



QUIC handshake and Connection ID

QUIC Interim 2018-01, Melbourne

Status Quo (draft-08)

Client chooses Connection ID (CCID)

- Initial, 0-RTT have CCID
- Retry, Version Negotiation have CCID (for verification)

Server chooses Connection ID (SCID)

- Client and Server Handshakes have Server-chosen Connection ID (SCID)
- All 1-RTT packets have SCID

Client must use 5-tuple to associate SCID with connection

Issues with Design

Server may want to redirect incoming connection to a different server with Retry (Issue [#713](#))

Simultaneous handshakes on a 5-tuple are impossible without trial decryption (Issue [#714](#))

This is the only time connection ID is changed unilaterally

Option 1: Transport Params

Server chosen connection ID goes in transport parameters

Pros:

- Part of the crypto handshake, so it can't be tampered with
- No new frames allowed in the handshake
- Could allow arbitrary length connection IDs

Cons:

- Adds a transport parameter
- Requires exposing transport params before the completing the handshake

PR#[1041](#)

Option 2: TLS Extension

Server chosen connection ID goes in TLS extension

Pros:

- Part of the crypto handshake, so it can't be tampered with
- No new frames allowed in the handshake
- Allow arbitrary length connection IDs

Cons:

- Requires a new TLS extension
- Requires exposing a new TLS extension
- QUIC will likely not allow asymmetric Connection IDs

[draft-rescorla-tls-dtls-connection-id-02](#)

Option 3: NEW_CONNECTION_ID frame

Server chosen connection ID goes in NEW_CONNECTION_ID

Pros:

- All changes of connection ID start with NEW_CONNECTION_ID
 - But this is still different, because clients **MUST** change ID

Cons:

- Bundling a NEW_CONNECTION_ID frame with a crypto stream frame is non-trivial.
- Another frame type is allowed in the handshake
- No protection from on-path elements tampering with the connection ID (not sure why they would, but...)

Questions

- 1) Do we want to change connection ID explicitly?
- 2) Are any of these mechanisms right?
 - a) If not, what mechanism should be used?
- 3) Do we want variable length connection IDs?
 - a) Keep this in your mind for the next slides



QUIC Connection ID Size

QUIC Interim 2018-01, Melbourne

Non-linkable Connection IDs

Goal: Given an existing connection, generate one or more alternate connection IDs which route to the same server.

Challenge #1: They must not be linkable.

Challenge #2: 8-byte connection ID is a draft invariant.

Non-linkable Connection IDs

What can we do?

Alternative #1: Server uses the load balancer as an oracle

Alternative #2: Server and load balancer share algorithm

Non-linkable Connection IDs: 8 bytes

Solution #1: Explicit communication with load balancer

Cons:

- Synchronization may be hard
- Explicit communication is brittle
- Explicit communication may be impossible

Pros:

- Compact (compared to 16 bytes)

Non-linkable Connection IDs: 8 bytes

Solution #2: Shared Algorithm: Format preserving encryption

Cons:

- Can't use simple AES-GCM
- Have to implement key rotation

Pros:

- Compact (compared to 16 bytes)

Non-linkable Connection IDs: 16 bytes

Solution #3: Shared Algorithm:Format preserving encryption

Cons:

- 8 bytes of extra overhead
- Have to Implement Key Rotation
- 'Too much' space - Could people do something bad?

Pros:

- Plenty of space for routing

Options

Option #1: Stick with 8 bytes and make it an invariant

Option #2: Stick with 8 bytes for v1 and allow for longer connection IDs in the invariants

Option #3: Change to 16 byte connection IDs as an invariant

Option #4: Variable length Connection IDs

Once we have direction on these, we can discuss mechanism