

QUIC Handshake Key Discard Alternatives

Eric Rescorla

November 10, 2019

1 Status Quo

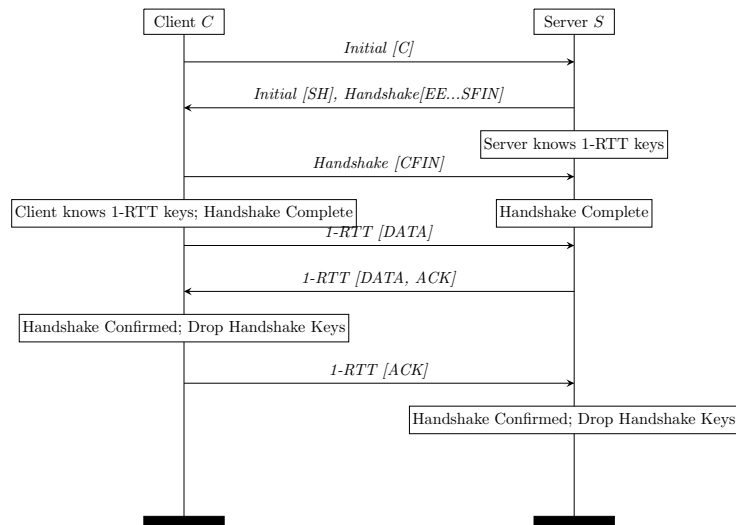


Figure 1: Basic QUIC handshake (without 0.5 RTT data).

Figure 1 shows the basic QUIC handshake along with the various checkpoints. This just for reference and hopefully non-controversial.

This design is known to have several problems, specifically:

1. A dropped ACK can cause the client to retransmit CFIN indefinitely.
2. It is possible for the server to have handshake data outstanding when the client initiates migration.

We show these scenarios below.

1.1 Deadlock with Missing ACKs

[TODO: Diagram needed]

1.2 Outstanding Handshake Data During Migration

Consider the situation shown in

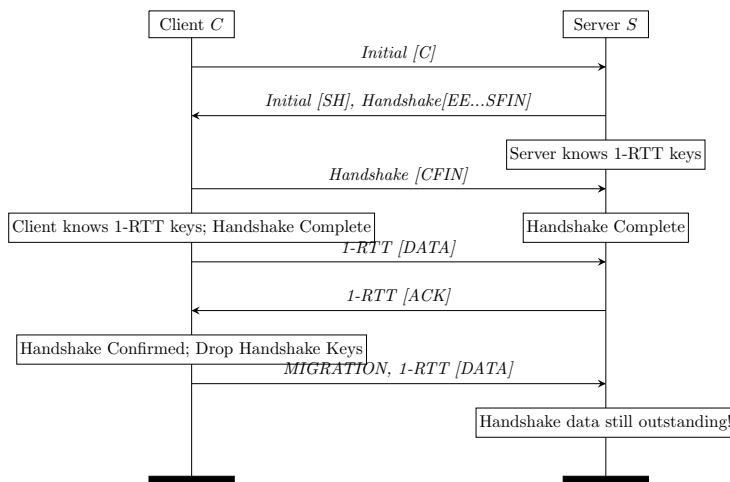


Figure 2: Outstanding handshake data during migration

The issue here is that the server's signal that the handshake is done is an ACK of its 1-RTT data, but there's no guarantee that this precedes the client's migration data, because the client can migrate as soon as it has received an ACK of its 1-RTT data. Even if the server sends its own 1-RTT data in the same packet as its ACK, there is no requirement that the client ACK at the same time as it sends its migration data.

Note that the converse case cannot happen: the client cannot initiate migration until handshake confirmed and it is required to drop the handshake keys at that point.

QUESTION: Do current implementations (especially clients) ACK handshake packets prior to handshake confirmed? If so, this case ought to be quite rare.

2 Never Drop the Keys (PR 3121)

The basic idea here is to (1) never drop the Handshake keys and (2) continue to send ACKs for packets you have received (as agreed upon separately in YUL, you MUST ACK each packet at least once). The result is that the handshake

goes quiescent even if no application data is ever transmitted, though you will not reach “handshake confirmed” in this case, as shown in Figure 3.

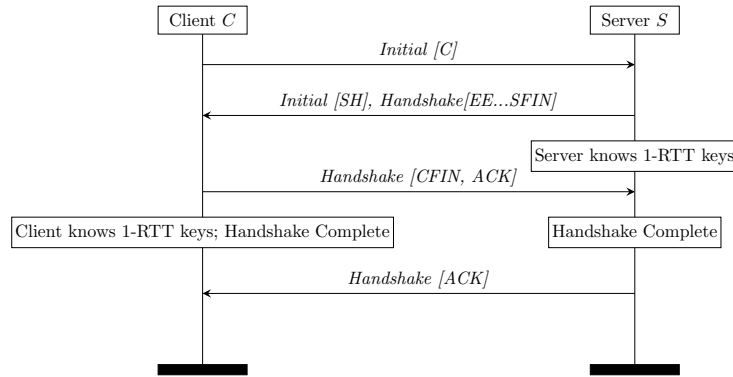


Figure 3: Never drop keys (no data sent)

One obvious question is what happens in the case where you *are* sending data. In the basic case, we would just expect ACK piggybacking on the data, as shown in Figure 4

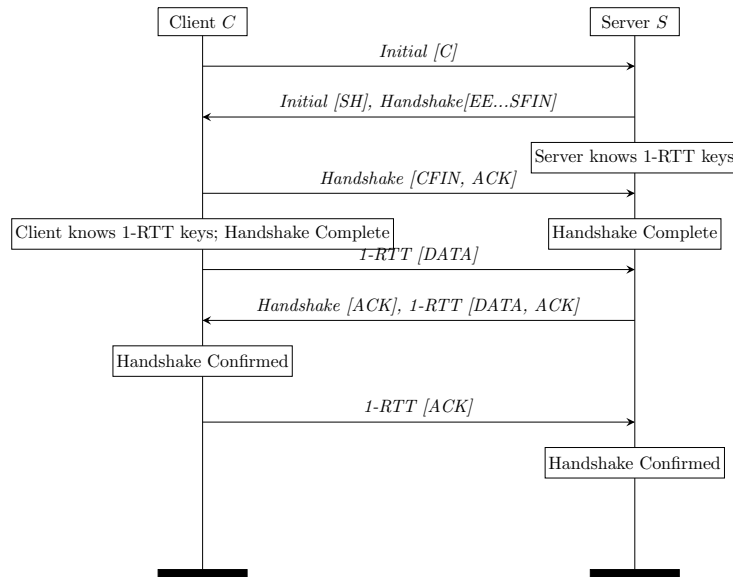


Figure 4: Never drop keys (data sent)

You can also safely implement implicit ACK. At the point where you have

handshake confirmed, you can stop sending any of your Handshake DATA frames (though you still need to ACK any of the peer's Handshake frames, or you will get deadlocks if the peer doesn't implement implicit ACKs.)

Kazuho had raised two concerns about this design:

1. Because handshake confirmed is not guaranteed, key update is confusing.
2. It is possible to have outstanding handshake data at the point where you want to do migration.

I don't really understand point (1). As noted in <https://github.com/quicwg/base-drafts/issues/3212>, the handshake confirmed test for key update is redundant, and so we can just re-code this as "don't initiate key update for key $n + 1$ until you are sure the other side has key n ". And this just lets us have the usual rule without a special case for the handshake \rightarrow 1-RTT transition.

Point (2) is the same issue as is raised in Section 1.2 above. It is possible that this case is less likely with the "never drop keys" approach because the client is required to send ACKs for handshake data. Thus, if the client piggybacks ACKs for the server's handshake packets on its own CFIN (which is natural), the server will never be able to process 1-RTT packets from the client when handshake data is outstanding, in which case the problem goes away. Note that this is separate from the packet loss edge cases, because the client can guarantee this condition.

An alternative approach is to have the server implement implicit ACK where it discards handshake data once it receives any valid 1-RTT data from the client. Note that it still needs to be able to receive handshake data and send ACKs for it, but the client can be required to send that down the initial path and/or implicitly ACK on migration success (which inevitably includes verification of a 1-RTT ACK).