Handshake Invariants

QUIC @IETF 99, Prague

In the beginning…

Client and server don’t share a key

It’s easy to prevent the handshake from completing

What, if anything, do we want to do about this?

https://github.com/quicwg/base-drafts/issues/645

Constraints on the Handshake

DECISION SLIDE

Can we agree that the 5-tuple doesn’t change?

- The server can’t reply from a different source address
- The client can’t continue from a different source address
- … or a different destination address
- Assume that routing won’t change
- … if it does, then no guarantees from the protocol

The current design already assumes this constraint

Taxonomy

An attacker that is on-path can block packets

... and modify or delay them arbitrarily

... nothing to be done about this

What about

... off-path attackers that spoof packets?

... an attacker that can also see packets?

Off-Path Attacks

Standard defense:

Have every packet include proof of receipt for the last

TCP: echo the sequence number (32 bits)

STUN: echo the transaction ID (96 bits)

QUIC: … it’s complicated

QUIC and Off-Path Attacks: Proofs

Version Negotiation echoes the packet number and connection ID from the Client Initial (95 bits)

Server Stateless Retry echoes the packet number and connection ID from the Client Initial (95 bits)

Client Initial after Version Negotiation contains no proof

Client Initial after Server Stateless Retry contains a cookie

Server Plaintext *might* include an ACK frame (31 or 0 bits)

Client Plaintext contains *TLS* ciphertext

This is a bit of a mess

Options for Off-Path Defense

DECISION SLIDE

1. Leave it be
2. Add fields to the long header

   @mikkelfj proposes separate connection IDs for peers
3. Change integrity protection for cleartext packets

   @ekr proposes using HMAC keyed by connection ID*

   Could use AES-GCM: just GHASH or encrypt as well

QUIC

Handshake Errors

A badly formed frame is a connection error

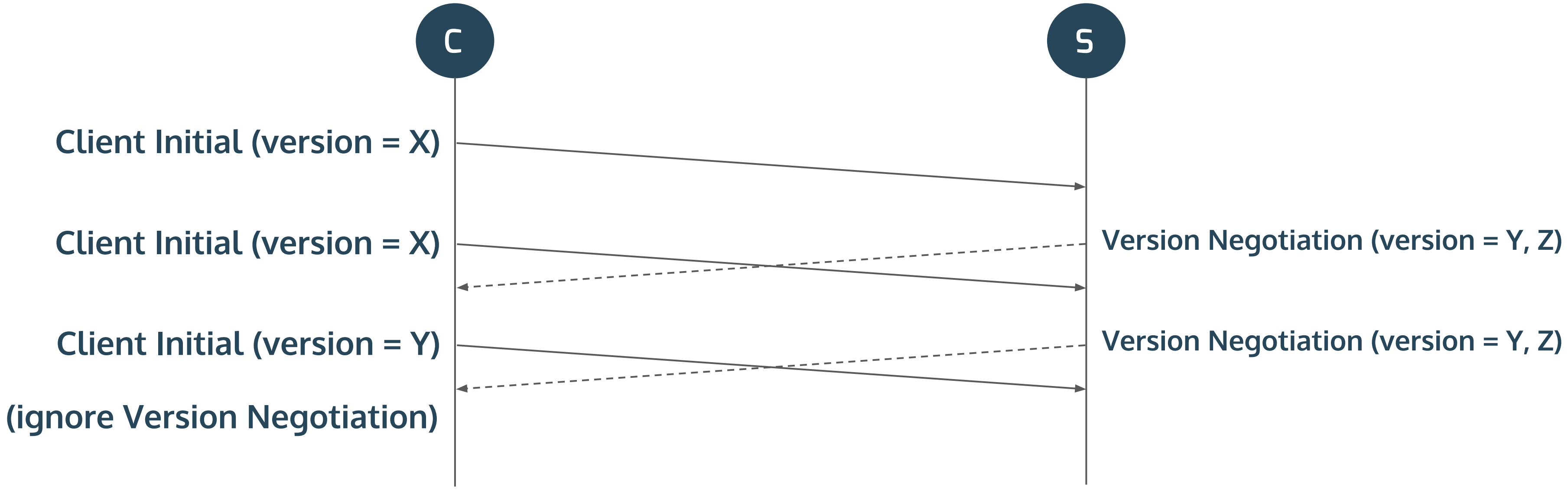
What if there is no connection yet?

… or what if there was a connection before, can we use that?

Do we ignore malformed packets?

At what point does the connection drop?

Example: Version Negotiation


C
S
Client Initial (version = X)
Client Initial (version = X)
Version Negotiation (version = Y, Z)
Client Initial (version = Y)
Version Negotiation (version = Y, Z)
(ignore Version Negotiation)

(This is the existing design.)

What if Version Negotiation is falsified by an attacker?

Proposal

DECISION SLIDE

Rely on defense against off-path attackers

Accept that disruption of the handshake is possible

We can maybe add defense for return visits *later*

QUIC

Signaling Handshake Errors

What happens if the handshake fails?

How do we signal this?

To consider:

- Prior to sending Server Cleartext, the server hasn’t committed any state
- Prior to receiving Server Cleartext, the client doesn’t know if the server has committed state

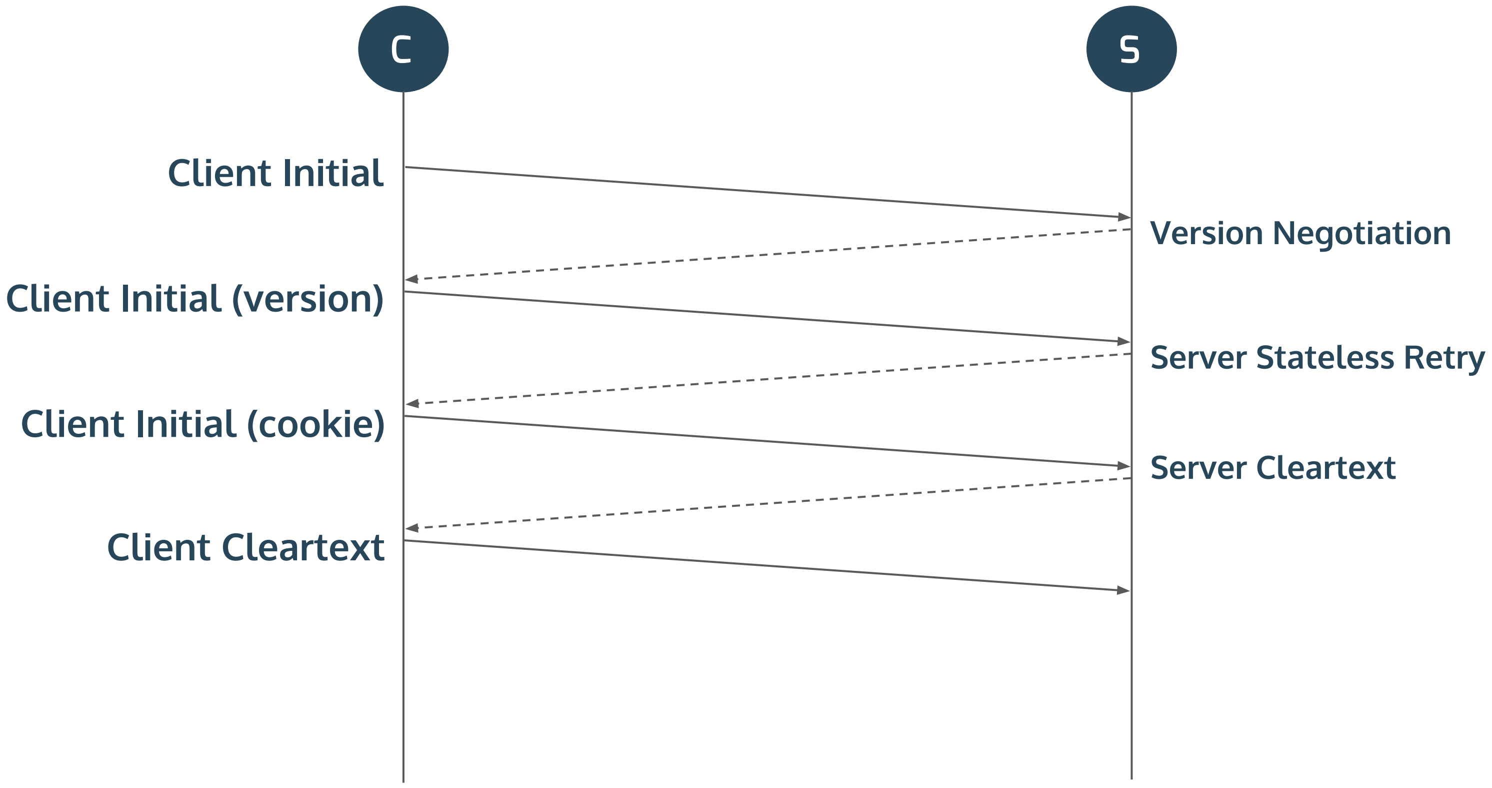
QUIC

Current Practice

We currently mandate almost all of these:

1. Send Version Negotiation (for some Client Initial)
2. Ignore the packet (for some Version Negotiation)
3. Drop the connection silently (for some Client Initial)
4. Send CONNECTION_CLOSE in the clear (prohibited)
5. Send TLS Alerts (Client Cleartext ?)

Extended Handshake



QUIC

Error Conditions

Client Initial
- Wrong version
- Too small
- Malformed (QUIC/TLS)

Client Cleartext
- Malformed (QUIC/TLS)
- Wrong connection ID
- Bad ACK

Wrong Packet Type
- Redundant retransmit
- Something else

Version Negotiation
- Same version
- Malformed
- Wrong PN/Connection ID
- Second attempt

Server Stateless Retry
- Malformed (QUIC/TLS)
- Wrong PN/Connection ID
- Second attempt
- Bad ACK

Server Cleartext
- Malformed (QUIC/TLS)

Wrong Packet Type

Proposed Principles

DECISION SLIDE

If a packet is potentially legitimate

    … and it can be accepted, accept it

    … and it is redundant, ignore it

    … and it is in error, fail

        … use CONNECTION_CLOSE if you can

        … use TLS Alerts if one is provided

If the packet is clearly bad, ignore it*

QUIC