MASQUE
Internet-Draft
Intended status: Experimental
Expires: 3 September 2026

T. Pauly
E. Rosenberg
Apple Inc.
D. Schinazi
Google LLC

2 March 2026

QUIC-Aware Proxying Using HTTP
draft-ietf-masque-quic-proxy-08

Abstract

This document extends UDP Proxying over HTTP to add optimizations for proxied QUIC connections. Specifically, it allows a proxy to reuse UDP 4-tuples for multiple proxied connections, and adds a mode of proxying in which QUIC short header packets can be forwarded and transformed through a HTTP/3 proxy rather than being fully re-encapsulated and re-encrypted.

About This Document

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

The latest revision of this draft can be found at https://ietf-wg-masque.github.io/draft-ietf-masque-quic-proxy/draft-ietf-masque-quic-proxy.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-masque-quic-proxy/.

Discussion of this document takes place on the MASQUE Working Group mailing list (mailto:masque@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/masque/. Subscribe at https://www.ietf.org/mailman/listinfo/masque/.

Source for this draft and an issue tracker can be found at https://github.com/ietf-wg-masque/draft-ietf-masque-quic-proxy.

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/.

Copyright Notice

Table of Contents

1. Introduction

UDP Proxying over HTTP [CONNECT-UDP] defines a way to send datagrams through an HTTP proxy, where UDP is used to communicate between the proxy and a target server. This can be used to proxy QUIC connections [QUIC], since QUIC runs over UDP datagrams.

This document uses the term "target" to refer to the server that a client is accessing via a proxy. This target may be an origin server hosting content, or another proxy for cases where proxies are chained together.

This document extends the UDP proxying protocol to add signalling about QUIC Connection IDs. QUIC Connection IDs are used to identify QUIC connections in scenarios where there is not a strict one-to-one mapping between QUIC connections and UDP 4-tuples (pairs of IP addresses and ports).

If a client permits proxy port reuse, once a proxy is aware of QUIC Connection IDs, it can reuse UDP 4-tuples between itself and a target for multiple proxied QUIC connections.

For proxies that are themselves running on HTTP/3 [HTTP3], and thus are accessed by clients over QUIC, QUIC Connection IDs can be used to treat packets differently on the link between clients and proxies. New QUIC Connection IDs can be assigned to perform transformations to the packets that allow for efficient forwarding of packets that don't require full re-encapsulation and re-encryption of proxied QUIC packets within datagrams inside the QUIC connection between clients and proxies.

This document defines two modes for proxying QUIC connections, "tunnelled" and "forwarded":

1. Tunnelled is the default mode for UDP proxying, defined in [CONNECT-UDP]. In this mode, packets in QUIC connection between the client and target are encapsulated inside the QUIC connection between the client and proxy. These packets use multiple layers of encryption and congestion control.

2. Forwarded is the mode of proxying added by this document. In this mode, packets in the QUIC connection between the client and target are sent with dedicated QUIC Connection IDs between the client and proxy, and use special-purpose tranforms instead of full re-encapsulation and re-encryption.

QUIC long header packets between clients and targets MUST be proxied in tunnelled mode. QUIC short header packets between clients and targets MAY be proxied in forwarded mode, subject to negotiation between a client and a proxy.

Forwarded mode is an optimization to reduce CPU and memory cost to clients and proxies and avoid encapsulation overhead for packets on the wire that reduce the effective MTU (Maximum Transmission Unit). This makes it suitable for deployment situations that otherwise relied on cleartext TCP proxies, which cannot support QUIC and have inferior security and privacy properties.

The properties provided by the forwarded mode are as follows:

* All packets sent between the client and the target traverse through the proxy device.

* The target server cannot know the IP address of the client solely based on the proxied packets the target receives.

* Observers of either or both of the links between client and proxy and between proxy and target are not able to learn more about the client-to-target communication than if no proxy was used.

Forwarded mode does not prevent correlation of packets on the link between client and proxy and the link between proxy and target by an entity that can observe both links. The precise risks depend on the negotiated transform (Section 6.3). See Section 10 for further discussion.

Both clients and proxies can unilaterally choose to disable forwarded mode for any client-to-target connection.

The forwarded mode of proxying is only defined for HTTP/3 [HTTP3] and not any earlier versions of HTTP.

QUIC proxies can proxy all versions of QUIC. See Section 9 for more information.

While Forwarded mode may improve overhead of per-packet processing, this doesn’t necessarily imply overall throughput is improved. Unlike tunnelled packets, packets sent in Forwarded mode are not congestion controlled between client and proxy. Deployments should consider whether or not the overhead advantages outweigh potentially superior throughput afforded by client-to-proxy congestion control.

1.1. 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.

1.2. Terminology

This document uses the following terms:

* Client: the client of all QUIC connections discussed in this document.

* Proxy: the endpoint that responds to the UDP proxying request.

* Target: the server that a client is accessing via a proxy.

* Client-to-proxy 4-tuple: the UDP 4-tuple (client IP address, client UDP port, proxy IP address, proxy UDP port) used to communicate between the client and the proxy.

* Proxy-to-target 4-tuple: the UDP 4-tuple (proxy IP address, proxy UDP port, target IP address, target UDP port) used to communicate between the proxy and the target.

* Client Connection ID (CID): a QUIC Connection ID that is chosen by the client, and is used in the Destination Connection ID field of packets from the target to the client.

* Target Connection ID (CID): a QUIC Connection ID that is chosen by the target, and is used in the Destination Connection ID field of packets from the client to the target.

* Virtual Connection ID (VCID): a fake QUIC Connection ID chosen by the proxy that is used on the client-to-proxy 4-tuple in forwarded mode.

* Client VCID: a VCID used by the proxy to send forwarded packets from the target to the client.

* Target VCID: a VCID used by the client to send forwarded packets to the target via the proxy.

* Packet Transform: the procedure used to modify packets before they enter the client-proxy link.

2. Protocol Overview

QUIC-aware proxying involves a client, a proxy, and a target. Although multiple proxies can be chained together in sequence (which is a main motivation for enabling forwarded mode), each subsequent proxy is just treated as a target by the preceding proxy.

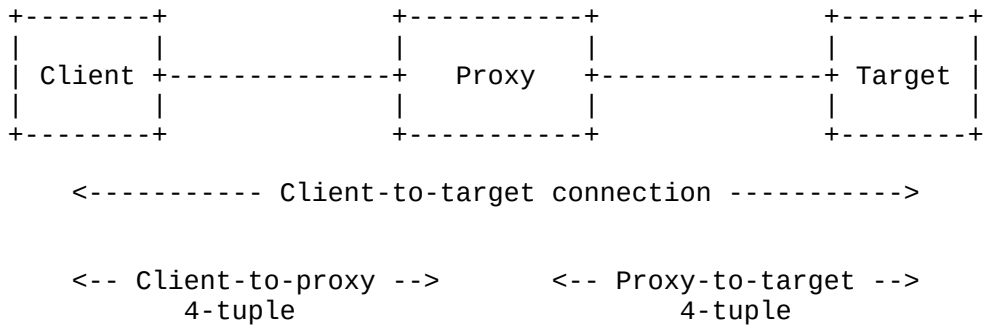

Figure 1: Diagram of roles in QUIC-aware proxying

All QUIC-aware proxying relies on the proxy learning information about QUIC connection IDs on the client-to-target QUIC connection (Section 2.1).

Forwarded mode adds the concept of Virtual Connection IDs that are assigned by the proxy to use to identify packets on the client-to-proxy 4-tuple (Section 2.2).

Negotation of modes and assignment of connection IDs is described in Section 2.3.

2.1. Connection ID Awareness

For a proxy to be aware of proxied QUIC connection IDs, it needs to know and correlate three values:

1. The HTTP stream used to proxy a client-to-target QUIC connection

2. The client-chosen connection ID on the client-to-target QUIC connection, or the "client CID"

3. The target-chosen connection ID on the client-to-target QUIC connection, or the "target CID"

For example, consider a proxy using HTTP/3 that has two clients (A and B) connected simultaneously, each client coming from a different IP address and port. Each client makes a request to proxy a UDP flow to "target.example.net" on port 443. If the proxy knows that client A’s connection to the target has negotiated a client CID AAAA0000 and a target CID 0000AAAA, and client B’s connection to the target has negotiated a client CID BBBB0000 and a target CID 0000BBBB, then the proxy would be able to use the same proxy-to-target 4-tuple for both connections, because it can route packets from the target to the correct client based on CID AAAA0000 or BBBB0000.

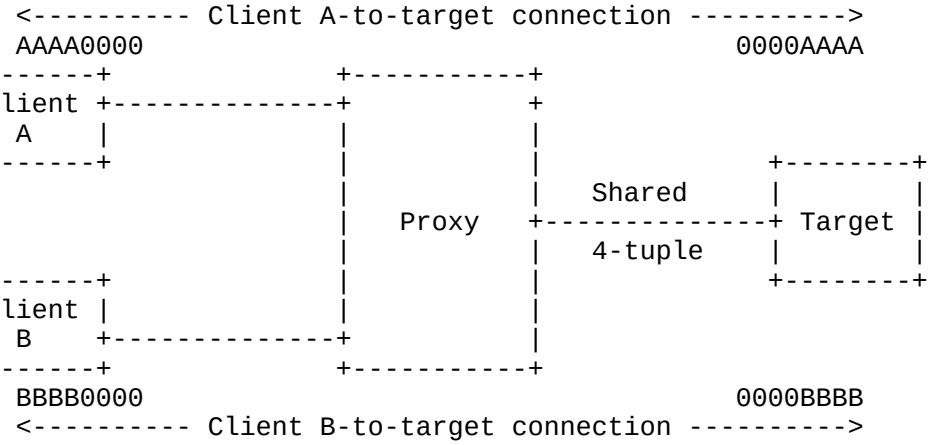

Figure 2: Example of sharing a proxy-to-target 4-tuple

In order to share a proxy-to-target 4-tuple between multiple proxied connections, the proxy MUST guarantee that the client CIDs do not conflict. See Section 5.8 for more discussion of handlng CID conflicts.

2.2. Virtual Connection IDs

Virtual Connection IDs (VCIDs) are QUIC Connection IDs used on the link between a client and proxy that do not belong to the QUIC connection between the client and proxy, but instead are aliases for particular client-to-target connections. VCIDs are only used in forwarded mode. They are established using HTTP capsules [HTTP-CAPSULES] as described in Section 5.

For example, consider a proxy using HTTP/3 that has a single client connected to it. The client-to-proxy QUIC connection has CCCC0000 as the client CID and 0000CCCC as the proxy CID. The client has connected to a single target through the proxy, and the client-to-target QUIC connection has CCCC1111 as the client CID and 1111CCCC as the target CID. In order to use forwarded mode, the proxy assigns VCIDs to use on the client-to-proxy link to represent the client-to-target connection. In this case, the VCIDs could be CCCC2222 for the client’s VCID and 2222CCCC for the proxy’s VCID that it uses to forward to the target.

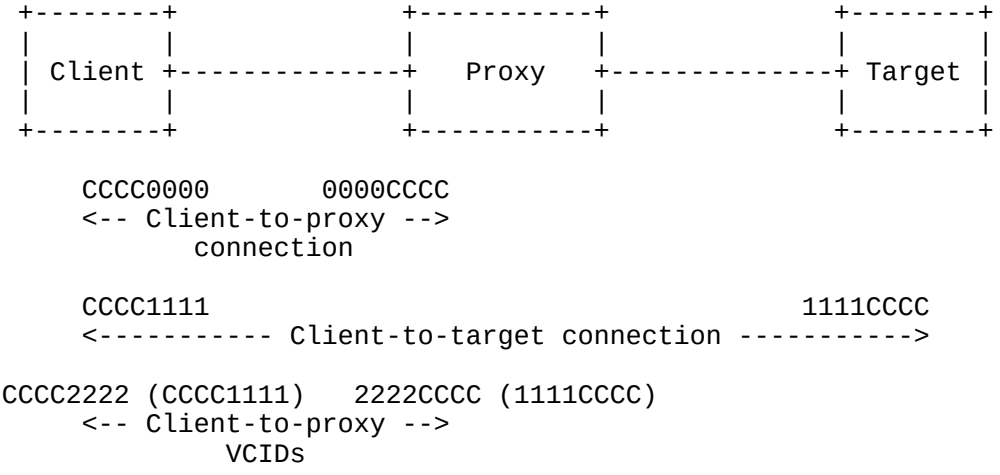

Figure 3: Diagram of VCIDs in forwarded mode

In order for a proxy to correctly route packets using VCIDs from client-to-target and target-to-client, the proxy MUST guarantee that the mappings between VCIDs, CIDs, and 4-tuples are unique. Specifically, in order to route packets sent by the client, the proxy needs to be able to observe the VCID and the client-to-proxy 4-tuple, and map them to a specific target CID and proxy-to-target 4-tuple. In order to route packets sent by a target, the proxy needs to be able to observe the client CID and the proxy-to-target 4-tuple, and map them to a specific VCID and client-to-proxy 4-tuple. Since proxies choose the VCID values, they can ensure that the VCIDs are distinguishable.

Servers receiving QUIC packets can employ load balancing strategies such as those described in [QUIC-LB] that encode routing information in the connection ID. When operating in forwarded mode, clients send QUIC packets destined for the target directly to the proxy. Since these packets are generated using the target CID, load balancers may not have the necessary information to route packets to the correct proxy. The target VCID is a VCID chosen by the proxy that the client uses when sending forwarded mode packets. The proxy replaces the target VCID with the target CID prior to forwarding the packet to the target.

Similarly, QUIC requires that connection IDs aren't reused over multiple network paths to avoid linkability. The client VCID is a connection ID chosen by the proxy that the proxy uses when sending forwarded mode packets. The proxy replaces the client CID with the client VCID prior to forwarding the packet to the client. Clients take advantage of this to avoid linkability when migrating a client

to proxy network path. The Virtual client CID allows the connection ID bytes to change on the wire without requiring the connection IDs on the client to target connection change. To reduce the likelihood of connection ID conflicts, the proxy MUST choose a client VCID that is at least as long as the original client CID. Similarly, clients multiplexing connections on the same UDP 4-tuple SHOULD choose a client CID that's sufficiently long to reduce the likelihood of a conflict with the proxy-chosen client VCID. For proxies that share a client-facing IP and port with other proxies, support target-facing port sharing, and cannot guarantee there are no proxies being proxied to, the client VCID MUST be constructed such that it is unpredictable to the client or to guarantee no conflicts among all proxies sharing an IP address and port. For such proxies, a non-empty client VCID MUST NOT be equal to the original client CID. See Section 10 for more discussion on client VCID construction.

Clients and Proxies not implementing forwarded mode do not need to consider VCIDs since all client-to-target datagrams will be encapsulated within the client-to-proxy connection.

2.3. Negotiating Modes and Connection IDs

In order to support QUIC-aware proxying, both clients and proxies need to support capsules [HTTP-CAPSULES], which is indicated by including the "Capsule-Protocol" header field in requests and responses. If this header field is not included, none of the functionality in this document can be used.

```
capsule-protocol = ?1
```

To permit the proxy to share target-facing ports, the client needs to include the "Proxy-QUIC-Port-Sharing" header field, as defined in Section 4. This indicates that the proxy may share target-facing 4-tuples concurrently with other QUIC connections. Clients that do not want the target-facing 4-tuple shared will not include this header or will provide it with a value of "?0".

```
proxy-quic-port-sharing = ?1
```

To support forwarded mode, both clients and proxies need to include the "Proxy-QUIC-Forwarding" header field, as defined in Section 3. This indicates support for forwarded mode, and allows negotiation of packet transforms to apply when forwarding (Section 6.3), along with any parameters for specific transforms. Clients or proxies that don't support forwarded mode will not include this header field.

```
proxy-quic-forwarding = ?1; accept-transform="scramble,identity"; \
    scramble-key=:abc...789=:
```

If neither header is supplied with a value of "?1", none of the functionality in this document can be used and handling reduces to normal connect-udp.

After negotiating support with header fields, clients and proxies use the capsules defined in Section 5 to communicate information about CIDs and VCIDs.

For QUIC-aware proxying without forwarded mode, the steps are as follows:

1. The client sends the REGISTER_CLIENT_CID capsule once it selects a CID that it will use for receiving packets from a target.

2. The proxy sends the ACK_CLIENT_CID capsule to acknowledge that CID, with no associated client VCID; alternatively, the proxy can send the CLOSE_CLIENT_CID if it detects a conflict with another CID.

3. The proxy sends the MAX_CONNECTION_IDS capsule to allow additional registration of new connection IDs via future REGISTER_CLIENT_CID and REGISTER_TARGET_CID capsules.

4. Whenever a client stops using a particular CID, the client sends a CLOSE_CLIENT_CID. The client can also initiate new REGISTER_CLIENT_CID exchanges at any time.

For QUIC-aware proxying with forwarded mode, the steps are as follows:

1. The client sends the REGISTER_CLIENT_CID capsule once it selects a CID that it will use for receiving packets from a target.

2. The proxy sends the ACK_CLIENT_CID capsule to acknowledge that CID, with a client VCID; alternatively, the proxy can send the CLOSE_CLIENT_CID if it detects a conflict with another CID.

3. The client sends the ACK_CLIENT_VCID capsule to acknowledge the client VCID, which allows forwarded packets for that VCID to be used.

4. The client sends the REGISTER_TARGET_CID capsule as soon as it learns the target CID on the client-to-target connection.

5. The proxy sends the ACK_TARGET_CID capsule to acknowledge that CID, with a target VCID; alternatively, the proxy can send the CLOSE_TARGET_CID if it detects a conflict with another CID. Once the client receives the target VCID, it can start sending forwarded packets using the target VCID.

6. The proxy sends the MAX_CONNECTION_IDS capsule to allow additional registration of new connection IDs via future REGISTER_CLIENT_CID and REGISTER_TARGET_CID capsules.

7. Whenever a client or target stops uses a particular CID, the client sends a CLOSE_CLIENT_CID or CLOSE_TARGET_CID capsule. The client can also initiate new REGISTER_CLIENT_CID or REGISTER_TARGET_CID exchanges at any time.

3. Proxy-QUIC-Forwarding Header

A client initiates UDP proxying via a CONNECT request as defined in [CONNECT-UDP]. Within its request, it includes the "Proxy-QUIC-Forwarding" header to indicate whether or not the request should support forwarding. If this header is not included, the client MUST NOT send any connection ID capsules.

"Proxy-QUIC-Forwarding" is an Item Structured Header [RFC8941]. Its value MUST be a Boolean.

If the client wants to enable QUIC packet forwarding for this request, it sets the value to "?1". If it doesn’t want to enable forwarding, but instead only provide information about QUIC Connection IDs for the purpose of allowing the proxy to share a proxy-to-target 4-tuple, it sets the value to "?0".

If advertising support for forwarding, the client MUST add an "accept-transform" parameter whose value is a String containing the supported packet transforms (Section 6.3) in order of descending preference, separated by commas. If the proxy receives a "Proxy-QUIC-Forwarding" header with a value of "?1" and without the "accept-transform" parameter, it MUST ignore the header and respond as if the client had not sent the "Proxy-QUIC-Forwarding" header.

If the proxy supports QUIC-aware proxying, it will include the "Proxy-QUIC-Forwarding" header in successful HTTP responses. The value indicates whether or not the proxy supports forwarding. If the client does not receive this header in responses, the client SHALL assume that the proxy does not support this extension. If the client receives a transform that it did not advertise support for, it MUST abort the request.

The proxy MUST include a "transform" parameter whose value is a String indicating the selected transform. If the proxy does not recognize or accept any of the transforms offered by the client, it MUST omit this parameter and set the header field value to "?0", or omit the header entirely.

Clients MUST NOT send the "transform" parameter and servers MUST NOT send the "accept-transform" parameter. Clients MUST ignore receipt of an "accept-transform" parameter and servers MUST ignore receipt of a "transform" parameter.

4. Proxy-QUIC-Port-Sharing Header

A client may include the "Proxy-QUIC-Port-Sharing" header to indicate whether or not the proxy is permitted to share ports between this QUIC connection and other proxied QUIC connections.

"Proxy-QUIC-Port-Sharing" is an Item Structured Header [RFC8941]. Its value MUST be a Boolean.

Clients SHOULD send this with a value of "?1" unless the client wishes to prohibit this behavior. Permitting the proxy to share ports may allow the proxy to conserve resources and support more clients. Clients should not advertise support for port sharing unless they are willing to share the same 4-tuple when communicating with the target. Sharing ports with other QUIC connections may result in fate-sharing with clients that might be considered to be untrusted or malicious by the target. Such fate-sharing may impact performance or may lead to being misclassified as malicious.

A proxy that does not support port sharing, SHOULD send "Proxy-QUIC-Port-Sharing" with a value of "?0". Doing so allows clients to stop sending capsules for this extension if forwarding mode is also not supported.

When port sharing is supported and forwarded mode is not, registration of target Connection IDs is permitted, but is not required since port sharing only requires demultiplexing QUIC packets in the target-to-client direction

5. Connection ID Capsules

Connection ID awareness relies on using capsules [HTTP-CAPSULES] to signal addition and removal of Connection IDs. Clients send capsules to let proxies know when Connection IDs on the client-to-target QUIC connection are changing. Proxies send capsules to acknowledge or reject these Connection IDs, and in forwarded mode to let clients know about Virtual Connection IDs to use on the client-to-proxy link.

Note that these capsules do not register contexts. QUIC packets are encoded using HTTP Datagrams with the context ID set to zero as defined in [CONNECT-UDP].

The REGISTER_CLIENT_CID (Section 5.1) and REGISTER_TARGET_CID (Section 5.2) capsule types allow a client to inform the proxy about a new client CID or a new target CID, respectively. These capsule types MUST only be sent by a client. These capsule types share a sequence number space which allows the proxy to limit the number of registrations. The first registration (of either client CID or target CID) has sequence number 0, and subsequent registrations increment the sequence number by 1. Every registration capsule consumes a sequence number, including registrations that are rejected and re-registrations of a previously registered connection ID.

The ACK_CLIENT_CID (Section 5.3) and ACK_TARGET_CID (Section 5.4) capsule types are sent by the proxy to the client to indicate that a mapping was successfully created for a registered connection ID as well as optionally provide the Virtual Connection IDs that can be used in forwarded mode. These capsule types MUST only be sent by a proxy. Note that Virtual Connection IDs are always sent by the proxy in order to avoid possible loop attacks Section 10.3.

The ACK_CLIENT_VCID (Section 5.5) capsule type MUST only be sent by the client and only when forwarded mode is enabled. It is sent by the client to the proxy in response to an ACK_CLIENT_CID capsule to indicate that the client is ready to receive forwarded mode packets with the specified virtual connection ID. The proxy MUST NOT send forwarded mode packets to the client prior to receiving this acknowledgement. This capsule also contains a Stateless Reset Token the client may respond with when receiving forwarded mode packets with the specified virtual connection ID.

The CLOSE_CLIENT_CID and CLOSE_TARGET_CID capsule types (Section 5.6) are used to close or reject connection ID registrations. Each capsule includes a reason code indicating why the connection ID is being closed.

Clients send CLOSE_CLIENT_CID or CLOSE_TARGET_CID capsules to retire connection IDs they no longer need, using the DEFAULT reason code. If a client cannot use the proxy-chosen client VCID (e.g., due to conflict or insufficient length), it can re-register the same client CID with an appropriate reason code to request a new VCID.

Proxies send CLOSE_CLIENT_CID or CLOSE_TARGET_CID capsules only to reject registrations. If a proxy sends CLOSE_CLIENT_CID without having sent an ACK_CLIENT_CID, or if a proxy sends CLOSE_TARGET_CID without having sent an ACK_TARGET_CID, it is rejecting a Connection

ID registration. A proxy MUST NOT send a CLOSE_CLIENT_CID or CLOSE_TARGET_CID capsule for a connection ID that it has already acknowledged. If a client receives such a capsule, it MUST reset the stream with H3_DATAGRAM_ERROR error code.

The MAX_CONNECTION_IDS capsule type Section 5.7 MUST only be sent by the proxy. It indicates to the client the cumulative number of connection ID registrations the client is allowed to request. This allows the proxy to limit the number of active registrations. The initial maximum is 2, allowing the client to send 2 registrations, one with sequence number 0 and another with sequence number 1. MAX_CONNECTION_IDS are only sent to increase the limit. Since the initial limit is 2, a proxy MUST NOT send a MAX_CONNECTION_IDS capsule with a value less than 3. Clients receiving a MAX_CONNECTION_IDS capsule with a value less than 3 MUST reset the stream with H3_DATAGRAM_ERROR error code.

When port sharing Section 4 is supported, the client MUST register and receive acknowledgements for client and target CIDs before using them. Packets with unknown connection IDs received by the proxy on a target-facing sockets that support port sharing MUST be dropped. In order to avoid introducing an additional round trip on setup, a REGISTER_CLIENT_CID capsule SHOULD be sent at the same time as the client’s first flight. If the proxy rejects the client CID, the proxy MUST drop all packets until it has sent an ACK_CLIENT_CID capsule and the client MUST NOT send any packets until receiving an ACK_CLIENT_CID. When port sharing is supported, a proxy SHOULD buffer a reasonable number of incoming packets while waiting for the first REGISTER_CLIENT_CID capsule.

Importantly, registering connection IDs does not introduce any delay in communication between client and target unless port sharing is supported and there is an unlikely client CID collision. The client and target can always communicate over the tunnel without having to wait for registrations to be acknowledged. Forwarded mode requires CID/VCID mappings be communicated and acknowledged, and, as a consequence, cannot be leveraged without a round trip. This is particularly pronounced when chaining proxies because registration happens sequentially. While waiting for forwarded mode to become enabled, clients SHOULD communicate over tunneled mode to avoid end-to-end delays.

5.1. REGISTER_CLIENT_CID

The REGISTER_CLIENT_CID capsule is sent by the client and contains a single connection ID that is the client-provided connection ID on the client-to-target QUIC connection.

```
Register Client CID Capsule {
  Type (i) = see {{iana}} for the value of the capsule type
  Length (i),
  Reason (i),
  Connection ID (0..2040),
}
```

Figure 4: Register Client CID Capsule Format

Reason: The reason for this registration. For initial registrations, this MUST be DEFAULT (0x00), although a different value can be used for future extensions. For re-registrations due to an unusable VCID, this indicates why the previous VCID was rejected. See Section 11.4.

Connection ID: A connection ID being registered, which is between 0 and 255 bytes in length. The length of the connection ID is implied by the length of the capsule. Note that in QUICv1, the length of the Connection ID is limited to 20 bytes, but QUIC invariants allow up to 255 bytes.

5.2. REGISTER_TARGET_CID

The REGISTER_TARGET_CID capsule is sent by the client and includes the target-provided connection ID on the client-to-target QUIC connection, and the corresponding Stateless Reset Token.

```
Register Target CID Capsule {
  Type (i) = see {{iana}} for the value of the capsule type
  Length (i),
  Reason (i),
  Connection ID Length (i)
  Connection ID (0..2040),
  Stateless Reset Token Length (i),
  Stateless Reset Token (..),
}
```

Figure 5: Register Target CID Capsule Format

Reason: The reason for this registration. This MUST be DEFAULT (0x00), although a different value can be used for future extensions. See Section 11.4.

Connection ID Length The length of the connection ID being registered, which is between 0 and 255. Note that in QUICv1, the length of the Connection ID is limited to 20 bytes, but QUIC invariants allow up to 255 bytes.

Connection ID A connection ID being registered whose length is equal to Connection ID Length. This is the real target CID.

Stateless Reset Token Length The length of the target-provided Stateless Reset Token.

Stateless Reset Token The target-provided Stateless Reset token allowing the proxy to correctly recognize Stateless Reset packets to be tunnelled to the client.

5.3. ACK_CLIENT_CID

The ACK_CLIENT_CID capsule is sent by the proxy in response to a REGISTER_CLIENT_CID capsule. It optionally assigns a Virtual Connection ID when forwarded mode is supported.

```
Acknowledge Client CID Capsule {
  Type (i) = see {{iana}} for the value of the capsule type
  Length (i)
  Connection ID Length (i)
  Connection ID (0..2040),
  Virtual Connection ID Length (i)
  Virtual Connection ID (0..2040),
}
```

Figure 6: Acknowledge Client CID Capsule Format

Connection ID Length The length of the connection ID being acknowledged, which is between 0 and 255. Note that in QUICv1, the length of the Connection ID is limited to 20 bytes, but QUIC invariants allow up to 255 bytes.

Connection ID A connection ID being acknowledged whose length is equal to Connection ID Length. This is the real Cilent Connection ID.

Virtual Connection ID Length The length of the client VCID being provided. This MUST be a valid connection ID length for the QUIC version used in the client-to-proxy QUIC connection. When forwarded mode is not negotiated, the length MUST be zero. The Virtual Connection ID Length and Connection ID Length SHOULD be equal when possible to avoid the need to resize packets during replacement. The client VCID Length MUST be at least as large as the Connection ID to reduce the likelihood of connection ID conflicts.

Virtual Connection ID The proxy-chosen connection ID that the proxy

MUST use when sending in forwarded mode. The proxy rewrites forwarded mode packets to contain the correct client VCID prior to sending them to the client.

5.4. ACK_TARGET_CID

The ACK_TARGET_CID capsule is sent by the proxy in response to a REGISTER_TARGET_CID capsule. It optionally assigns a Virtual Connection ID and Stateless Reset Token if forwarded mode is enabled.

```
Acknowledge Target CID Capsule {
  Type (i) = see {{iana}} for the value of the capsule type
  Length (i)
  Connection ID Length (i)
  Connection ID (0..2040),
  Virtual Connection ID Length (i)
  Virtual Connection ID (0..2040),
  Stateless Reset Token Length (i),
  Stateless Reset Token (..),
}
```

Figure 7: Acknowledge Target CID Capsule Format

Connection ID Length The length of the connection ID being acknowledged, which is between 0 and 255. Note that in QUICv1, the length of the Connection ID is limited to 20 bytes, but QUIC invariants allow up to 255 bytes.

Connection ID A connection ID being acknowledged whose length is equal to Connection ID Length. This is the real target CID.

Virtual Connection ID Length The length of the target VCID being provided. This MUST be a valid connection ID length for the QUIC version used in the client-to-proxy QUIC connection. When forwarded mode is not negotiated, the length MUST be zero. The Virtual Connection ID Length and Connection ID Length SHOULD be equal when possible to avoid the need to resize packets during replacement.

Virtual Connection ID The proxy-chosen connection ID that the client MUST use when sending in forwarded mode. The proxy rewrites forwarded mode packets to contain the correct target CID prior to sending them.

Stateless Reset Token Length The length of the Stateless Reset Token sent by the proxy in response to forwarded mode packets in order to reset the client-to-target QUIC connection. When forwarded mode is not negotiated, the length MUST be zero. Proxies choosing

not to support stateless resets MAY set the length to zero. Clients receiving a zero-length stateless reset token MUST ignore it.

Stateless Reset Token A Stateless Reset Token allowing reset of the client-to-target connection in response to client-to-target forwarded mode packets.

5.5. ACK_CLIENT_VCID

The ACK_CLIENT_VCID capsule type is sent by the client in response to an ACK_CLIENT_CID capsule that contains a virtual connection ID.

```
Acknowledge Client VCID Capsule {
  Type (i) = see {{iana}} for the value of the capsule type
  Length (i)
  Connection ID Length (i)
  Connection ID (0..2040),
  Virtual Connection ID Length (i)
  Virtual Connection ID (0..2040),
  Stateless Reset Token Length (i),
  Stateless Reset Token (..),
}
```

Figure 8: Acknowledge Client VCID Capsule Format

Connection ID Length The length of the connection ID being acknowledged, which is between 0 and 255. Note that in QUICv1, the length of the Connection ID is limited to 20 bytes, but QUIC invariants allow up to 255 bytes.

Connection ID A connection ID being acknowledged whose length is equal to Connection ID Length. This is the real Cilent Connection ID.

Virtual Connection ID Length The length of the client VCID being acknowledged.

Virtual Connection ID The proxy-chosen virtual connection ID being acknowledged whose length is equal to Virtual Connection ID Length.

Stateless Reset Token Length The length of the Stateless Reset Token that may be sent by the client in response to forwarded mode packets to reset the client-to-target connection. Clients choosing not to support stateless resets MAY set the length to zero. Proxies receiving a zero-length stateless reset token MUST ignore it.

Stateless Reset Token A Stateless Reset Token allowing reset of the target-to-client forwarding rule in response to target-to-client forwarded mode packets.

5.6. CLOSE_CLIENT_CID and CLOSE_TARGET_CID

CLOSE_CLIENT_CID and CLOSE_TARGET_CID capsule types include a reason code and a connection ID. Clients send these capsules to retire connection IDs they no longer need. Proxies send these capsules to reject connection ID registrations.

```
Close CID Capsule {
  Type (i) = see {{iana}} for the values of the capsule types
  Length (i),
  Reason (i),
  Connection ID (0..2040),
}
```

Figure 9: Close CID Capsule Format

Reason: The reason for closing or rejecting the connection ID registration. See Section 11.4 for the list of reason codes.

Connection ID: A connection ID being closed, which is between 0 and 255 bytes in length. The length of the connection ID is implied by the length of the capsule. Note that in QUICv1, the length of the Connection ID is limited to 20 bytes, but QUIC invariants allow up to 255 bytes.

5.7. MAX_CONNECTION_IDS

The MAX_CONNECTION_IDS capsule is sent by the proxy to the client to define the cumulative number of connection ID registrations the client is allowed to request.

```
Maximum Connection IDs Capsule {
  Type (i) = see {{iana}} for the value of the capsule type
  Length (i)
  Maximum Connection IDs (i)
}
```

Figure 10: Maximum Connection IDs Capsule Format

Maximum Connection IDs A count of the cumulative number of connection ID registrations the client is allowed to request. For example, if the value is 4, the maximum allowed sequence number for a connection ID registration would be 3.

The value advertised in the capsule MUST be greater than any value previously sent in a MAX_CONNECTION_IDS capsule, and MUST be greater than the initial allowed limit of 2. Thus, any value sent in the capsule MUST be at least 3.

Clients receiving a MAX_CONNECTION_IDS capsule with an invalid value MUST reset the stream with H3_DATAGRAM_ERROR error code.

5.8. Detecting Conflicts

In order to be able to route packets correctly in both tunnelled and forwarded mode, proxies check for conflicts before creating a new CID mapping. If a conflict is detected, the proxy will reject the client's registration using a CLOSE_CLIENT_CID or CLOSE_TARGET_CID capsule with the CONFLICT reason code.

Two 4-tuples conflict if and only if all members of the 4-tuple (local IP address, local UDP port, remote IP address, and remote UDP port) are identical.

Two Connection IDs conflict if and only if one Connection ID is equal to or a prefix of another. For example, a zero-length Connection ID conflicts with all connection IDs. This definition of a conflict originates from the fact that QUIC short headers do not carry the length of the Destination Connection ID field, and therefore if two short headers with different Destination Connection IDs are received on a shared 4-tuple, one being a prefix of the other prevents the receiver from identifying which mapping this corresponds to.

The proxy treats two mappings as being in conflict when a conflict is detected for all elements on the left side of the mapping diagrams above.

Since very short Connection IDs are more likely to lead to conflicts, particularly zero-length Connection IDs, a proxy MAY choose to reject registrations for very short Connection IDs using the TOO_SHORT reason code.

5.9. Client Considerations

The client sends a REGISTER_CLIENT_CID capsule before it advertises a new client CID to the target, and a REGISTER_TARGET_CID capsule when it has received a new target CID for the target. In order to change the connection ID bytes on the wire, a client can solicit new virtual connection IDs by re-registering the same connection IDs. Note that re-registrations consume sequence numbers like any other registration. The client may solicit a new target VCID by sending a REGISTER_TARGET_CID capsule with a previously registered target CID.

Similarly, the client may solicit a new client VCID by sending a REGISTER_CLIENT_CID with a previously registered client CID. The client MUST acknowledge the new client VCID with an ACK_CLIENT_VCID capsule or close the registration. The proxy MUST NOT send in forwarded mode until ACK_CLIENT_VCID has been received. Clients are responsible for changing Virtual Connection IDs when the HTTP stream's network path changes to avoid linkability across network paths. Note that initial REGISTER_CLIENT_CID capsules MAY be sent prior to receiving an HTTP response from the proxy.

If the client receives a VCID it cannot use, it may re-register the same client CID with a reason code indicating why the previous VCID was unusable. The proxy SHOULD use this information to select a more suitable VCID. If the reason is TOO_SHORT, the proxy MUST either select a longer VCID or close the registration. If the reason is CONFLICT, the proxy MUST select a different VCID.

Connection ID registrations are subject to a proxy-advertised limit. Each registration has a corresponding sequence number. The client MUST NOT send a registration capsule with a sequence number greater than what the proxy advertises via the MAX_CONNECTION_IDS capsule. The initial MAX_CONNECTION_IDS value is 1, allowing both sequence numbers 0 and 1 for a total of two registrations without receiving a MAX_CONNECTION_IDS capsule from the proxy.

Clients that cannot register new connection IDs within a reasonable time due to the MAX_CONNECTION_IDS limit SHOULD abort the proxied connection by resetting the HTTP stream with error code NO_ERROR. This may happen, for example, if the target server sends a NEW_CONNECTION_ID frame with Sequence Number and Retire Prior To equal to the same value.

Clients can cease receiving with forwarded mode over an existing tunnel while retaining the same client-to-target connection by creating a new tunnel with "Proxy-QUIC-Forwarding" set to "?0" and migrating the client-to-target connection.

5.9.1. New Proxied Connection Setup

To initiate QUIC-aware proxying, the client sends a REGISTER_CLIENT_CID capsule containing the initial client CID that the client has advertised to the target.

If the mapping is created successfully, the client will receive a ACK_CLIENT_CID capsule that contains the same client CID that was requested as well as a client VCID that the client MUST use when sending forwarded mode packets, assuming forwarded mode is supported.

If forwarded mode is supported, the client MUST respond with an ACK_CLIENT_VCID to signal to the proxy that it may start sending forwarded mode packets. If forwarded mode is not supported, an ACK_CLIENT_VCID capsule MUST NOT be sent.

Since clients are always aware whether or not they are using a QUIC proxy, clients are expected to cooperate with proxies in selecting client CIDs. A proxy detects a conflict when it is not able to create a unique mapping using the client CID (Section 5.8). It can reject registrations that would cause a conflict by replying with a CLOSE_CLIENT_CID capsule with the CONFLICT reason code. Proxies may also reject registrations for short CIDs using the TOO_SHORT reason code. In order to avoid rejections, clients SHOULD select client CIDs of at least 8 bytes in length with unpredictable values. A client also SHOULD NOT select a client CID that matches the ID used for the QUIC connection to the proxy, as this inherently creates a conflict.

If the rejection reason was CONFLICT, the client MUST select a new Connection ID before sending a new registration request, and generate a new packet. For example, if a client is sending a QUIC Initial packet and chooses a Connection ID that conflicts with an existing mapping to the same target server, it will need to generate a new QUIC Initial. If the rejection reason was TOO_SHORT, the client MUST select a longer Connection ID before retrying.

5.9.2. Adding New Client Connection IDs

Since QUIC connection IDs are chosen by the receiver, an endpoint needs to communicate its chosen connection IDs to its peer before the peer can start using them. In QUICv1, this is performed using the NEW_CONNECTION_ID frame.

Prior to informing the target of a new chosen client CID, the client MUST send a REGISTER_CLIENT_CID capsule to the proxy containing the new client CID.

The client should only inform the target of the new client CID once an ACK_CLIENT_CID capsule is received that contains the echoed connection ID.

If forwarded mode is enabled, the client MUST reply to the ACK_CLIENT_CID with an ACK_CLIENT_VCID capsule with the real and virtual connection IDs along with an optional Stateless Reset Token.

5.10. Proxy Considerations

The proxy MUST reply to each REGISTER_CLIENT_CID capsule with either an ACK_CLIENT_CID or CLOSE_CLIENT_CID capsule containing the Connection ID that was in the registration capsule.

Similarly, the proxy MUST reply to each REGISTER_TARGET_CID capsule with either an ACK_TARGET_CID or CLOSE_TARGET_CID capsule containing the Connection ID that was in the registration capsule.

When a proxy receives a REGISTER_CLIENT_CID with a non-zero reason code, it indicates the client is requesting a new VCID because the previous one was unusable. The proxy SHOULD attempt to address the issue indicated by the reason code when selecting the new VCID.

The proxy then determines the proxy-to-target 4-tuple to associate with the client's request. This will generally involve performing a DNS lookup for the target hostname in the CONNECT request, or finding an existing proxy-to-target 4-tuple to the authority. The proxy-to-target 4-tuple might already be open due to a previous request from this client, or another. If the 4-tuple is not already created, the proxy creates a new one. Proxies can choose to reuse proxy-to-target 4-tuples across multiple UDP proxying requests, or have a unique proxy-to-target 4-tuple for every UDP proxying request. If the client did not send a value of "?1" for the "Proxy-QUIC-Port-Sharing" header, port reuse is not permitted and the proxy MUST allocate a new UDP 4-tuple.

If a proxy reuses proxy-to-target 4-tuples, it SHOULD store which authorities (which could be a domain name or IP address literal) are being accessed over a particular proxy-to-target 4-tuple so it can avoid performing a new DNS query and potentially choosing a different target server IP address which could map to a different target server.

Proxy-to-target 4-tuples MUST NOT be reused across QUIC and non-QUIC UDP proxy requests, since it might not be possible to correctly demultiplex or direct the traffic. Any packets received on a proxy-to-target 4-tuple used for proxying QUIC that does not correspond to a known CID MUST be dropped.

When the proxy recieves a REGISTER_CLIENT_CID capsule, it is receiving a request to be able to route traffic matching the client CID back to the client using. If the pair of this client CID and the selected proxy-to-target 4-tuple does not create a conflict, the proxy creates the mapping and responds with an ACK_CLIENT_CID capsule. If forwarded mode is enabled, the capsule contains a proxy-chosen client VCID. If forwarded mode is enabled, and after

receiving an ACK_CLIENT_VCID capsule from the client, any packets received by the proxy from the proxy-to-target 4-tuple that match the client CID can be sent to the client after the proxy has replaced the CID with the client VCID and executed the negotiated transform (Section 6.3). If forwarded mode is not supported, the proxy MUST NOT send a client VCID by setting the length to zero. The proxy MUST use tunnelled mode (HTTP Datagram frames) for any long header packets. The proxy SHOULD forward directly to the client for any matching short header packets if forwarding is supported by the client, but the proxy MAY tunnel these packets in HTTP Datagram frames instead. If the mapping would create a conflict, the proxy responds with a CLOSE_CLIENT_CID capsule with the CONFLICT reason code.

When the proxy recieves a REGISTER_TARGET_CID capsule, it is receiving a request to allow the client to forward packets to the target. The proxy generates a target VCID for the client to use when sending packets in forwarded mode. If forwarded mode is not supported, the proxy MUST NOT send a target VCID by setting the length to zero. If forwarded mode is supported, the proxy MUST use a target VCID that does not introduce a conflict with any other Connection ID on the client-to-proxy 4-tuple. The proxy creates the mapping and responds with an ACK_TARGET_CID capsule. Once the successful response is sent, the proxy will forward any short header packets received on the client-to-proxy 4-tuple that use the target VCID using the correct proxy-to-target 4-tuple after first rewriting the target VCID to be the correct target CID and executing the negotiated transform.

Proxies MUST choose unpredictable client and target VCIDs to avoid forwarding loop attacks.

The proxy MUST only forward non-tunnelled packets from the client that are QUIC short header packets (based on the Header Form bit) and have mapped target VCIDs. Packets sent by the client that are forwarded SHOULD be considered as activity for restarting QUIC’s Idle Timeout [QUIC].

In order to permit the client to change client-to-target connection IDs, the proxy SHOULD send MAX_CONNECTION_IDS capsules allowing the client additional connection ID registrations.

5.10.1. Closing Proxy State

For any registration capsule for which the proxy has sent an acknowledgement, the mapping lasts until the client sends a close capsule or either side of the HTTP stream closes.

A client that no longer wants a given Connection ID to be forwarded by the proxy sends a CLOSE_CLIENT_CID or CLOSE_TARGET_CID capsule with the DEFAULT reason code.

If a client's connection to the proxy is terminated for any reason, all mappings associated with all requests are removed.

A proxy can close its proxy-to-target 4-tuple once all UDP proxying requests mapped to that 4-tuple have been removed.

6. Using Forwarded Mode

All packets sent in forwarded mode use a transform in which CIDs are switched into VCIDs, and the contents of packets are either left the same, or modified (Section 6.3).

Forwarded mode also raises special considerations for handling connection maintenance (Section 6.4), connection migration (Section 6.5), server preferred addresses (Section 6.6), ECN markings (Section 6.7), and stateless resets (Section 6.8).

Forwarded mode packets are not part of the QUIC connection the tunnel is managed by, but instead an independent flow of QUIC packets.

6.1. Sending With Forwarded Mode

Support for forwarded mode is determined by the "Proxy-QUIC-Forwarding" header, see Section 3.

Once the client has learned the target server's Connection ID, such as in the response to a QUIC Initial packet, it can send a REGISTER_TARGET_CID capsule containing the target CID to request the ability to forward packets.

The client MUST wait for an ACK_TARGET_CID capsule that contains the echoed connection ID and target VCID before using forwarded mode.

Prior to receiving the proxy server response, the client MUST send short header packets tunnelled in HTTP Datagram frames. The client MAY also choose to tunnel some short header packets even after receiving the successful response.

If the target CID registration is rejected, for example with a CLOSE_TARGET_CID capsule, it MUST NOT forward packets to the requested target CID, but only use tunnelled mode. The registration might also be rejected if the proxy does not support forwarded mode or has it disabled by policy.

QUIC long header packets MUST NOT be forwarded. These packets can only be tunnelled within HTTP Datagram frames to avoid exposing unnecessary connection metadata.

When forwarding, the client sends a QUIC packet with the target VCID in the QUIC short header, using the same 4-tuple between client and proxy that was used for the main QUIC connection between client and proxy.

When forwarding, the proxy sends a QUIC packet with the client VCID in the QUIC short header, using the same 4-tuple between client and proxy that was used for the main QUIC connection between client and proxy.

Prior to sending a forwarded mode packet, the sender MUST replace the Connection ID with the Virtual Connection ID. If the Virtual Connection ID is larger than the Connection ID, the sender MUST extend the length of the packet by the difference between the two lengths, to include the entire Virtual Connection ID. If the Virtual Connection ID is smaller than the Connection ID, the sender MUST shrink the length of the packet by the difference between the two lengths.

Clients and proxies supporting forwarded mode MUST be able to handle Virtual Connection IDs of different lengths than the corresponding Connection IDs.

6.2. Receiving With Forwarded Mode

If the client has indicated support for forwarded mode with the "Proxy-QUIC-Forwarding" header, the proxy MAY use forwarded mode for any client CID for which it has a valid mapping.

Once a client has sent an ACK_CLIENT_VCID capsule to the proxy, it MUST be prepared to receive forwarded short header packets on the 4-tuple between itself and the proxy for the specified client VCID.

The client uses the Destination Connection ID field of the received packet to determine if the packet was originated by the proxy, or merely forwarded from the target. The client replaces the client VCID with the real client CID before processing the packet further.

6.3. Packet Transforms

A packet transform is the procedure applied to encode packets as they are sent on the link between the client and proxy, along with the inverse decode step applied on receipt. When sending packets, the packet transform is applied after replacing a CID with a VCID. When receiving packets, the packet transform is applied before replacing a VCID with a CID.

Simple transforms can be modeled as a function as follows:

Inputs:

1. A QUIC short header packet with a VCID.

2. The mode (encode or decode).

3. The direction (client-to-proxy or proxy-to-client).

4. Any configuration information negotiated at startup.

Output:

* A UDP payload that conforms to the QUIC invariants [RFC8999] and does not modify the Connection ID.

More complex transform behaviors could have internal state, but no such transforms are presented here.

Packet transforms are identified by an IANA-registered name, and negotiated in the HTTP headers (see Section 3). This document defines two initial transforms: the identity transform and the scramble transform.

6.3.1. The identity transform

The identity transform does not modify the packet in any way. When this transform is in use, a global passive adversary can trivially correlate pairs of packets that crossed the forwarder, providing a compact proof that a specific client was communicating to a specific target.

The identity transform is identified by the value "identity" Section 11.3.

Use of this transform is NOT RECOMMENDED if the scramble transform is supported by both the client and the proxy. Implementations MAY choose to not implement or support the identity transform, depending on the use cases and privacy requirements of the deployment.

6.3.2. The scramble transform

The scramble transform implements length-preserving unauthenticated re-encryption of QUIC packets while preserving the QUIC invariants. When the scramble transform is in use, a global passive adversary cannot simply compare the packet contents on both sides of the proxy to link the client and target. However, the scramble transform does not defend against analysis of packet sizes and timing, nor does it protect privacy against an active attacker. For example, an active attacker that is able to modify packets prior to being scrambled by the proxy can manipulate the packets in ways that will cause them to be recognizable (either by interfering with the material that is used for the IV, or by changing sizes or timing of packets).

Deployments that implement the version of the scramble transform defined in this document MUST use the value "scramble-dt". The finalized version is expected to use the reserved value "scramble" Section 11.3.

The scramble transform is initialized using a 32-byte random symmetric key. When offering or selecting this transform, the client and server each generate the key that they will use to encrypt scrambled packets and MUST add it to the "Proxy-QUIC-Forwarding" header in an sf-binary parameter named "scramble-key". If either side receives a scramble transform without the "scramble-key" parameter, forwarded mode MUST be disabled. Note that each side (client and server) generates its own "scramble-key" that it uses when sending scrambled packets; the value received in the "scramble-key" parameter is thus used to unscramble packets received from the peer.

This transform relies on the AES-128 block cipher, which is represented by the syntax AES-ECB(key, plaintext_block) as in [RFC9001]. The corresponding decryption operation is written here as AES-ECB-inv(key, ciphertext_block). It also uses AES in Counter Mode ([SP800-38A], Section 6.5), which is represented by the syntax AES-CTR(key, iv, input) for encryption and decryption (which are identical). In this syntax, iv is an array of 16 bytes containing the initial counter block. The counter is incremented by the standard incrementing function ([SP800-38A], Appendix B.1) on the full block width.

In brief, the transform applies AES in counter mode (AES-CTR) using an initialization vector drawn from the packet, then encrypts the initialization vector with AES-ECB. The detailed procedure is as follows:

1. Let k1, k2 = scramble_key[:16], scramble_key[16:32].

2. Let L be the Connection ID length.

3. Let cid = packet[1:L+1], i.e., the Connection ID.

4. Let iv = packet[L+1:L+17], i.e., the 16 bytes following the Connection ID.

5. Let ctr_input = packet[0] | packet[L+17:].

6. Let ctr_output = AES-CTR(k1, iv, ctr_input).

7. Let header = ctr_output[0] & 0x7F. This ensures that the Header Form bit is zero, as required by the QUIC invariants ([RFC8999], Section 5.2).

8. Encrypt iv with the block cipher: encrypted_iv = AES-ECB(k2, iv).

9. Produce the output packet as:
header | cid | encrypted_iv | ctr_output[1:].

The inverse transform operates as follows:

1. Decrypt the AES-CTR initialization vector:
iv = AES-ECB-inv(k2, packet[L+1:L+17]).

2. Compute the other variables exactly as in the forward transform. (AES-CTR encryption and decryption are identical.)

3. Produce the output: header | cid | iv | ctr_output[1:].

The encryption keys used in this procedure do not depend on the packet contents, so each party only needs to perform AES initialization once for each connection.

NOTE: The security of this arrangement relies on every short-header QUIC packet containing a distinct 16 bytes following the Connection ID. This is true for the original ciphersuites of QUICv1, but it is not guaranteed by the QUIC Invariants. Future ciphersuites and QUIC versions could in principle produce packets that are too short or repeat the values at this location. When using the scramble transform, clients MUST NOT offer any configuration that could cause the client or target to violate this requirement.

6.4. Connection Maintenance in Forwarded Mode

When a client and proxy are using forwarded mode, it is possible that there can be long periods of time in which no ack-eliciting packets (see Section 2 of [QUIC-RETRANSMISSION]) are exchanged between the client and proxy. If these periods extend beyond the effective idle timeout for the client-to-proxy QUIC connection (see Section 10.1 of [QUIC]), the QUIC connection might be closed by the proxy if the proxy does not use forwarded packets as an explicit liveness signal. To avoid this, clients SHOULD send keepalive packets to the proxy before the idle timeouts would be reached, which can be done using a PING frame or another ack-eliciting frame as described in Section 10.1.1 of [QUIC].

6.5. Handling Connection Migration

If a proxy supports QUIC connection migration, it needs to ensure that a migration event does not end up sending too many tunnelled or forwarded packets on a new path prior to path validation.

Specifically, the proxy MUST limit the number of packets that it will proxy to an unvalidated client address to the size of an initial congestion window. Proxies additionally SHOULD pace the rate at which packets are sent over a new path to avoid creating unintentional congestion on the new path.

When operating in forwarded mode, the proxy reconfigures or removes forwarding rules as the network path between the client and proxy changes. In the event of passive migration, the proxy automatically reconfigures forwarding rules to use the latest active and validated network path for the HTTP stream. In the event of active migration, the proxy removes forwarding rules in order to not send packets with the same connection ID bytes over multiple network paths. After initiating active migration, clients are no longer able to send forwarded mode packets since the proxy will have removed forwarding rules. Clients can proceed with tunnelled mode or can request new forwarding rules via REGISTER_CLIENT_CID and REGISTER_TARGET_CID capsules. Each of the acknowledging capsules will contain new virtual connection IDs to prevent packets with the same connection ID

bytes being used over multiple network paths. Note that the client CID and target CID can stay the same while the target VCID and client VCID change.

6.6. Handling Server Preferred Addresses

QUIC allows servers to tell clients about a preferred address the server would like to use (Section 9.6 of [QUIC]). When this happens, the client can migrate to the preferred address.

When a client using a proxy wants to migrate to the preferred address of the target server, it needs to create a new UDP proxying request to the proxy (using the method defined in [CONNECT-UDP]) and using the preferred IP address of the target as the host to which to connect. This is the behavior clients using a proxy will have regardless of using the QUIC-aware mechanisms defined in this document. From the proxy's perspective, the migrating request is separate from the original request.

Note that clients can be aware of the target address being used for the original proxy request by looking at next-hop parameter on a Proxy-Status header sent by the proxy in its response (Section 2.1.2 of [RFC9209]). This allows a client to know if it ought to migrate to the preferred address, or is already connected to the preferred address. To support the ability of clients to do this, proxies implementing this specification SHOULD send the Proxy-Status header in responses and include the next-hop parameter.

6.7. Handling ECN Marking

Explicit Congestion Notification marking [ECN] uses two bits in the IP header to signal congestion from a network to endpoints. When using forwarded mode, the proxy replaces IP headers for packets exchanged between the client and target; these headers can include ECN markings. Proxies SHOULD preserve ECN markings on forwarded packets in both directions, to allow ECN to function end-to-end. If the proxy does not preserve ECN markings, it MUST set ECN marks to zero on the IP headers it generates.

Forwarded mode does not create an IP-in-IP tunnel, so the guidance in [ECN-TUNNEL] about transferring ECN markings between inner and outer IP headers does not apply.

A proxy MAY additionally add ECN markings to signal congestion being experienced on the proxy itself.

Forwarding ECN markings introduces certain active attacks. See Section 10.2 for more detail.

6.8. Stateless Resets for Forwarded Mode QUIC Packets

While the lifecycle of forwarding rules are bound to the lifecycle of the client-to-proxy HTTP stream, a peer may not be aware that the stream has terminated. If the above mappings are lost or removed without the peer's knowledge, they may send forwarded mode packets even though the client or proxy no longer has state for that connection. To allow the client or proxy to reset the client-to-target connection in the absence of the mappings above, a stateless reset token corresponding to the Virtual Connection ID can be provided.

Consider a proxy that initiates closure of a client-to-proxy QUIC connection. If the client is temporarily unresponsive or unreachable, the proxy might have considered the connection closed and removed all connection state (including the stream mappings used for forwarding). If the client never learned about the closure, it might send forwarded mode packets to the proxy, assuming the stream mappings and client-to-proxy connection are still intact. The proxy will receive these forwarded mode packets, but won't have any state corresponding to the destination connection ID in the packet. If the proxy has provided a stateless reset token for the target VCID, it can send a stateless reset packet to quickly notify the client that the client-to-target connection is broken.

6.8.1. Stateless Resets from the Target

Reuse of proxy-to-target 4-tuples is only possible because QUIC connection IDs allow distinguishing packets for multiple QUIC connections received with the same 5-tuple. One exception to this is Stateless Reset packets, in which the connection ID is not used, but rather populated with unpredictable bits followed by a Stateless Reset token, to make it indistinguishable from a regular packet with a short header. In order for the proxy to correctly recognize Stateless Reset packets, the client SHOULD share the Stateless Reset token for each registered target CID. When the proxy receives a Stateless Reset packet, it can send the packet to the client as a tunnelled datagram. Although Stateless Reset packets look like short header packets, they are not technically short header packets and do not contain negotiated connection IDs, and thus are not eligible for forwarded mode.

7. Example Exchange

Consider a client that is establishing a new QUIC connection through the proxy. In this example, the client prefers the scramble transform, but also offers the identity transform. It has selected a client CID of 0x31323334. In order to inform a proxy of the new QUIC client CID, the client also sends a REGISTER_CLIENT_CID capsule.

The client will also send the initial QUIC packet with the Long Header form in an HTTP datagram.

```
Client                                                  Server

STREAM(44): HEADERS             -------->
  :method = CONNECT
  :protocol = connect-udp
  :scheme = https
  :path = /target.example.com/443/
  :authority = proxy.example.org
  proxy-quic-port-sharing = ?1
  proxy-quic-forwarding = ?1; accept-transform="scramble,identity"; \
      scramble-key=:abc...789=:
  capsule-protocol = ?1

STREAM(44): DATA                -------->
  Capsule Type = REGISTER_CLIENT_CID
  Connection ID = 0x31323334

           <--------  STREAM(44): DATA
                        Capsule Type = MAX_CONNECTION_IDS
                        Maximum Sequence Number = 3

DATAGRAM                        -------->
  Quarter Stream ID = 11
  Context ID = 0
  Payload = Encapsulated QUIC initial

           <--------  STREAM(44): HEADERS
                        :status = 200
                        proxy-quic-forwarding = ?1; \
                            transform="scramble"; \
                            scramble-key=:ABC...321=:
                        capsule-protocol = ?1

           <--------  STREAM(44): DATA
                        Capsule Type = ACK_CLIENT_CID
                        Connection ID = 0x31323334
                        Virtual CID = 0x62646668
```

The proxy has acknowledged the client CID and provided a client VCID. Even if there were Short Header packets to send, the proxy cannot send forwarded mode packets because the client hasn't acknowledged the client VCID.

The proxy indicates to the client that it will allow connection ID registrations with sequence numbers 0-3, allowing for registrations beyond the initial maximum of 1.

```
STREAM(44): DATA                --------->
  Capsule Type = ACK_CLIENT_VCID
  Connection ID = 0x31323334
  Virtual CID = 0x62646668
  Stateless Reset Token = Token
```

The client acknowledges the client VCID. The proxy still doesn't have any Short Header Packets to send, but, if it did, it would be able to send with forwarded mode.

```
/* Wait for target server to respond to UDP packet. */

           <--------  DATAGRAM
                        Quarter Stream ID = 11
                        Context ID = 0
                        Payload = Encapsulated QUIC initial

/* All Client -> Target QUIC packets must still be encapsulated  */

DATAGRAM                        --------->
  Quarter Stream ID = 11
  Context ID = 0
  Payload = Encapsulated QUIC packet

/* Forwarded mode packets possible in Target -> Client direction  */

           <--------  UDP Datagram
                        Payload = Forwarded QUIC SH packet
```

The client may receive forwarded mode packets from the proxy with a Virtual client CID of 0x62646668 which it will replace with the real client CID of 0x31323334. All forwarded mode packets sent by the proxy will have been modified to contain the client VCID instead of the client CID, and processed by the negotiated "scramble" packet transform. However, in the unlikely event that a forwarded packet arrives before the proxy's HTTP response, the client will not know which transform the proxy selected. In this case, the client will have to ignore the packet or buffer it until the HTTP response is received.

Once the client learns which Connection ID has been selected by the target server, it can send a new request to the proxy to establish a mapping for forwarding. In this case, that ID is 0x61626364. The client sends the following capsule:

```
STREAM(44): DATA                --------->
  Capsule Type = REGISTER_TARGET_CID
  Connection ID = 0x61626364
  Stateless Reset Token = Token

           <--------  STREAM(44): DATA
                        Capsule Type = ACK_TARGET_CID
                        Connection ID = 0x61626364
                        Virtual Connection ID = 0x123412341234
                        Stateless Reset Token = Token

/* Client -> Target QUIC short header packets may use forwarded mode */

UDP Datagram                     --------->
  Payload = Forwarded QUIC SH packet
```

Upon receiving an ACK_TARGET_CID capsule, the client starts sending Short Header packets with a Destination Connection ID of 0x123412341234 directly to the proxy (not tunnelled), and these are rewritten by the proxy to have the Destination Connection ID 0x61626364 prior to being forwarded directly to the target. In the reverse direction, Short Header packets from the target with a Destination Connection ID of 0x31323334 are modified to replace the Destination Connection ID with the client VCID of 0x62646668 and forwarded directly to the client.

8. Packet Size Considerations

Since Initial QUIC packets must be at least 1200 bytes in length, the HTTP Datagram frames that are used for a QUIC-aware proxy MUST be able to carry at least 1200 bytes.

Additionally, clients that connect to a proxy for purpose of proxying QUIC SHOULD start their connection with a larger packet size than 1200 bytes, to account for the overhead of tunnelling an Initial QUIC packet within an HTTP Datagram frame. If the client does not begin with a larger packet size than 1200 bytes, it will need to perform Path MTU (Maximum Transmission Unit) discovery to discover a larger path size prior to sending any tunnelled Initial QUIC packets.

Once a proxied QUIC connections moves into forwarded mode, the client SHOULD initiate Path MTU discovery to increase its end-to-end MTU.

9. QUIC Version Invariance

QUIC proxies only need to understand the Header Form bit, and the connection ID fields from packets in client-to-target QUIC connections. Since these fields are all invariant across future QUIC versions [INVARIANTS], QUIC proxies can proxy all versions of QUIC.

While QUIC proxies can proxy all versions of QUIC, some optional capabilities are limited to certain versions. Specifically, some of the connection ID registration capsules in Section 5 include a Stateless Reset Token field. This field is defined in [QUIC] and not part of [INVARIANTS]. If a future QUIC version removes or changes the behavior of Stateless Reset Tokens, the Stateless Reset Token field in each capsule MUST have a length of zero.

When forwarding mode is enabled, the client and target MAY negotiate any QUIC version, and MAY send packets of that version through the forwarding path. The proxy does not know what version they are using, so it can only require that these packets conform to the QUIC invariants for short-header packets ([RFC8999], Section 5.2).

QUIC version 1 specifies a Fixed Bit (a.k.a. the "QUIC bit") with a fixed value to support sharing a 5-tuple with other protocols such as DTLS, but the QUIC invariants do not guarantee the value of this bit. Accordingly proxies with forwarding mode enabled MUST NOT rely on this bit for protocol identification, and SHOULD send and accept the grease_quic_bit transport parameter [QUIC-GREASE] to avoid ossification of the forwarding mode path.

10. Security Considerations

Proxies that support this extension SHOULD provide protections to rate-limit or restrict clients from opening an excessive number of proxied connections, so as to limit abuse or use of proxies to launch Denial-of-Service attacks.

Sending QUIC packets by forwarding through a proxy without tunnelling exposes clients to additional information exposure and deanonymization attacks which need to be carefully considered. Analysis should consider both passive and active attackers which may be global or localized to the network paths used on one side of the proxy. The following sections highlight deanonymization risks with using forwarded mode.

10.1. Passive Attacks

A passive attacker aims to deanonymize a client by correlating traffic across both sides of the proxy. When using forwarded mode with the identity packet transform (see Section 6.3.1), such correlation is trivial by matching a subset of QUIC packet bytes as packets enter the proxy on one side and exit on the other. Packet transforms such as scramble mitigate this by cryptographically preventing such byte comparisons (see Section 6.3.2).

Regardless of which packet transform is used, both tunnelled and forwarded mode are still vulnerable to size and timing attacks, without the addition of techniques that go beyond the analysis in this document, such as padding and adding chaff packets. Such techniques could be supported in future packet transforms, subject to additional security analysis.

Unlike tunnelled mode where packets are fully encapsulated in the client-to-proxy connection, clients using forwarded mode to access multiple target servers over the same client-to-proxy connection expose the number of target servers they are communicating with on each connection to passive attackers that can observe the client-to-proxy traffic. This additional metadata revealed on each packet simplifies size and timing attacks.

10.2. Active Attacks

An active attacker is an adversary that can inject, modify, drop, and view packets in the network. Some active attacks have different effects between forwarded mode and tunnelled mode, but active attacks can be used to correlate flows in either mode.

Both tunnelled mode and forwarded mode (regardless of packet transform) are vulnerable to packet injection in the target-to-client direction. An attacker can inject a burst of packets with a known QUIC Connection ID and see which Connection ID is used for the corresponding burst on the proxy-to-client network path.

Forwarded mode is vulnerable to some active attacks that tunneled mode is not. For example, packet injection with a known QUIC Connection ID can also happen in the client-to-proxy direction, which only affects forwarded mode since tunnelled mode sends packets within an authenticated and integrity protected QUIC connection to the proxy (see [RFC9001]). None of the packet transforms defined in this document provide integrity protection. Even if a packet transform did provide integrity protection, attackers can inject replayed packets. Protection against replayed packets is similarly provided by QUIC in tunnelled mode, but not provided by any of the forwarded

mode packet transforms defined in this document. Similarly, Forwarded mode packets are vulnerable to active attacks when [ECN] markings are forwarded. Specifically, an attacker could embed a signal over a series of packets by clearing or setting ECN bits. This attack is possible without injecting, dropping, or modifying the QUIC packet, but instead modifying the packet's IP header.

An active attacker can modify packets in the client-to-proxy direction, which would cause a tunnelling proxy to silently drop packets, while a forwarding proxy would forward the packets. In this way, forwarded mode is less vulnerable to flow recognition based on corrupting a portion of packets in a burst.

10.3. Connection ID Registration Attacks

Chaining of proxies using forwarded mode introduces the risk of forwarding loop attacks. Proxies are known to be vulnerable to one such forwarding loop attack when sharing an IP address and port with other proxies, supporting target-facing port sharing, and proxying to other proxies. Preventing client VCID conflicts across proxies sharing an IP address and port mitigates one such forwarding loop attack. Conflicts can be avoided by partitioning the client VCID space across proxies, using sufficiently long and random values, or by other means.

11. IANA Considerations

11.1. HTTP Header Fields

This document registers the "Proxy-QUIC-Forwarding" and "Proxy-QUIC-Port-Sharing" header fields in the "Hypertext Transfer Protocol (HTTP) Field Name Registry" <https://www.iana.org/assignments/http-fields>.

Field Name	Status	Structured Type	Reference	Comments
Proxy-QUIC-Forwarding	permanent	Item	This document	None
Proxy-QUIC-Port-Sharing	permanent	Item	This document	None

Figure 11: Registered HTTP Header Fields

11.2. Proxy QUIC Forwarding Parameter Names

This document establishes a new registry, "Proxy QUIC Forwarding Parameter Names", for parameter names to use with the "Proxy-QUIC-Forwarding" header field, in <https://www.iana.org/assignments/masque/masque.xhtml>. Registrations in this registry are assigned using the Specification Required policy (Section 4.6 of [IANA-POLICY]).

Parameter Name	Description	Reference	Notes
accept-transform	contains supported transforms	This document	Section {{forwarding-header}}
transform	indicates selected transforms	This document	Section {{forwarding-header}}
scramble-key	contains key for scramble transform	This document	Section {{scramble-transform}}

Figure 12: Initial Proxy QUIC Forwarding Parameter Names

11.3. Packet Transform Names

This document establishes a new registry for packet transform names in <https://www.iana.org/assignments/masque/masque.xhtml> and defines two initial transforms: "identity" and "scramble". Prior to finalization, deployments that implement the version of the scramble transform defined in this document should use the value "scramble-dt". Once the design team proposal is adopted and a new draft is submitted, the wire identifier will become "scramble-XX" where XX is the draft number. Registrations in this registry are assigned using the Specification Required policy (Section 4.6 of [IANA-POLICY]).

Transform Name	Description	Specification	Notes
identity	no transformation	This Document	Section Section 6.3.1
scramble	Reserved (will be used for final version)	This Document	Section Section 6.3.2

Table 1: Initial Packet Transform Names

11.4. CID Capsule Reason Codes

This document establishes a new registry, "CID Capsule Reason Codes", for reason codes used in REGISTER_CLIENT_CID, REGISTER_TARGET_CID, CLOSE_CLIENT_CID, and CLOSE_TARGET_CID capsules, in <https://www.iana.org/assignments/masque/masque.xhtml>. This registry governs a 62-bit space and operates under the QUIC registration policy documented in Section 22.1 of [QUIC]. This new registry includes the common set of fields listed in Section 22.1.1 of [QUIC]. In addition to those common fields, all registrations in this registry MUST include a "Name" field that contains a short name or label for the Reason.

Permanent registrations in this registry are assigned using the Specification Required policy (Section 4.6 of [IANA-POLICY]), except for values between 0x00 and 0x3f (in hexadecimal; inclusive), which are assigned using Standards Action or IESG Approval as defined in Sections 4.9 and 4.10 of [IANA-POLICY].

Value	Name	Description	Specification
0x00	DEFAULT	Normal operation	This Document
0x01	TOO_SHORT	CID/VCID rejected for being too short	This Document
0x02	CONFLICT	CID/VCID conflicts with existing mapping	This Document

Table 2: Initial CID Capsule Reason Codes

11.5. Capsule Types

This document registers six new values in the "HTTP Capsule Types" registry established by [HTTP-CAPSULES]. Note that the codepoints below will be replaced with lower values before publication.

Capule Type	Value	Specification
REGISTER_CLIENT_CID	0xffe700	This Document
REGISTER_TARGET_CID	0xffe701	This Document
ACK_CLIENT_CID	0xffe702	This Document
ACK_CLIENT_VCID	0xffe703	This Document
ACK_TARGET_CID	0xffe704	This Document
CLOSE_CLIENT_CID	0xffe705	This Document
CLOSE_TARGET_CID	0xffe706	This Document
MAX_CONNECTION_IDS	0xffe707	This Document

Table 3: Registered Capsule Types

All of these new entries use the following values for these fields:

Status: provisional (permanent when this document is published)
Reference: This document
Change Controller: IETF
Contact: masque@ietf.org
Notes: None

12. References

12.1. Normative References

[CONNECT-UDP]
Schinazi, D., "Proxying UDP in HTTP", RFC 9298, DOI 10.17487/RFC9298, August 2022, <https://www.rfc-editor.org/rfc/rfc9298>.

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

[HTTP-CAPSULES]
Schinazi, D. and L. Pardue, "HTTP Datagrams and the Capsule Protocol", RFC 9297, DOI 10.17487/RFC9297, August 2022, <https://www.rfc-editor.org/rfc/rfc9297>.

[HTTP3] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <https://www.rfc-editor.org/rfc/rfc9114>.

[IANA-POLICY] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

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

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

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

[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/rfc/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/rfc/rfc8174>.

[RFC8941] Nottingham, M. and P. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <https://www.rfc-editor.org/rfc/rfc8941>.

[SP800-38A] Dworkin, M., "Recommendation for Block Cipher Modes of Operation: Methods and Techniques", 1 December 2001, <https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38a.pdf>.

12.2. Informative References

[ECN-TUNNEL]
Briscoe, B., "Tunnelling of Explicit Congestion Notification", RFC 6040, DOI 10.17487/RFC6040, November 2010, <https://www.rfc-editor.org/rfc/rfc6040>.

[QUIC-GREASE]
Schinazi, D. and L. Pardue, "HTTP Datagrams and the Capsule Protocol", RFC 9297, DOI 10.17487/RFC9297, August 2022, <https://www.rfc-editor.org/rfc/rfc9297>.

[QUIC-LB] Duke, M., Banks, N., and C. Huitema, "QUIC-LB: Generating Routable QUIC Connection IDs", Work in Progress, Internet-Draft, draft-ietf-quic-load-balancers-21, 27 August 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-load-balancers-21>.

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

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

[RFC9209] Nottingham, M. and P. Sikora, "The Proxy-Status HTTP Response Header Field", RFC 9209, DOI 10.17487/RFC9209, June 2022, <https://www.rfc-editor.org/rfc/rfc9209>.

Appendix A. Appendix A. Example of a Forwarded Mode QUIC Packet

The following is an example of a QUIC packet that could have been sent by a client or proxy in Forwarded Mode.

Original QUIC Destination Connection ID

002e9184cb0022ca7aecf1128c91d809e1b6853f

Virtual QUIC Destination Connection ID

0123456789abcdef0123456789abcdef01234567

Original QUIC Packet Generated for a Peer.

50002e9184cb0022ca7aecf1128c91d8
09e1b6853f1ba3bed7043a2163202304
8def32f4f8f260c290490413d24ea6

Forwarded mode packet with the identity transform

```
500123456789abcdef0123456789abcd
ef012345671ba3bed7043a2163202304
8def32f4f8f260c290490413d24ea6

Forwarded mode packet with scramble key
f13a915f96fb8919d9d8655488ffea5778cac8cffbc27cd38c173bcbad955cff

320123456789abcdef0123456789abcd
ef012345678ebe6906e16ec5fc90a02c
0109994c3fed03f9d5d88c5f408bb6
```

Acknowledgments

Thanks to Lucas Pardue, Ryan Hamilton, and Mirja K端hlewind for their inputs on this document. The scramble transform design came out of the encryption design team whose members were Antoine Fressancourt, Mirja Kuehlwind, Tiru Reddy, Ben Schwartz, and the authors of this document. The authors would also like to thank Martin Duke for identifying a proxy looping vulnerability that influenced the capsule registration design.

Authors' Addresses

Tommy Pauly
Apple Inc.
One Apple Park Way
Cupertino, California 95014,
United States of America
Email: tpauly@apple.com

Eric Rosenberg
Apple Inc.
One Apple Park Way
Cupertino, California 95014,
United States of America
Email: eric_rosenberg@apple.com

David Schinazi
Google LLC
1600 Amphitheatre Parkway
Mountain View, California 94043,
United States of America
Email: dschinazi.ietf@gmail.com