

QUIC
Internet-Draft
Intended status: Standards Track
Expires: September 10, 2020

M. Duke
F5 Networks, Inc.
N. Banks
Microsoft
March 9, 2020

QUIC-LB: Generating Routable QUIC Connection IDs
draft-ietf-quic-load-balancers-02

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.

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 September 10, 2020.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of

Internet-Draft

QUIC-LB

March 2020

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	3
1.1.	Terminology	4
2.	Protocol Objectives	5
2.1.	Simplicity	5
2.2.	Security	5
3.	First CID octet	5
3.1.	Config Rotation	6
3.2.	Configuration Failover	6
3.3.	Length Self-Description	7
4.	Routing Algorithms	7
4.1.	Plaintext CID Algorithm	8
4.1.1.	Configuration Agent Actions	8
4.1.2.	Load Balancer Actions	9
4.1.3.	Server Actions	9
4.2.	Obfuscated CID Algorithm	9
4.2.1.	Configuration Agent Actions	9
4.2.2.	Load Balancer Actions	10
4.2.3.	Server Actions	10
4.3.	Stream Cipher CID Algorithm	10
4.3.1.	Configuration Agent Actions	11
4.3.2.	Load Balancer Actions	11
4.3.3.	Server Actions	12
4.4.	Block Cipher CID Algorithm	12
4.4.1.	Configuration Agent Actions	12
4.4.2.	Load Balancer Actions	13
4.4.3.	Server Actions	13
5.	Retry Service	13
5.1.	Common Requirements	14
5.2.	No-Shared-State Retry Service	14
5.2.1.	Configuration Agent Actions	14
5.2.2.	Service Requirements	15
5.2.3.	Server Requirements	16
5.3.	Shared-State Retry Service	17
5.3.1.	Configuration Agent Actions	18

5.3.2.	Service Requirements	18
5.3.3.	Server Requirements	18
6.	Configuration Requirements	19
7.	Additional Use Cases	20
7.1.	Load balancer chains	20

7.2.	Moving connections between servers	21
8.	Security Considerations	21
8.1.	Attackers not between the load balancer and server	21
8.2.	Attackers between the load balancer and server	22
9.	IANA Considerations	22
10.	References	22
10.1.	Normative References	22
10.2.	Informative References	22
Appendix A.	Load Balancer Test Vectors	22
A.1.	Obfuscated Connection ID Algorithm	23
A.2.	Stream Cipher Connection ID Algorithm	24
A.3.	Block Cipher Connection ID Algorithm	25
Appendix B.	Acknowledgments	26
Appendix C.	Change Log	26
C.1.	since-draft-ietf-quic-load-balancers-01	26
C.2.	since-draft-ietf-quic-load-balancers-00	26
C.3.	Since draft-duke-quic-load-balancers-06	26
C.4.	Since draft-duke-quic-load-balancers-05	26
C.5.	Since draft-duke-quic-load-balancers-04	26
C.6.	Since draft-duke-quic-load-balancers-03	27
C.7.	Since draft-duke-quic-load-balancers-02	27
C.8.	Since draft-duke-quic-load-balancers-01	27
C.9.	Since draft-duke-quic-load-balancers-00	27
Authors' Addresses	27

[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.

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.

[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 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 the configuration phase of that ID. QUIC-LB messages indicate the phase of the algorithm and parameters that they encode.

A new configuration may change one or more parameters of the old configuration, or change the algorithm used.

It is possible for servers to have mutually exclusive sets of supported algorithms, or for a transition from one algorithm to another to result in Fail Payloads. The four states encoded in these two bits allow two mutually exclusive server pools to coexist, and for each of them to transition to a new set of parameters.

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.

Servers **MUST NOT** generate new connection IDs using an old configuration when it has sent an Ack payload for a new configuration.

Load balancers **SHOULD NOT** use a codepoint to represent a new configuration until it takes precautions to make sure that all connections using IDs with an old configuration at that codepoint have closed or transitioned. They **MAY** drop connection IDs with the old configuration after a reasonable interval to accelerate this process.

[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_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.

[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":

- o The DCID might not be long enough for the routing algorithm to process.
- o The extracted server mapping might not correspond to an active server.
- o A field that should be all zeroes after decryption may not be so.

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. 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.

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

short header.

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.

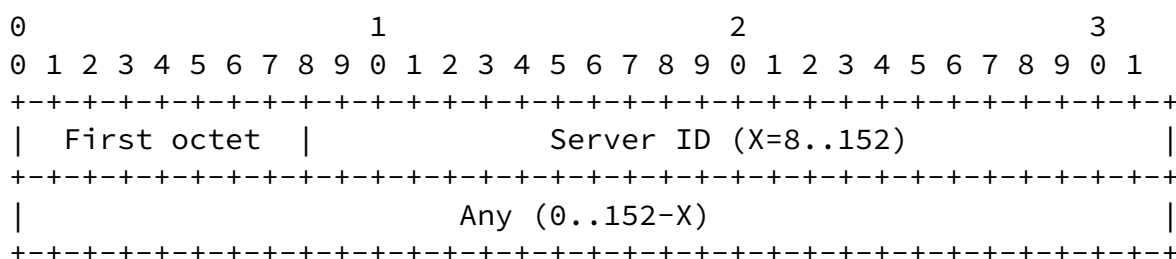


Figure 1: Plaintext CID Format

[4.1.1.](#) Configuration Agent Actions

The configuration agent selects a number of bytes of the server connection ID (SCID) 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.](#) Obfuscated CID Algorithm

The Obfuscated CID Algorithm makes an attempt to obscure the mapping of connections to servers to reduce linkability, while not requiring true encryption and decryption. The format is depicted in the figure below.

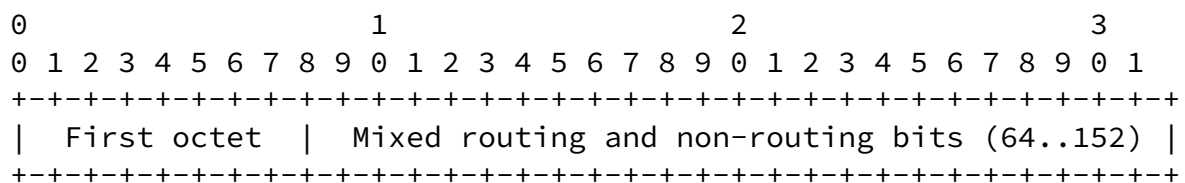


Figure 2: Obfuscated CID Format

[4.2.1.](#) Configuration Agent Actions

The configuration agent selects an arbitrary set of bits of the server connection ID (SCID) that it will use to route to a given server, called the "routing bits". The number of bits **MUST** have enough entropy to have a different code point for each server, and **SHOULD** have enough entropy so that there are many codepoints for each server.

The configuration agent **MUST NOT** select a routing mask with more than 136 routing bits set to 1, which allows for the first octet and up to 2 octets for server purposes in a maximum-length connection ID.

The configuration agent selects a divisor that **MUST** be larger than the number of servers. It **SHOULD** be large enough to accommodate

reasonable increases in the number of servers. The divisor **MUST** be

an odd integer so certain addition operations do not always produce an even number.

The configuration agent also assigns each server a "modulus", an integer between 0 and the divisor minus 1. These **MUST** be unique for each server, and **SHOULD** be distributed across the entire number space between zero and the divisor.

[4.2.2.](#) Load Balancer Actions

Upon receipt of a QUIC packet, the load balancer extracts the selected bits of the SCID and expresses them as an unsigned integer of that length. The load balancer then divides the result by the chosen divisor. The modulus of this operation maps to the modulus for the destination server.

Note that any SCID that contains a server's modulus, plus an arbitrary integer multiple of the divisor, in the routing bits is routable to that server regardless of the contents of the non-routing bits. Outside observers that do not know the divisor or the routing bits will therefore have difficulty identifying that two SCIDs route to the same server.

Note also that not all Connection IDs are necessarily routable, as the computed modulus may not match one assigned to any server. These DCIDs are non-compliant as described above.

[4.2.3.](#) Server Actions

The server chooses a connection ID length. This **MUST** contain all of the routing bits and **MUST** be at least 9 octets to provide adequate entropy.

When a server needs a new connection ID, it adds an arbitrary nonnegative integer multiple of the divisor to its modulus, without exceeding the maximum integer value implied by the number of routing bits. The choice of multiple should appear random within these constraints.

The server encodes the result in the routing bits. It MAY put any other value into bits that used neither for routing nor config rotation. These bits SHOULD appear random to observers.

4.3. Stream Cipher CID Algorithm

The Stream Cipher CID algorithm provides true cryptographic protection, rather than mere obfuscation, at the cost of additional

per-packet processing at the load balancer to decrypt every incoming connection ID. The CID format is depicted below.

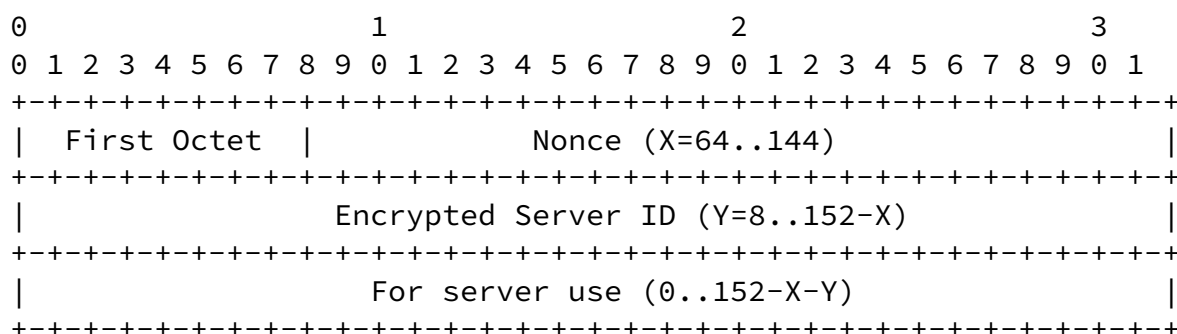


Figure 3: Stream Cipher CID Format

4.3.1. 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 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.

4.3.2. Load Balancer Actions

Upon receipt of a QUIC packet that is not of type Initial or 0-RTT, 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 server ID using 128-bit AES Electronic Codebook (ECB) mode, much like QUIC header protection. 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 encrypted server id.

$$\text{server_id} = \text{encrypted_server_id} \wedge \text{AES-ECB}(\text{key}, \text{padded-nonce})$$

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 of the connection ID for the nonce, zero-pads it to 16 octets using the first 6 octets of the token, and uses this to decrypt the server ID in the twelfth and thirteenth octet.

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

[4.3.3.](#) 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 nonce and additional bits MAY encode additional information, but SHOULD appear essentially random to observers.

The server decrypts the server ID using 128-bit AES Electronic Codebook (ECB) mode, much like QUIC header protection. The nonce octets are zero-padded to 16 octets using the as many of the first octets of the token as necessary. AES-ECB encrypts this nonce using its key to generate a mask which it applies to the server id.

$$\text{encrypted_server_id} = \text{server_id} \wedge \text{AES-ECB}(\text{key}, \text{padded-nonce})$$

[4.4.](#) 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.

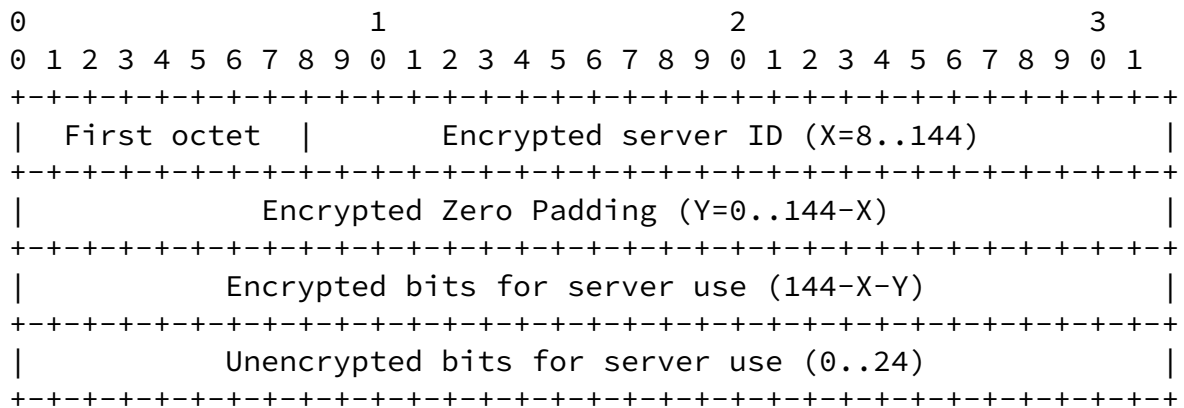


Figure 4: Block Cipher CID Format

[4.4.1.](#) 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.

The configuration agent selects a zero-padding length. This SHOULD be at least four octets to allow detection of non-compliant DCIDs. The server ID and zero- padding length MUST sum to no more than 16 octets. They 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.

[4.4.2.](#) 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, zero padding, and opaque server data in that order. The load balancer uses the server ID octets for routing.

[4.4.3.](#) 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, zeroes into the zero-padding octets, and arbitrary bits into the remaining bits. These arbitrary bits **MAY** encode additional information. Bits in the first, 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.](#) 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_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_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.

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.

[5.1.](#) 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.

[5.2.](#) 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').

[5.2.1.](#) Configuration Agent Actions

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

[5.2.2.](#) 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.

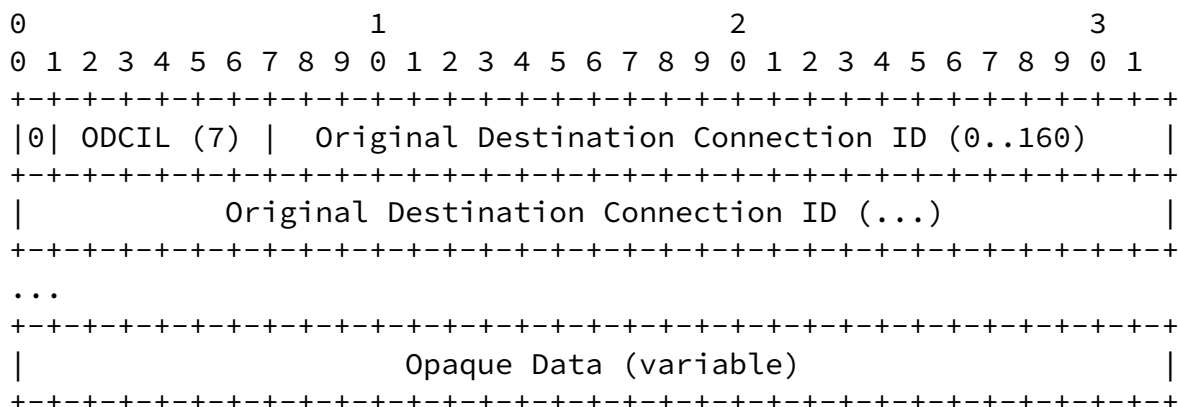


Figure 5: Format of non-shared-state retry service tokens

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.

Original Destination 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 encode a secure hash of the original destination connection ID field to verify that this field has not been edited.

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. It must also verify that the secure hash of the Connect ID is correct. If incorrect, the token is invalid.

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.

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.

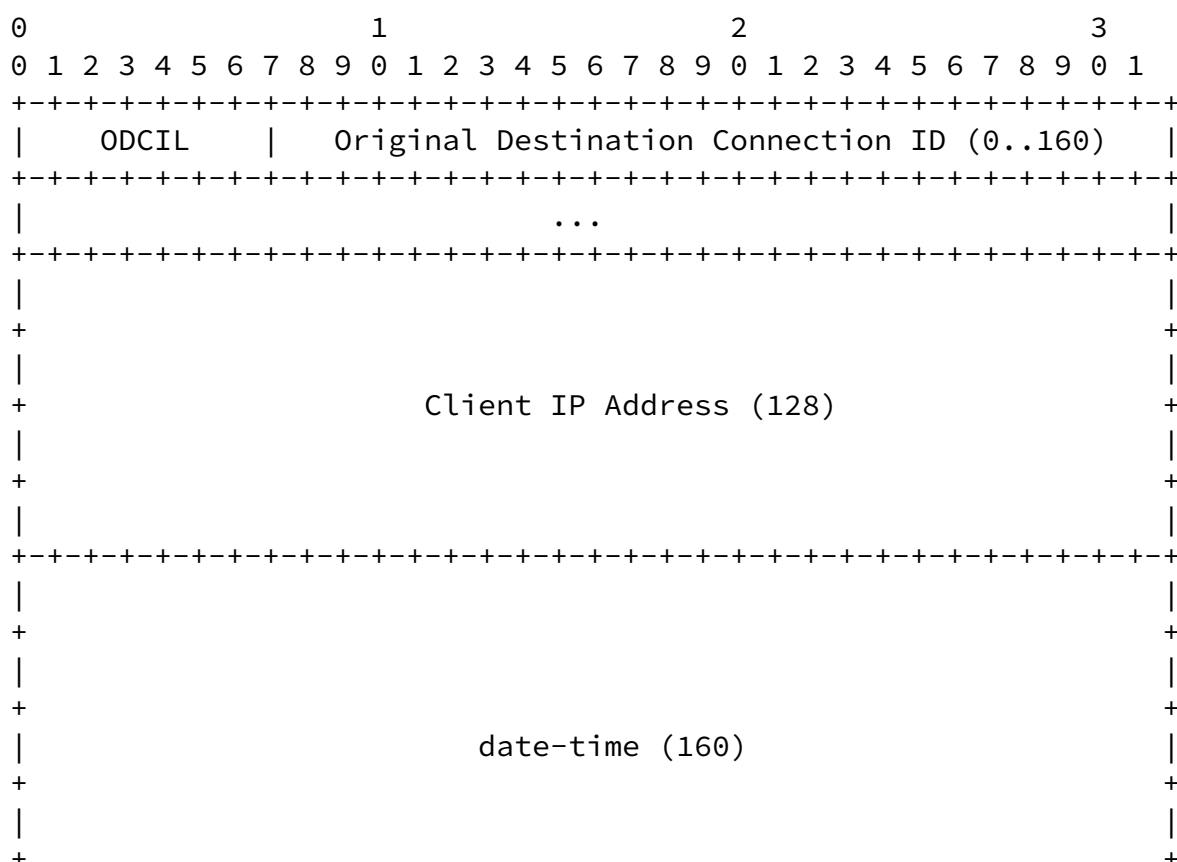
In inactive mode, the service MUST forward all packets that have no token or a token with the first bit set to '1'. It MUST validate all tokens with the first bit set to '0'. If successful, the service MUST forward the packet with the token intact. If unsuccessful, it MUST either drop the packet or forward 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_connection_id` transport parameter, which preserves the normal QUIC signal to the client that there is an unauthorized man in the middle.

[5.2.3.](#) 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, assuming the format described in [Section 5.2.2](#).



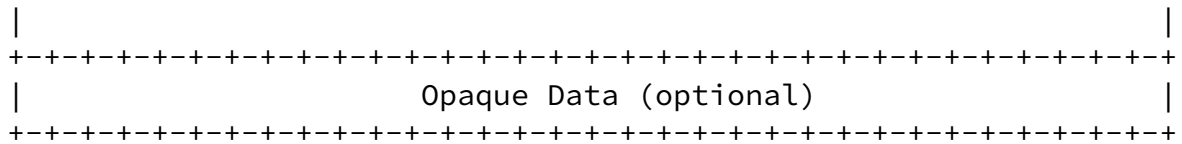


Figure 6: Cleartext format of shared-state retry tokens

The tokens have the following fields:

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

Original Destination Connection ID: This is copied from the field in the client Initial 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_connection_id transport parameter) and NEW_TOKEN frame tokens.

[5.3.1.](#) Configuration Agent Actions

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

[5.3.2.](#) 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.

[5.3.3.](#) 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.

Duke & Banks

Expires September 10, 2020

[Page 18]

Internet-Draft

QUIC-LB

March 2020

A server MUST NOT send a Retry packet in response to an Initial packet that contains a retry token.

[6.](#) 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 Obfuscated CID Routing, this consists of the Routing Bits, Divisor, and Modulus. The Modulus is unique to each server, but the others MUST be 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 depicts 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, obfuscated, 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 obfuscated: struct {
        uint8 routing_bit_mask[19];
        uint16 divisor; /* Must be odd */
    } obfuscated_config;
}
```

```

        uint16 modulus; /* 0..(divisor - 1) */
    } obfuscated_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  zero_padding_length; /* 0..(16 - server_id_length) */
        uint8  server_id[server_id_length];
        uint8  key[16];
    } block_cipher_config;
} routing_algorithm_config;

```

[7.](#) Additional Use Cases

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

[7.1.](#) 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 forwarding address. Note that this solution is extensible to arbitrarily large numbers of load-balancing tiers, as the maximum server ID space is quite large.

[7.2.](#) 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.

8. 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. 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. 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.

8.1. Attackers not between the load balancer and server

Any attacker might open a connection to the server infrastructure and aggressively retire connection IDs 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 Encrypted CID algorithm provides robust entropy to making any sort of linkage. The Obfuscated CID obscures the mapping and prevents trivial brute-force attacks to determine the routing parameters, but does not provide robust protection against sophisticated attacks.

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

8.2. 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.

9. IANA Considerations

There are no IANA requirements.

10. References

10.1. Normative References

[QUIC-TRANSPORT]

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

[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>>.

10.2. 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 connection 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.](#) Obfuscated Connection ID Algorithm

The following section lists a set of OCID load balancer configuration, followed by five CIDs from which the load balancer can extract the server ID.

```
cr_bits 0x0 length_self_encoding: y bitmask ddc2f17788d77e3239b4ea  
divisor 345
```

```
cid 0b72715d4745ce26cca8c750 sid b cid 0b63a1785b6c0b0857225e96 sid  
3f cid 0b66474fa11329e6bb947818 sid 147 cid 0b34bd7c0882deb0252e2a58  
sid ca cid 0b0506ee792163bf9330dc0a sid 14d
```

```
cr_bits 0x1 length_self_encoding: n bitmask  
4855d35f5b88ddada153af61b6707ee646 divisor 301
```

```
cid 542dc4c09e2d548e508dc825bbbca991c131 sid 8 cid  
47988071f9f03a25c322cc6fb1d57151d26f sid 93 cid  
6a13e05071f74cdb7d0dc24d72687b21e1d1 sid c0 cid  
4323c129650c7ee66f37266044ef52e74ffa sid 60 cid  
5e95f77e7e66891b57c224c5781c8c5dd8ba sid 8f
```

```
cr_bits 0x0 length_self_encoding: y bitmask 9f98bd3df66338c2d2c6  
divisor 459
```

```
cid 0ad52216e7798c28340fd6 sid 125 cid 0a78f8ecbd087083639f94 sid 4b  
cid 0ac7e70a5fe6b353b824aa sid 12 cid 0af9612ae5ccba3ef98b81 sid d1  
cid 0a94ab209ea1d2e1e23751 sid 5d
```

```
cr_bits 0x2 length_self_encoding: n bitmask dfba93c4f98f57103f5ae331  
divisor 461
```

```
cid 8b70b8c69e40ef2f3f8937e817 sid d3 cid b1828830ea1789dab13a043795  
sid 44 cid 90604a580baa3eb0a47812e490 sid 137 cid  
a5b4bc309337ff73e143ff6deb sid 9f cid fce75c0a984a79d3b4af40d155 sid  
127
```

```
cr_bits 0x0 length_self_encoding: y bitmask 8320fefc5309f7aa670476  
divisor 379
```

```
cid 0bb110af53dca7295e7d4b7e sid 101 cid 0b0d284cdff364a634a4b93b sid  
e3 cid 0b82ff1555c4a95f9b198090 sid 14e cid 0b7a427d3e508ad71e98b797  
sid 14e cid 0b71d1d4e3e3cd54d435b3fd sid eb
```

Internet-Draft

QUIC-LB

March 2020

[A.2.](#) Stream 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 nonce_len 13 sid_len 1 key
16eff325e8bf8dfabda003543fb845f
```

```
cid 0eb9eb1fc72eed820cf5658cdd7888 sid 9c cid
0e6f4de5beb5aa4170f44104318c5b sid c0 cid
0e78f0325a8e34a40661f51f235906 sid 1d cid
0ef37923f81c32632299bceabd1d92 sid fa cid
0ea30788c012daa94a83865a2c7f28 sid b3
```

```
cr_bits 0x1 length_self_encoding: n nonce_len 9 sid_len 2 key
906220f402ba3bd893ccc4dd9cfc04b0
```

```
cid 7b33366764888138f1465352 sid b839 cid 4329458bbe6cb9befc04bdeb
sid 3b27 cid 61e4e8235c4ebd5442d85bb0 sid bb5c cid
4fd790d1d0cf2b50796cad12 sid 4ecd cid 725325eceaca3528d8c0314b sid
54fd
```

```
cr_bits 0x0 length_self_encoding: y nonce_len 8 sid_len 3 key
0a9b8ccdee977a65e3519693fcd55c8c
```

```
cid 0bfcd0b5727be40af49102e sid 08d342 cid 0b160042b34fe728a9f05376
sid 4d61e0 cid 0b933157fc8c352ee9490ae7 sid 34a912 cid
0b80d1d567aafedb737ed0eb sid 4f2a92 cid 0b3133feac7ae7125b1d0702 sid
1a5db3
```

```
cr_bits 0x0 length_self_encoding: n nonce_len 8 sid_len 4 key
66c5acdb45a40c91da8cfbbdc77c157e
```

```
cid 2da078bbf87c71264879c58a5a sid 20f1e37e cid
04577ce3800cf22ead7f9ba9a5 sid 29e462c4 cid
1a0f6592fcd9167d0aa201e228 sid a0b0fb8a cid
11e4df0eb7db00363b1721e4a4 sid 31f15006 cid
3d54b24c7bd39f081f00f44295 sid 551b8c28
```

```
cr_bits 0x0 length_self_encoding: y nonce_len 12 sid_len 5 key
ba4909a865c19d0234e090197d61bab9
```

```
cid 11325919a7205f4f5e222c2ac94ec3309c1e sid 10f115363a cid
11ca85a9e5d02563ebb119acfacb3007993d sid 4108093aaf cid
1196ef4f0936cb6062b5db441395ef9f3831 sid 383c14e754 cid
11ce3a6611da0e75f59dc8fe3cf4cfc6a61d sid d0da150dbf cid
116bd4cf085659d26b39dd5dd107ae87a694 sid b2945466df
```

[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: fec8 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-01

- o Test vectors for load balancer decoding
- o Deleted remnants of in-band protocol
- o Light edit of Retry Services section
- o Discussed load balancer chains

[C.2.](#) since-draft-ietf-quic-load-balancers-00

- o Removed in-band protocol from the document

[C.3.](#) Since [draft-duke-quic-load-balancers-06](#)

- o Switch to IETF WG draft.

C.4. Since [draft-duke-quic-load-balancers-05](#)

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

C.5. Since [draft-duke-quic-load-balancers-04](#)

- o Added standard for retry services

C.6. Since [draft-duke-quic-load-balancers-03](#)

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

C.7. Since [draft-duke-quic-load-balancers-02](#)

- o Added Config Rotation
- o Added failover mode
- o Tweaks to existing CID algorithms
- o Added Block Cipher CID algorithm
- o Reformatted QUIC-LB packets

[C.8.](#) Since [draft-duke-quic-load-balancers-01](#)

- o Complete rewrite
- o Supports multiple security levels
- o Lightweight messages

[C.9.](#) Since [draft-duke-quic-load-balancers-00](#)

- o Converted to markdown
- o 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