Workgroup: QUIC
Internet-Draft:
draft-ietf-quic-load-balancers-04
Published: 14 August 2020
Intended Status: Standards Track
Expires: 15 February 2021
Authors: M. Duke N. Banks
F5 Networks, Inc. Microsoft

QUIC-LB: Generating Routable QUIC Connection IDs

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

Copyright Notice

Table of Contents

1. Introduction
  1.1. Terminology
2. Protocol Objectives
  2.1. Simplicity
  2.2. Security
3. First CID octet
  3.1. Config Rotation
  3.2. Configuration Failover
  3.3. Length Self-Description
  3.4. Format
4. Routing Algorithms
  4.1. Plaintext CID Algorithm
    4.1.1. Configuration Agent Actions
    4.1.2. Load Balancer Actions
    4.1.3. Server Actions
  4.2. Stream Cipher CID Algorithm
    4.2.1. Configuration Agent Actions
    4.2.2. Load Balancer Actions
    4.2.3. Server Actions
  4.3. Block Cipher CID Algorithm
    4.3.1. Configuration Agent Actions
    4.3.2. Load Balancer Actions
    4.3.3. Server Actions
5. ICMP Processing
6. Retry Service
  6.1. Common Requirements
  6.2. No-Shared-State Retry Service
    6.2.1. Configuration Agent Actions
    6.2.2. Service Requirements
    6.2.3. Server Requirements
  6.3. Shared-State Retry Service
    6.3.1. Configuration Agent Actions
    6.3.2. Service Requirements
    6.3.3. Server Requirements
7. Configuration Requirements
8. Additional Use Cases
  8.1. Load balancer chains
  8.2. Moving connections between servers
9. Version Invariance of QUIC-LB
10. Security Considerations
  10.1. Attackers not between the load balancer and server

10.2. Attackers between the load balancer and server
10.3. Multiple Configuration IDs
10.4. Limited configuration scope
10.5. Stateless Reset Oracle
11. IANA Considerations
12. References
12.1. Normative References
12.2. Informative References
Appendix A. Load Balancer Test Vectors
A.1. Plaintext Connection ID Algorithm
A.2. Stream Cipher Connection ID Algorithm
A.3. Block Cipher Connection ID Algorithm
Appendix B. Acknowledgments
Appendix C. Change Log
C.1. since-draft-ietf-quic-load-balancers-03
C.2. since-draft-ietf-quic-load-balancers-02
C.3. since-draft-ietf-quic-load-balancers-01
C.4. since-draft-ietf-quic-load-balancers-00
C.5. Since draft-duke-quic-load-balancers-06
C.6. Since draft-duke-quic-load-balancers-05
C.7. Since draft-duke-quic-load-balancers-04
C.8. Since draft-duke-quic-load-balancers-03
C.9. Since draft-duke-quic-load-balancers-02
C.10. Since draft-duke-quic-load-balancers-01
C.11. Since draft-duke-quic-load-balancers-00
Authors' Addresses

1. Introduction

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

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

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

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

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

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

1.1. Terminology

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

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

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

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

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

2. Protocol Objectives

2.1. Simplicity

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

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

2.2. Security

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

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

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

3. First CID octet

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

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

3.1. Config Rotation

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

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

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

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

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

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

3.2. Configuration Failover

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

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

3.3. Length Self-Description

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

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

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

3.4. Format

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|C R|  CID Len  |
+-+-+-+-+-+-+-+-+
```

Figure 1: First Octet Format

The first octet has the following fields:

CR: Config Rotation bits.

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

4. Routing Algorithms

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

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

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

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

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

Load balancers MUST forward packets with long headers with non-compliant DCIDs to an active server using an algorithm of its own choosing. It need not coordinate this algorithm with the servers.

The algorithm SHOULD be deterministic over short time scales so that related packets go to the same server. The design of this algorithm SHOULD consider the version-invariant properties of QUIC described in [QUIC-INVARIANTS] to maximize its robustness to future versions of QUIC. For example, a non-compliant DCID might be converted to an integer and divided by the number of servers, with the modulus used to forward the packet. The number of servers is usually consistent on the time scale of a QUIC connection handshake. See also Section 9.

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

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

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

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

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

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

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

4.1. Plaintext CID Algorithm

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

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  First octet  |             Server ID (X=8..152)              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Any (0..152-X)                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2: Plaintext CID Format

4.1.1. Configuration Agent Actions

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

It also assigns a server ID to each server.

4.1.2. Load Balancer Actions

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

4.1.3. Server Actions

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

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

4.2. Stream Cipher CID Algorithm

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

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  First Octet  |            Nonce (X=64..128)                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Encrypted Server ID (Y=8..152-X)              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   For server use (0..152-X-Y)                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3: Stream Cipher CID Format

4.2.1. Configuration Agent Actions

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

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

4.2.2. Load Balancer Actions

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

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

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

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

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

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

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

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

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

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

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

4.2.3. Server Actions

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

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

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

4.3. Block Cipher CID Algorithm

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

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  First octet  |       Encrypted server ID (X=8..128)          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Encrypted bits for server use (128-X)               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Unencrypted bits for server use (0..24)             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4: Block Cipher CID Format

4.3.1. Configuration Agent Actions

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

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

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

4.3.2. Load Balancer Actions

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

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

4.3.3. Server Actions

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

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

5. ICMP Processing

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

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

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

6. Retry Service

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

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

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

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

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

6.1. Common Requirements

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

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

6.2. No-Shared-State Retry Service

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

6.2.1. Configuration Agent Actions

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

6.2.2. Service Requirements

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

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

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

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0| ODCIL (7) |   RSCIL (8)   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Original Destination Connection ID (0..160)            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             ...                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              Retry Source Connection ID (0..160)              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             ...                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Opaque Data (variable)                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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

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

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

RSCIL: The retry source connection ID length.

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

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

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

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

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

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

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

6.2.3. Server Requirements

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

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

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

6.3. Shared-State Retry Service

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

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

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

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    ODCIL    |      RSCIL    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Original Destination Connection ID (0..160)            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             ...                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              Retry Source Connection ID (0..160)              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             ...                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                     Client IP Address (128)                   +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                                                               +
|                          date-time (160)                      |
+                                                               +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Opaque Data (optional)                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 6: Cleartext format of shared-state retry tokens

The tokens have the following fields:

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

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

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

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

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

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

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

6.3.1. Configuration Agent Actions

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

6.3.2. Service Requirements

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

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

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

6.3.3. Server Requirements

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

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

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

7. Configuration Requirements

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

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

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

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

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

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

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

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

8. Additional Use Cases

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

8.1. Load balancer chains

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

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

8.2. Moving connections between servers

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

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

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

9. Version Invariance of QUIC-LB

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

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

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

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

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

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

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

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

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

10. Security Considerations

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

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

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

10.1. Attackers not between the load balancer and server

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

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

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

10.2. Attackers between the load balancer and server

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

10.3. Multiple Configuration IDs

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

10.4. Limited configuration scope

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

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

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

10.5. Stateless Reset Oracle

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

11. IANA Considerations

There are no IANA requirements.

12. References

12.1. Normative References

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

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

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

12.2. Informative References

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

Appendix A. Load Balancer Test Vectors

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

A.1. Plaintext Connection ID Algorithm

TBD

A.2. Stream Cipher Connection ID Algorithm

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

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

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

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

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

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

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

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

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

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

A.3. Block Cipher Connection ID Algorithm

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

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

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

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

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

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

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

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

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

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

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

Appendix B. Acknowledgments

Appendix C. Change Log

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

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

* Improved Config Rotation text

* Added stream cipher test vectors

* Deleted the Obfuscated CID algorithm

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

* Replaced stream cipher algorithm with three-pass version

* Updated Retry format to encode info for required TPs

* Added discussion of version invariance

* Cleaned up text about config rotation

* Added Reset Oracle and limited configuration considerations

* Allow dropped long-header packets for known QUIC versions

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

* Test vectors for load balancer decoding

* Deleted remnants of in-band protocol

* Light edit of Retry Services section

* Discussed load balancer chains

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

* Removed in-band protocol from the document

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

* Switch to IETF WG draft.

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

* Editorial changes

* Made load balancer behavior independent of QUIC version

* Got rid of token in stream cipher encoding, because server might not have it

* Defined "non-compliant DCID" and specified rules for handling them.

* Added psuedocode for config schema

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

* Added standard for retry services

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

* Renamed Plaintext CID algorithm as Obfuscated CID

* Added new Plaintext CID algorithm

* Updated to allow 20B CIDs

* Added self-encoding of CID length

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

* Added Config Rotation

* Added failover mode

* Tweaks to existing CID algorithms

* Added Block Cipher CID algorithm

* Reformatted QUIC-LB packets

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

* Complete rewrite

* Supports multiple security levels

* Lightweight messages

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

* Converted to markdown

* Added variable length connection IDs

Authors' Addresses

Martin Duke
F5 Networks, Inc.

Email: martin.h.duke@gmail.com

Nick Banks
Microsoft

Email: nibanks@microsoft.com