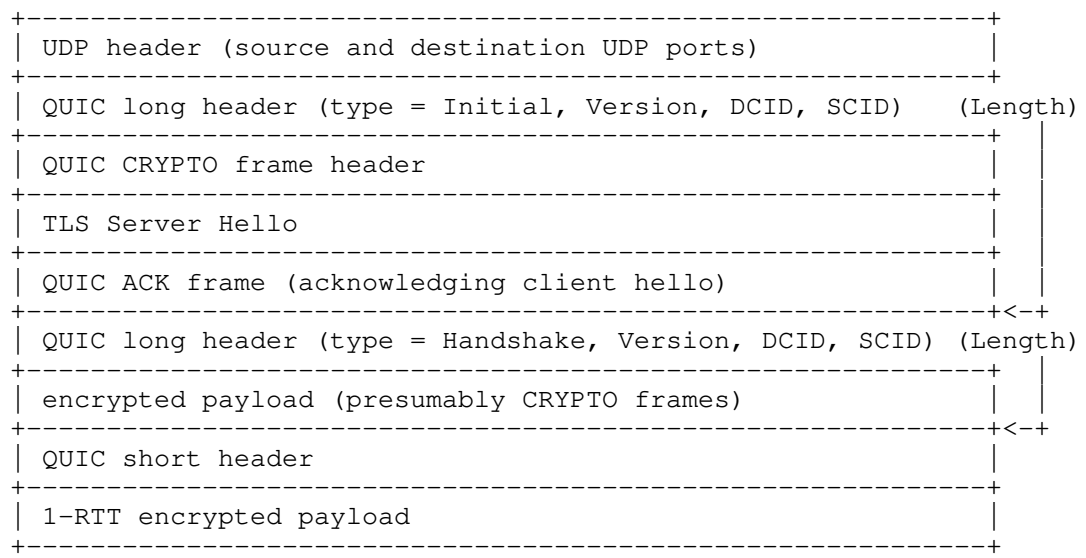


Figure 3: Typical QUIC Server Hello datagram pattern

The Server Hello datagram exposes version number, source and destination connection IDs, and information in the TLS Server Hello message.

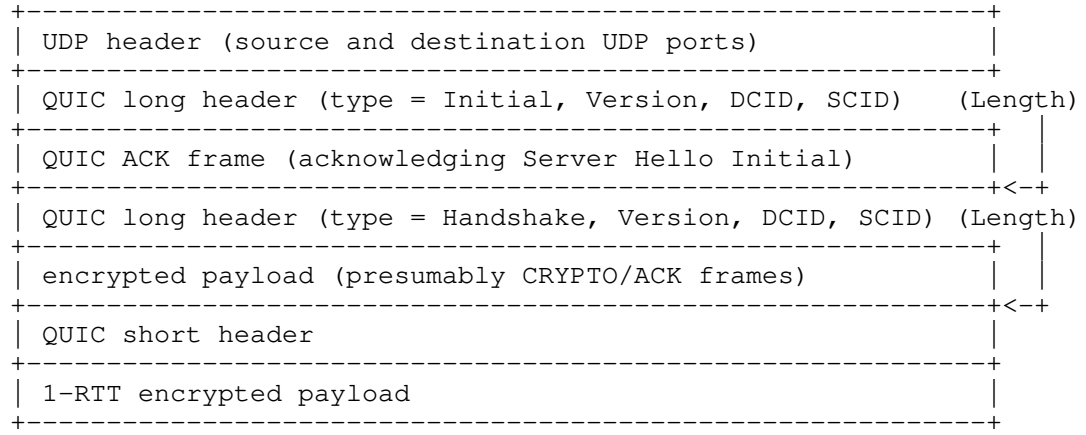


Figure 4: Typical QUIC Initial Completion datagram pattern

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

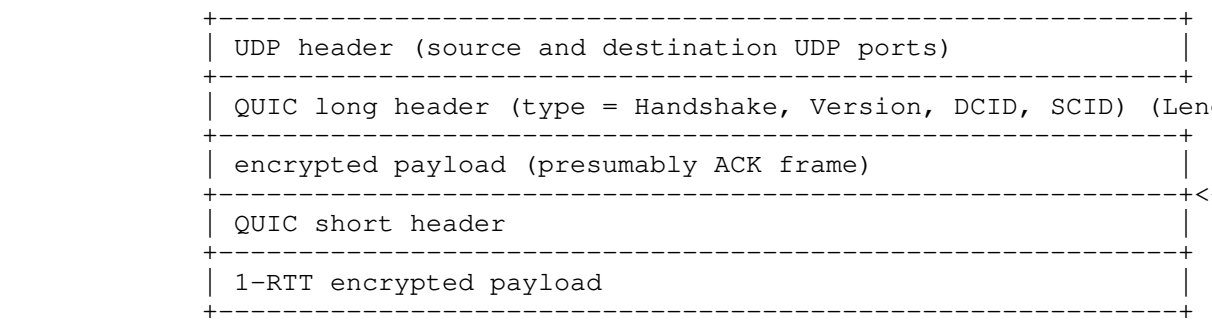


Figure 5: Typical QUIC Handshake Completion datagram pattern

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

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

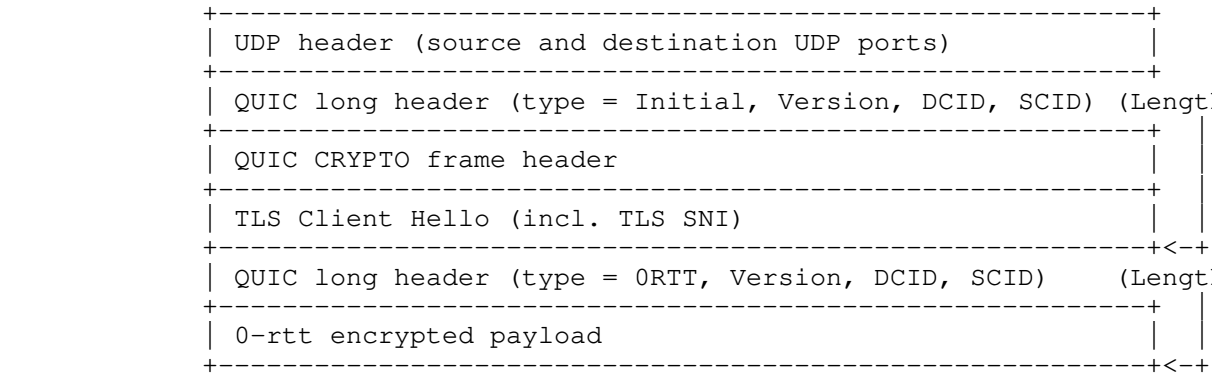


Figure 6: Typical 0-RTT QUIC Client Hello datagram pattern

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

2.5. Integrity Protection of the Wire Image

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

2.6. Connection ID and Rebinding

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

2.7. Packet Numbers

The packet number field is always present in the QUIC packet header; however, it is always encrypted. The encryption key for packet number protection on handshake packets sent before cryptographic context establishment is specific to the QUIC version, while packet number protection on subsequent packets uses secrets derived from the end-to-end cryptographic context. Packet numbers are therefore not part of the wire image that is visible to on-path observers.

2.8. Version Negotiation and Greasing

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

Also note that the list of versions in the Version Negotiation packet may contain reserved versions. This mechanism is used to avoid ossification in the implementation on the selection mechanism.

Further, a client may send a Initial Client packet with a reserved version number to trigger version negotiation. In the Version Negotiation packet the connection ID and packet number of the Client Initial packet are reflected to provide a proof of return-routability. Therefore changing these information will also cause the connection to fail.

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

3. Network-visible information about QUIC flows

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

3.1. Identifying QUIC traffic

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

The only application binding defined by the IETF QUIC WG is HTTP/3 [QUIC-HTTP] at the time of this writing; however, many other applications are currently being defined and deployed over QUIC, so an assumption that all QUIC traffic is HTTP/3 is not valid. HTTP over QUIC uses UDP port 443 by default, although URLs referring to resources available over HTTP over QUIC may specify alternate port numbers. Simple assumptions about whether a given flow is using QUIC based upon a UDP port number may therefore not hold; see also [RFC7605] section 5.

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

3.1.1. Identifying Negotiated Version

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

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

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

3.1.2. Rejection of Garbage Traffic

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

3.2. Connection confirmation

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

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

3.3. Application Identification

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

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

3.4. Flow association

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

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

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

3.5. Flow teardown

QUIC does not expose the end of a connection; the only indication to on-path devices that a flow has ended is that packets are no longer observed. Stateful devices on path such as NATs and firewalls must therefore use idle timeouts to determine when to drop state for QUIC flows, see further section Section 4.1.

3.6. Flow symmetry measurement

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

3.7. Round-Trip Time (RTT) Measurement

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

3.7.1. Measuring initial RTT

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

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

3.7.2. Using the Spin Bit for Passive RTT Measurement

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

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

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

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

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

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

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

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

4. Specific Network Management Tasks

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

4.1. Stateful treatment of QUIC traffic

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

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

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

4.2. Passive network performance measurement and troubleshooting

Limited RTT measurement is possible by passive observation of QUIC traffic; see Section 3.7. No passive measurement of loss is possible with the present wire image. Extremely limited observation of upstream congestion may be possible via the observation of CE markings on ECN-enabled QUIC traffic.

4.3. Server cooperation with load balancers

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

4.4. DDoS Detection and Mitigation

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

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

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

It is questionable whether connection migrations must be supported during a DDoS attack. If the connection migration is not visible to the network that performs the DDoS detection, an active, migrated QUIC connection may be blocked by such a system under attack. As soon as the connection blocking is detected by the client, the client may rely on the fast resumption mechanism provided by QUIC. When clients migrate to a new path, they should be prepared for the migration to fail and attempt to reconnect quickly.

4.5. UDP Policing

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

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

4.6. Distinguishing acknowledgment traffic

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

4.7. QoS support and ECMP

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

QUIC does not provide any additional information on requirements on Quality of Service (QoS) provided from the network. QUIC assumes that all packets with the same 5-tuple {dest addr, source addr, protocol, dest port, source port} will receive similar network treatment. That means all stream that are multiplexed over the same QUIC connection require the same network treatment and are handled by the same congestion controller. If differential network treatment is desired, multiple QUIC connections to the same server might be used, given that establishing a new connection using 0-RTT support is cheap and fast.

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

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

5. IANA Considerations

This document has no actions for IANA.

6. Security Considerations

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

7. Contributors

Dan Druta contributed text to Section 4.4. Igor Lubashev contributed text to Section 4.3 on the use of the connection ID for load balancing. Marcus Ilhar contributed text to Section 3.7 on the use of the spin bit.

8. 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.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

9.2. Informative References

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

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

[I-D.ietf-quic-load-balancers] Duke, M. and N. Banks, "QUIC-LB: Generating Routable QUIC Connection IDs", Work in Progress, Internet-Draft, draft-ietf-quic-load-balancers-02, 9 March 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-load-balancers-02.txt>.

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

[QUIC-APPLICABILITY] Kuehlewind, M. and B. Trammell, "Applicability of the QUIC Transport Protocol", Work in Progress, Internet-Draft, draft-ietf-quic-applicability-06, 6 January 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-applicability-06.txt>.

[QUIC-HTTP]
Bishop, M., "Hypertext Transfer Protocol Version 3 (HTTP/3)", Work in Progress, Internet-Draft, draft-ietf-quic-http-29, 9 June 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-http-29.txt>.

[QUIC-INVARIANTS]
Thomson, M., "Version-Independent Properties of QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-invariants-09, 9 June 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-invariants-09.txt>.

[QUIC-TLS] Thomson, M. and S. Turner, "Using TLS to Secure QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-tls-29, 9 June 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-tls-29.txt>.

[QUIC-TRANSPORT]
Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", Work in Progress, Internet-Draft, draft-ietf-quic-transport-29, 9 June 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-29.txt>.

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

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

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

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

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <https://www.rfc-editor.org/info/rfc7301>.

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

[TLS-ESNI] Rescorla, E., Oku, K., Sullivan, N., and C. Wood, "TLS Encrypted Client Hello", Work in Progress, Internet-Draft, draft-ietf-tls-esni-07, 1 June 2020, <http://www.ietf.org/internet-drafts/draft-ietf-tls-esni-07.txt>.

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

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

Authors' Addresses

Mirja Kuehlewind
Ericsson

Email: mirja.kuehlewind@ericsson.com

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

Email: ietf@trammell.ch

QUIC
Internet-Draft
Intended status: Experimental
Expires: October 22, 2020

T. Li
K. Zheng
R. Jadhav
J. Kang
Huawei Technologies

April 20, 2020

Optimizing ACK mechanism for QUIC
draft-li-quic-optimizing-ack-in-wlan-00

Abstract

This document analyzes the problems caused by contentions and collisions on wireless medium between data packets and ACKs in WLAN and it proposes an optimized ACK mechanism that can minimize the intensity of ACK Frame in QUIC, improving the performance of transport layer connection.

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 October 22, 2020.

Copyright Notice

Table of Contents

1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Problem Statement

High-throughput transport over wireless local area network (WLAN) becomes a demanding requirement with the emergence of 4K wireless projection, VR/AR-based interactive gaming, and more. However, the shared nature of the wireless medium induces contention between data transport and backward signaling, such as acknowledgement. ACKs share the same medium route with data packets, causing similar medium access overhead despite the much smaller size of the ACKs. Contentions and collisions, as well as the wasted wireless resources by ACKs, lead to significant throughput decline on the data path.

3. ACK Mechanism in Current QUIC

[QUIC-TRANSPORT] specifies a simple delayed ACK mechanism that a receiver can send an ACK for every other packet, and for every packet when reordering is observed, or when the max_ack_delay timer expires. However, this ACK mechanism may not match the number of ACKs to the

transport's required intensity under different network conditions. For example, when the data throughput of a WLAN transport is extremely high, QUIC will generate a large number of ACKs. In this case, minimizing the ACK intensity of QUIC is not only a win for data throughput improvement but also a win for energy and CPU efficiency.

4. Optimized ACK Mechanism for QUIC

4.1. Reducing ACK intensity

ACK intensity can be quantified by the unit of Hz, i.e., number of ACKs per second. Byte-counting ACK and periodic ACK are two fundamental ways to reduce ACK intensity on the transport layer.

1. Byte-counting ACK: ACK intensity is controlled by sending an ACK for every L (L >= 2) incoming full-sized packets, in which the packet size equals to the Max Packet Size (set in the max_packet_size parameter in QUIC). The intensity of byte-counting ACK (f_b) is proportional to data throughput (bw):

f_b = bw/L*max_packet_size (1)

In general, f_b can be reduced by setting a large value of L. However, for a given L, f_b increases with bw. This means when data throughput is extremely high, the ACK intensity still might be comparatively large. In other words, the intensity of byte-counting ACK changes proportionately with bandwidth.

2. Periodic ACK: Byte-counting ACK's unbounded intensity can be attributed to the coupling between ACK sending and packet arrivals. Periodic ACK can decouple ACK intensity from packet arrivals, achieving a bounded ACK intensity when bw is high. The intensity of periodic ACK (f_pack) is:

f_pack = 1/alpha (2)

Where alpha is the time interval between two ACKs and is a function of RTT. However, when bw is extremely low, the ACK intensity is always as high as that in the case of a high throughput. In other words, the intensity of periodic ACK is unadaptable to bandwidth change, which wastes resources.

Combining these two ways, the minimum ACK intensity in a QUIC connection can be set as f_quic = min{f_b,f_pack}. Through Equations (1) and (2), we have

f_quic = min{bw/(L*max_packet_size), 1/alpha} (3)

We set alpha = RTTmin/beta, which means sending beta ACKs per RTTmin. RTTmin is the minimum RTT observed for a given network path. As a consequence, the minimum ACK intensity in a QUIC connection can be given as follow:

f_quic = min{bw/(L*max_packet_size), beta/RTTmin} (4)

where beta indicates the number of ACKs per RTT, and L indicates the number of full-sized data packets counted before sending an ACK. To minimize the ACK intensity, a smaller beta or a larger L is expected. Sara Landstrom et al. has given a lower bound of beta in [Sara], i.e., beta >= 2. An upper bound of L can also be derived according to the loss rate on the data path (plr_data) and the ack path (plr_ack), i.e., L <= feedback_info/(plr_data*plr_ack), where feedback_info denotes the amount of information carried by an ACK

Qualitatively, periodic ACK is applied when bandwidth-delay product (bdp) is large (i.e., bdp >= beta*L* max_packet_size), and byte-counting ACK is applied when bdp is small (i.e., bdp < beta*L* max_packet_size).

In terms of a transport with a large bdp, beta = 2 should be sufficient to ensure utilization, but the large bottleneck buffer (i.e., one bdp) makes it necessary to acknowledge data more often. In general, the minimum send window (SWNDmin) can be roughly estimated as follow:

SWNDmin = beta*bdp/(beta-1) (5)

Ideally, the bottleneck buffer requirement is decided by the minimum send window, i.e., SWNDmin - bdp. Since doubling the ACK frequency reduces the bottleneck buffer requirement substantially from 1 bdp to 0.33 bdp, beta = 4 is RECOMMENDED to provide redundancy [Sara], being more robust in practice.

4.2. OWD-based RTTmin estimation

In this document, the RTTmin is the minimum RTT samples observed at the sender for a given network path during a period of time, and OWDmin is the minimum OWD samples observed on the same network path during a period of time.

When multiple packets carrying departure timestamps are transported between endpoints via the same path, an RTT of this path can be sampled at the sender upon receiving an ACK frame. However, when sending fewer ACK frames, more data packets might be received during the ACK interval, generating only one RTT sample among multiple packets is likely to result in biases. For example, a larger minimum

RTT estimate. In general, the higher the throughput, the larger the biases. One alternative way to reduce biases can be that, each ACK frame carries multiple timestamps (as well as ACK delays in [QUIC-RECOVERY] for the sender to generate more RTT samples. However, (1) the overhead is high, which is unacceptable especially under high-bandwidth transport. Also, (2) the number of data packets might be far more than the maximum number of timestamps that an ACK frame is capable to carry.

An RTT estimation system contains a sender and a receiver. The sender can hardly generate per-packet RTT samples, which is the root cause of the minimum RTT estimation biases in the case of sending fewer ACKs. When multiple packets carrying departure timestamps are transported between endpoints via the same path , an RTT of this path can be sampled at the sender upon receiving an ACK frame. However, when sending fewer ACK frames, more data packets might be received during the ACK interval, generating only one RTT sample among multiple packets is likely to result in biases. For example, a larger minimum RTT estimate. In general, the higher the throughput, the larger the biases. One alternative way to reduce biases can be that, each ACK frame carries multiple timestamps (as well as ACK delays in [QUIC-RECOVERY]) for the sender to generate more RTT samples. However, (1) the overhead is high, which is unacceptable especially under high-bandwidth transport. Also, (2) the number of data packets might be far more than the maximum number of timestamps that an ACK frame is capable to carry. Since the receiver is capable to monitor per-packet state, the one-way delay (OWD) of each packet can be easily computed according to the departure timestamps (carried in the packet) and the arrival timestamps of each packet. In this case, QUIC SHOULD adopt the OWD-based RTTmin estimation. The rationale is that the variation of OWD reflects the variation of RTT over near-symmetric links. The OWD-based RTTmin estimation requires the sender to record the departure timestamp in each ack-eliciting packet. Meanwhile, at the receiver, the per-packet OWD samples SHOULD be computed upon packet arrivals and a function of computing the minimum OWD SHOULD be newly added. The receiver then generates an ACK frame to the sender, in which the ACK delay and departure timestamp for the packet that achieves the minimum OWD is reported. The ACK delay is defined as the delay incurred between when the packet is received and when the ACK frame is sent. Based on the information reported by the incoming ACK frames and the ACK arrival timestamps, the sender can generate RTT samples and then compute RTTmin accordingly.

In this document, RTTmin is used to update the ACK intensity. In general, RTTmin can also be used by other modules. For example, some congestion controllers depends on RTTmin to estimate the congestion

window [Neal]. RTTmin is also used by QUIC loss detection to reject implausibly small rtt samples [QUIC-RECOVERY].

4.3. Sender-Side Operation

According to Formula (4), the run-time ACK intensity in QUIC are decided by bw, and RTTmin. Generally, the RTTmin and bw are calculated at the sender.

Before estimating the RTTmin, the RTT samples should be computed based on the ACK frames collected during a period of time. Assume that a packet is sent by the sender at time t_1 and arrives at time t_3, and the ACK frame is sent at time t_4. The ACK delay can be computed at the receiver. For example, the receiver computes the ACK delay delta_t = t_4 - t_3, and syncs the ACK delay to the sender via an ACK frame. The ACK delay can also be computed at the sender. For example, the receiver directly syncs an ACK frame carrying t_4 and t_3 to the sender, the sender then computes the ACK delay delta_t = t_4 - t_3.

The sender therefore computes an RTT sample according to delta_t, t_1, and the arrival time (t_2) of the ACK frame, i.e., RTT_sample = t_2 - t_1 - delta_t. Measuring delta_t at the receiver assures an explicit correction for a more accurate RTT estimate. RTT samples SHOULD be smoothed using exponentially weighted moving average (EWMA) as specified in [RFC6298]. The sender then computes the RTTmin according to these RTT samples during a period of time.

The bw estimation can be acquired in a similar manner to BBR [Neal]. Since minimizing the ACK intensity induces excessive ACK delay, the value of bw may be the average value over a long period of time. However, the biases introduced in ACK intensity computation is limited.

After computing the f_quic, the sender periodically syncs it to the receiver to update the intensity of ACK Frame by sending a new ACK-INTENSITY frame.

The sender SHOULD generate an ACK-INTENSITY frame on a regular basis. For example, when the change of f_quic exceeds a threshold, the ACK-INTENSITY frame should be sent to update the ACK intensity in time. The interval of ACK-INTENSITY frame can also be set according to the update window of RTTmin and bw.

4.4. Receiver-side Operation

Currently, the QUIC receiver reports ACK delays for only the largest acknowledged packet in an ACK frame, hence an RTT sample is generated using only the largest acknowledged packet in the received ACK frame. For a more accurate RTTmin estimate when sending fewer ACK frames, QUIC SHOULD adopt the OWD-based RTTmin estimation. The OWD-based RTTmin estimation requires the QUIC receiver to filter the departure timestamp for the packet that achieves the minimum OWD during the interval between two ACK frames and report the ACK delay of this packet. Whether redefining the meaning of ACK delay or not, it depends on the negotiation between endpoints of the QUIC connection.

Upon packet arrivals, the receiver is capable to generate per-packet OWD samples according to the difference between packet departure timestamp and packet arrival timestamp. The receiver then computes the minimum OWD by comparing the per-packet OWD samples. The OWD estimation does not require clock synchronization here because the relative values are adopted.

Afterwards, based on the ACK delay and the departure timestamp corresponding to the packet that achieves the minimum OWD, the sender calculates the RTT of this packet as a minimum RTT sample. Ultimately, the minimum RTT is computed according to these minimum RTT samples.

The ACK Delay field SHOULD be carried in the ACK Frame. Other fields carried in the ACK frame have the same meaning as defined in [QUIC-RECOVERY].

The receiver adopts the newly updated ACK intensity once it receives the ACK-INTENSITY frame from the sender.

4.5. Generating ACK

The newly proposed ACK mechanismSHOULD be applied when there is no out-of-order delivery. When reordering happens, the ACK Frame SHOULD be generated immediately.

4.6. Modification to QUIC Protocol

4.6.1. Transport Parameter: ack-intensity-support

A new field named ack-intensity-support should be added for negotiation between both parties whether starting the dynamic ACK intensity function in QUIC connection. The endpoints sends this parameter during handshakes. Only when both parties agree, ACK intensity refreshment can be adopted.

ack-intensity-support (0x XX):This parameter has two values (0 or 1) specifying whether the sending endpoint is willing to adopt ACK intensity refreshment. When the value is set as 1, it means that the sending endpoint want to start ACK intensity refreshment during connection. When the value is set as 0, it means that the sending endpoint does not support this function.

4.6.2. ACK-INTENSITY Frame

An ACK-INTENSITY frame is shown in Figure 1.

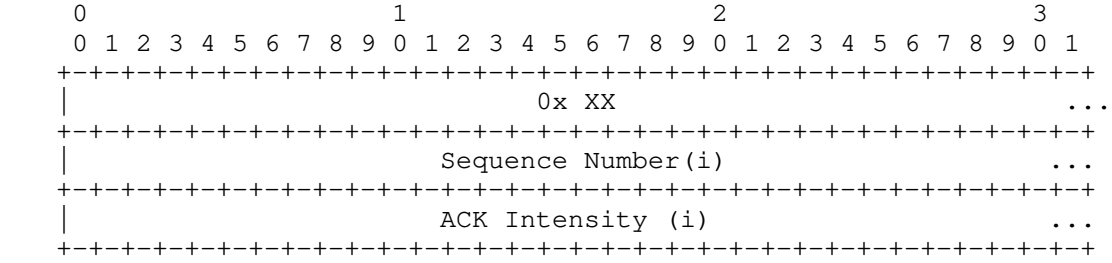


Figure 1: ACK-INTENSITY Frame

An ACK-INTENSITY frame contains the following fields:

Sequence Number: A variable-length integer indicating the sequence number assigned to the ACK-INTENSITY frame by the sender.

ACK Intensity: A variable-length integer indicating the updated f_quic calculated by the sender.

ACK-INTENSITY frames are ack-eliciting. However, their loss does not require retransmission.

Multiple ACK-INTENSITY frames SHOULD be generated by the sender during a connection to notify the receiver the variation of ACK intensity requirement under network dynamics.

5. Security Considerations

TBD

6. IANA Considerations

The value for ack-intensity-support transport parameter and ACK-INTENSITY frame should be allocated.

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <https://www.rfc-editor.org/info/rfc6298>.

7.2. Informative References

[Neal] Cardwell, N., Cheng, Y., Gunn, C. S., Yeganeh, S. H., and V. Jacobson, "BBR: Congestion-based congestion control", ACM QUEUE 14(5):20-53, 2016.

[QUIC-RECOVERY] Iyengar, J., Ed. and I. Swett, Ed., "QUIC Loss Detection and Congestion Control", draft-ietf-quic-recovery-27 (work in progress), February 2020.

[QUIC-TRANSPORT] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-27 (work in progress), March 2020.

[Sara] Landstrom, S. and L. Larzon, "Reducing the tcp acknowledgment frequency", ACM SIGCOMM CCR 37(3):5-16, 2007.

Authors' Addresses

Tong Li
Huawei Technologies
D2-03,Huawei Industrial Base
Longgang District
Shenzhen
China

Email: li.tong@huawei.com

Kai Zheng
Huawei Technologies
Information Road, Haidian District
Beijing
China

Email: kai.zheng@huawei.com

Rahul Arvind Jadhav
Huawei Technologies
D2-03,Huawei Industrial Base
Longgang District
Shenzhen
China

Email: rahul.jadhav@huawei.com

Jiao Kang
Huawei Technologies
D2-03,Huawei Industrial Base
Longgang District
Shenzhen
China

Email: kangjiao@huawei.com

quic
Internet-Draft
Intended status: Standards Track
Expires: 28 December 2020

M. Thomson
Mozilla

26 June 2020

Greasing the QUIC Bit
draft-thomson-quic-bit-grease-00

Abstract

This document describes a method for negotiating the ability to send an arbitrary value for the second-to-most significant bit in QUIC packets.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the QUIC Working Group mailing list (quic@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/quic/ (https://mailarchive.ietf.org/arch/browse/quic/).

Source for this draft and an issue tracker can be found at https://github.com/martinthomson/quic-bit-grease (https://github.com/martinthomson/quic-bit-grease).

Table of Contents

1. Introduction

QUIC [QUIC] intentionally describes a very narrow set of fields that are visible to entities other than endpoints. Beyond those characteristics that are defined as invariant [QUIC-INVARIANTS], very little about the "wire image" [RFC8546] of QUIC is visible.

The second-to-most significant bit of the first byte in every QUIC packet is defined as having a fixed value in QUIC version 1 [QUIC]. The purpose of having a fixed value is to allow intermediaries and endpoints to efficiently distinguish between QUIC and other protocols; see [DEMUX] for a description of a scheme that QUIC can integrate with as a result. As this bit effectively identifies a packet as QUIC, it is sometimes referred to as the "QUIC Bit".

Where endpoints and the intermediaries that support them do not depend on the QUIC Bit having a fixed value, sending the same value in every packet is more of liability than an asset. If systems come to depend on a fixed value, then it might become infeasible to define a version of QUIC that attributes semantics to this bit.

In order to safeguard future use of this bit, this document defines a QUIC transport parameter that indicates that an endpoint is willing to receive QUIC packets containing any value for this bit. By sending different values for this bit, the hope is that the value will remain available for future use [USE-IT].

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document uses terms and notational conventions from [QUIC].

3. The Grease QUIC Bit Transport Parameter

The grease_quic_bit transport parameter (0xTBD) can be sent by both client and server. The transport parameter is sent with an empty value; an endpoint that understands this transport parameter MUST treat receipt of a non-empty value as a connection error of type TRANSPORT_PARAMETER_ERROR.

Advertising the grease_quic_bit transport parameter indicates that packets sent to this endpoint MAY set a value of 0 for the QUIC Bit. The QUIC Bit is defined as the second-to-most significant bit of the first byte of QUIC packets (that is, the value 0x40).

3.1. Clearing the QUIC Bit

Endpoints that receive the grease_quic_bit transport parameter from a peer MAY set the QUIC Bit to any value in packets they sent to that peer. Endpoints SHOULD set the QUIC Bit to an unpredictable value unless another extension assigns specific meaning to the value of the bit. All packets sent after receiving and processing transport parameters are affected, including Retry, Initial, and Handshake packets.

A client MAY also clear the QUIC Bit in Initial packets that are sent to establish a new connection. A client can only clear the QUIC Bit if the packet includes a token provided by the server in a NEW_TOKEN

frame on a connection where the server also included the grease_quic_bit transport parameter. To allow for changes in server configuration, clients SHOULD set the QUIC Bit if the token was provided more than 7 days prior.

3.2. Using the QUIC Bit

The purpose of this extension is to allow for the use of the QUIC Bit by later extensions.

Extensions to QUIC that define semantics for the QUIC Bit can be negotiated at the same time as the grease_quic_bit transport parameter. In this case, a recipient needs to be able to distinguish a randomized value from a value carrying information according to the extension. Extensions that use the QUIC Bit MUST negotiate their use prior to acting on any semantic. Endpoints MAY send a signal prior to this negotiation completing, but any value carried by the bit cannot be used until it is clear that the peer is using the extension.

For example, an extension might define a transport parameter that is sent in addition to the grease_quic_bit transport parameter. Though the value of the QUIC Bit in packets received by a peer might be set according to rules defined by the extension, they might also be randomized according to the definition of the grease_quic_bit extension. Receiving the transport parameter for the extension could be used to confirm that a peer supports the semantic defined in the extension. To avoid acting on a randomized signal, the extension can require that endpoints set the QUIC Bit according to the rules of the extension, but defer acting on the information conveyed until the transport parameter for the extension is received.

Extensions that define semantics for the QUIC Bit can be negotiated without using the grease_quic_bit transport parameter.

4. Security Considerations

This document introduces no new security considerations for endpoints or entities that can rely on endpoint cooperation. However, this change makes the task of identifying QUIC more difficult without cooperation of endpoints. This sometimes works counter to the security goals of network operators who rely on network classification to identify threats.

5. IANA Considerations

This document registers the grease_quic_bit transport parameter in the "QUIC Transport Parameters" registry established in Section 22.2 of [QUIC]. The following fields are registered:

Value: 0x2ab2

Parameter Name: grease_quic_bit

Status: Permanent

Specification: This document.

Date: Date of registration.

Contact: IETF QUIC Working Group (quic@ietf.org)

Notes: (none)

6. References

6.1. 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-29, 9 June 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-29.txt>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

6.2. Informative References

[DEMUX] 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>.

[QUIC-INVARIANTS]
Thomson, M., "Version-Independent Properties of QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-invariants-09, 9 June 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-invariants-09.txt>.

[RFC8546] 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>.

[USE-IT] Thomson, M., "Long-term Viability of Protocol Extension Mechanisms", Work in Progress, Internet-Draft, draft-iab-use-it-or-lose-it-00, 7 August 2019, <http://www.ietf.org/internet-drafts/draft-iab-use-it-or-lose-it-00.txt>.

Acknowledgments

TODO

Author's Address

Martin Thomson
Mozilla

Email: mt@lowentropy.net