



The QUIC Handshake

QUIC Tech Deep Dive, IETF 115 London
Martin Thomson

Overview

QUIC layering

TLS integration

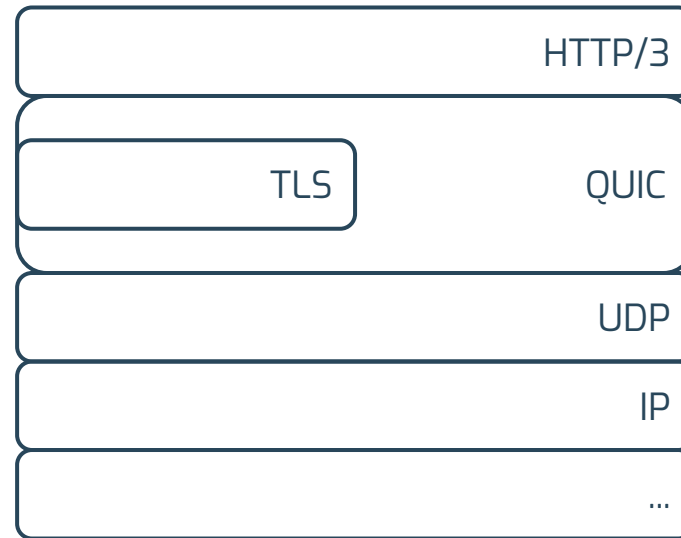
Handshake

Denial of service mitigations

Migration

Connection reset

QUIC



QUIC Handshake Overview

QUIC connection setup does what TCP and TLS do

QUIC uses TLS to do most of the TLS bits

QUIC provides all the TCP bits

including providing TLS with reliable, ordered delivery

Security Goals

Core TLS guarantees

authentication, confidentiality, integrity

Core TCP guarantees

consent to receive/anti-amplification

Better where possible

TLS Handshake

TLS 1.3 is an optimistic handshake

Clients guess key agreement parameters to save 1 RTT

Typical handshake is 3 way / 1.5 RTT

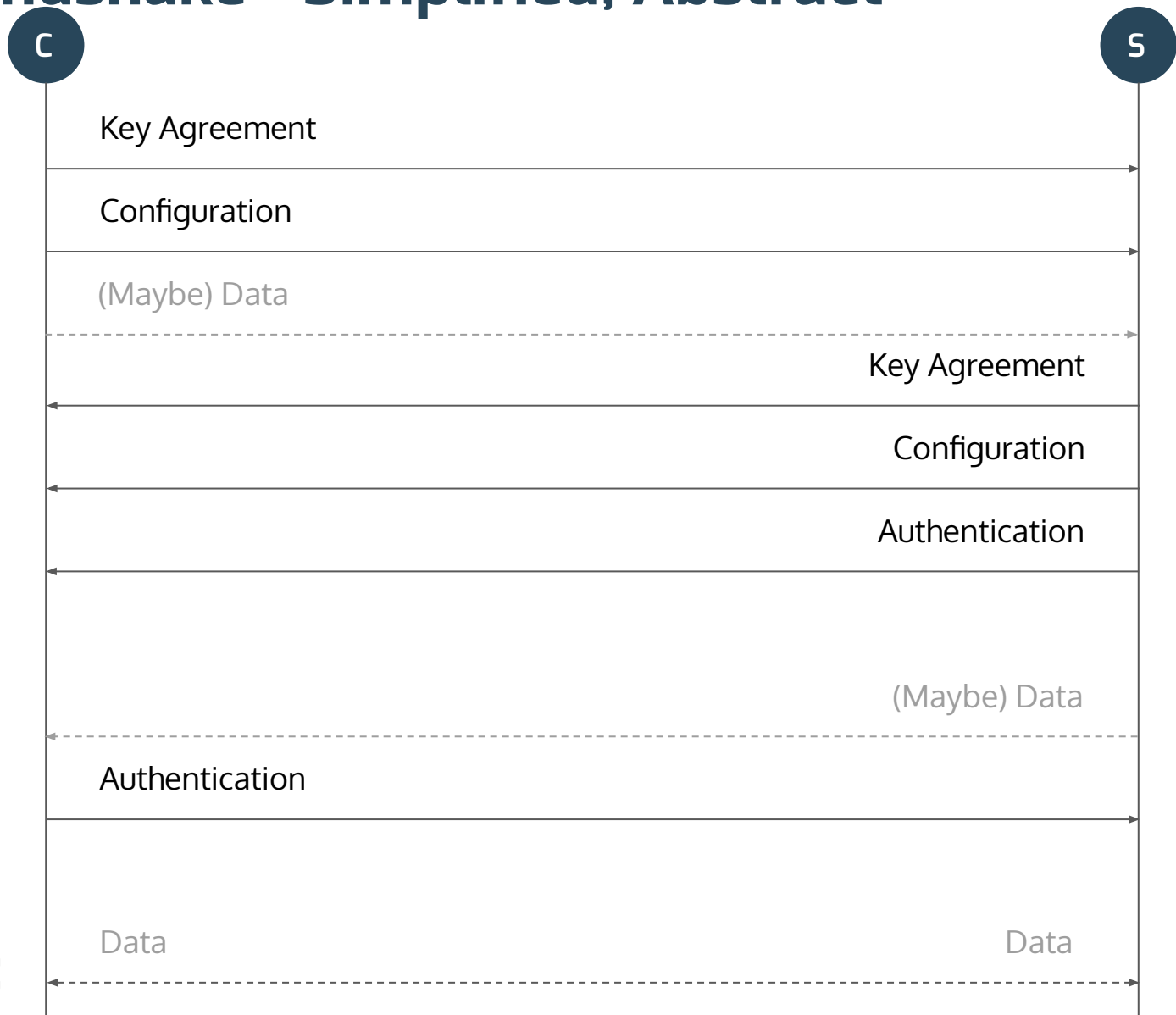
1 extra RTT if the client guesses wrong

Endpoints can - sometimes - send 1 RTT early

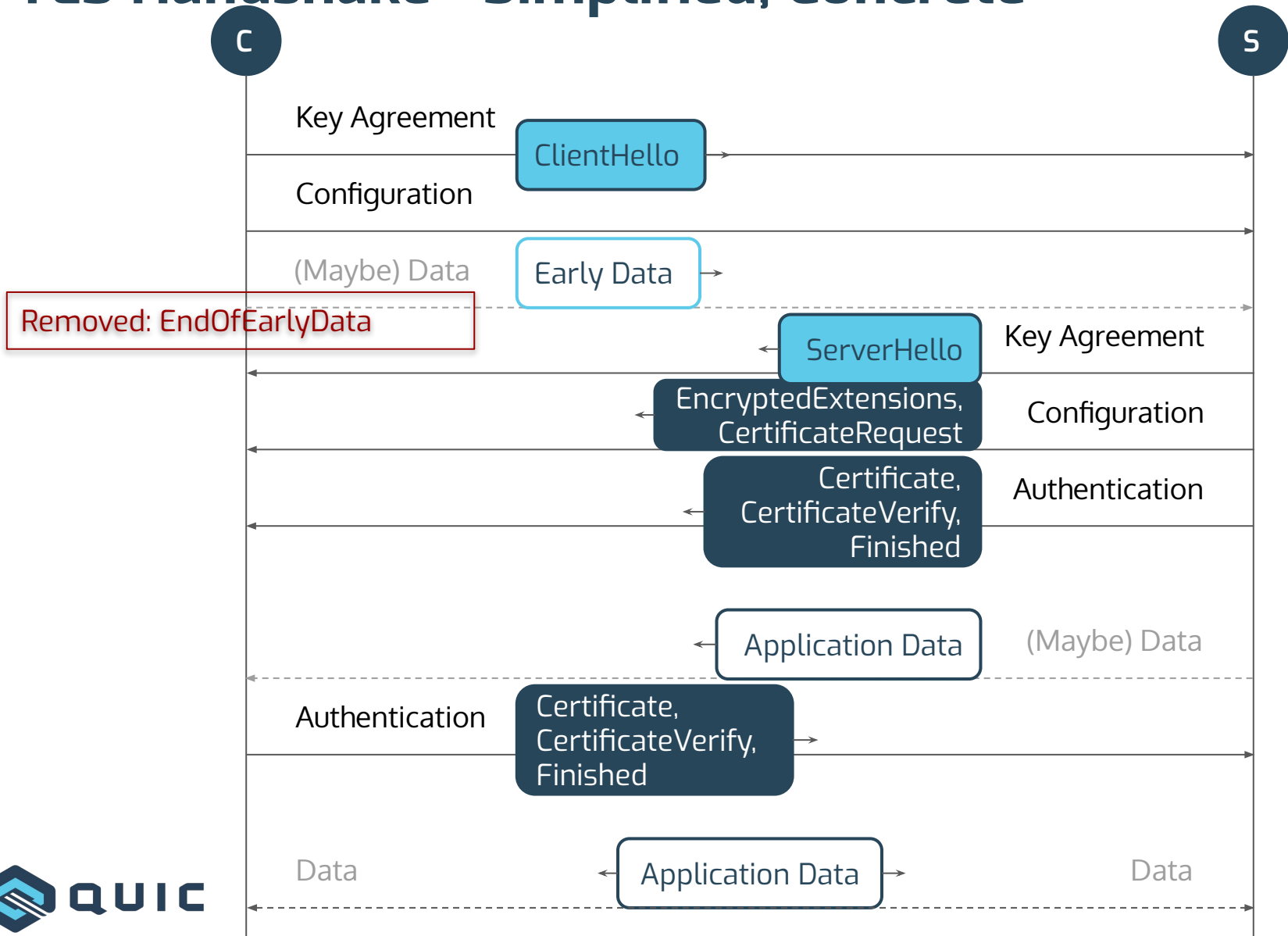
Client sends 0-RTT if they have a pre-shared key

Server send at 0.5-RTT if they don't need client auth'n

TLS Handshake - Simplified, Abstract



TLS Handshake - Simplified, Concrete



QUIC Handshake

Take TLS messages by type of key



Give handshake messages a packet type each

Initial

Handshake

Treat TLS messages as raw bytes

Put those bytes in CRYPTO frames

QUIC uses the secrets generated by TLS to protect packets

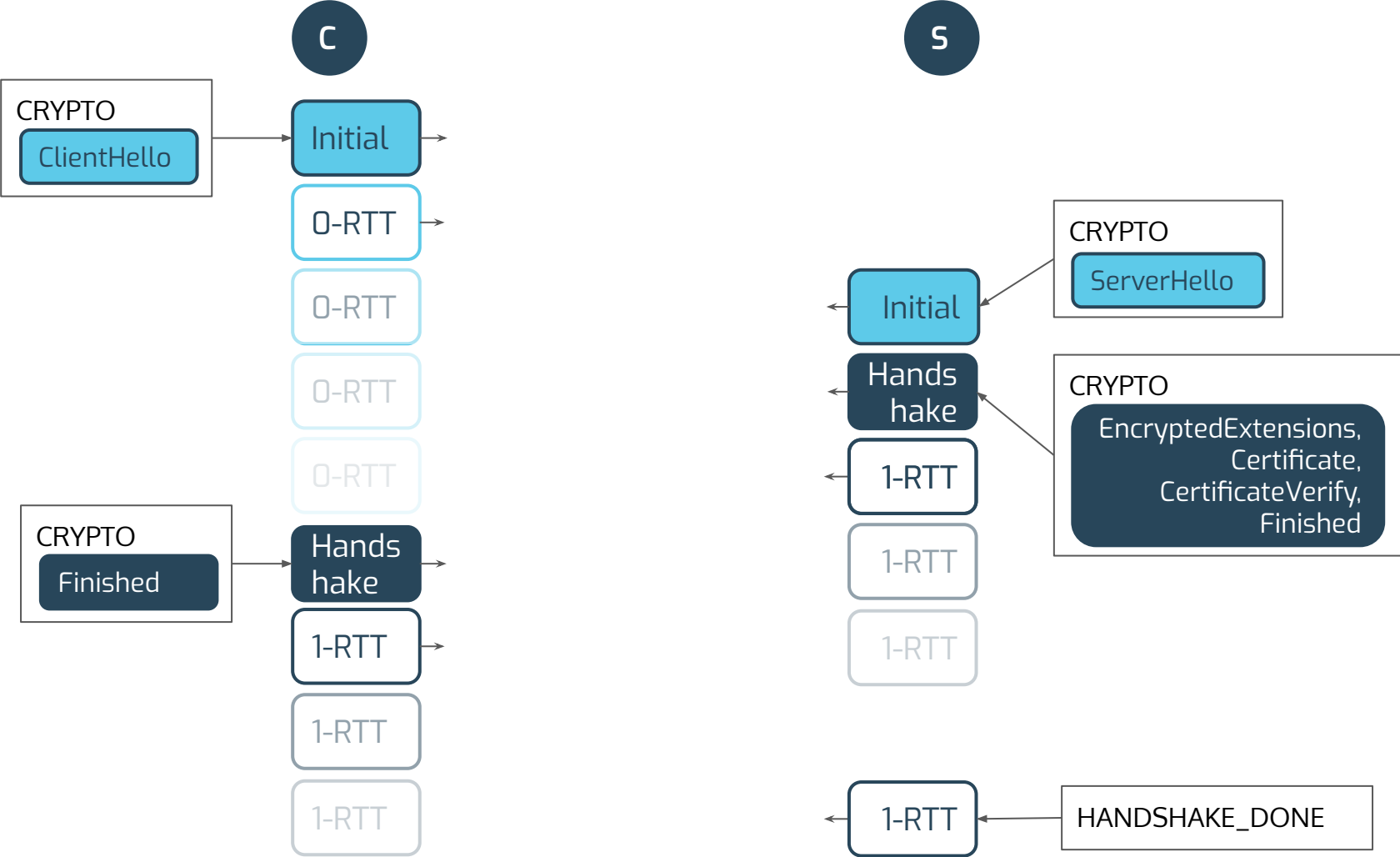
Initial

Handshake

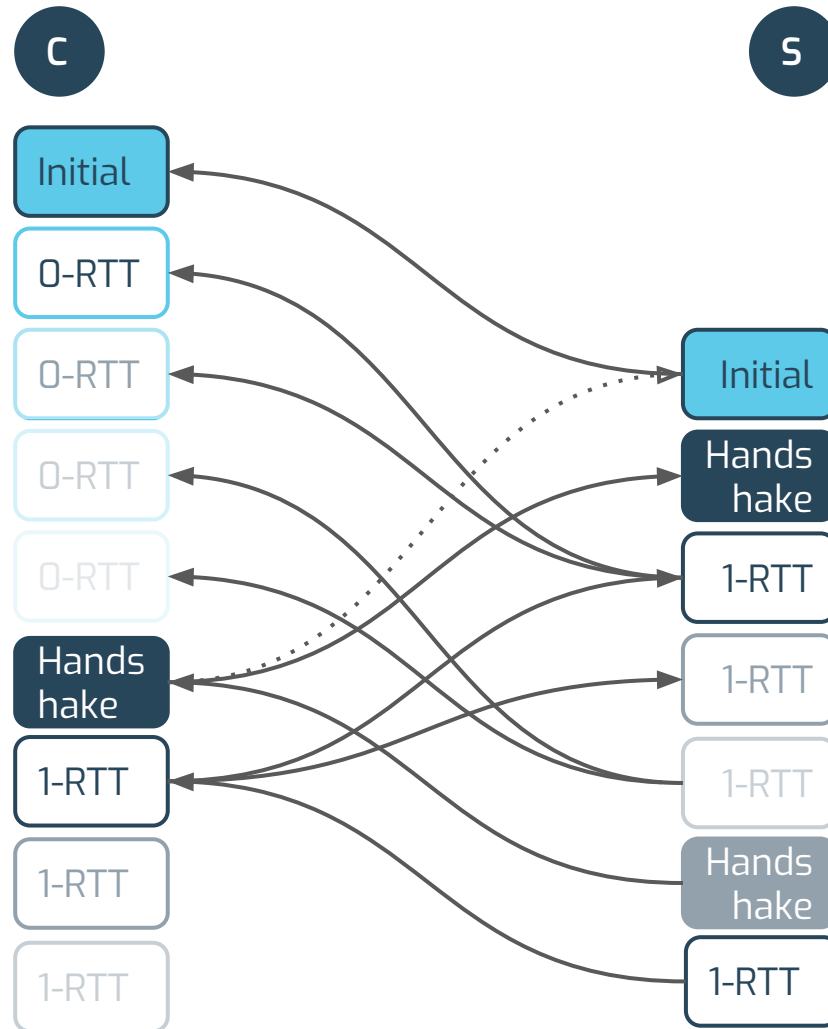
0-RTT

1-RTT (Short)

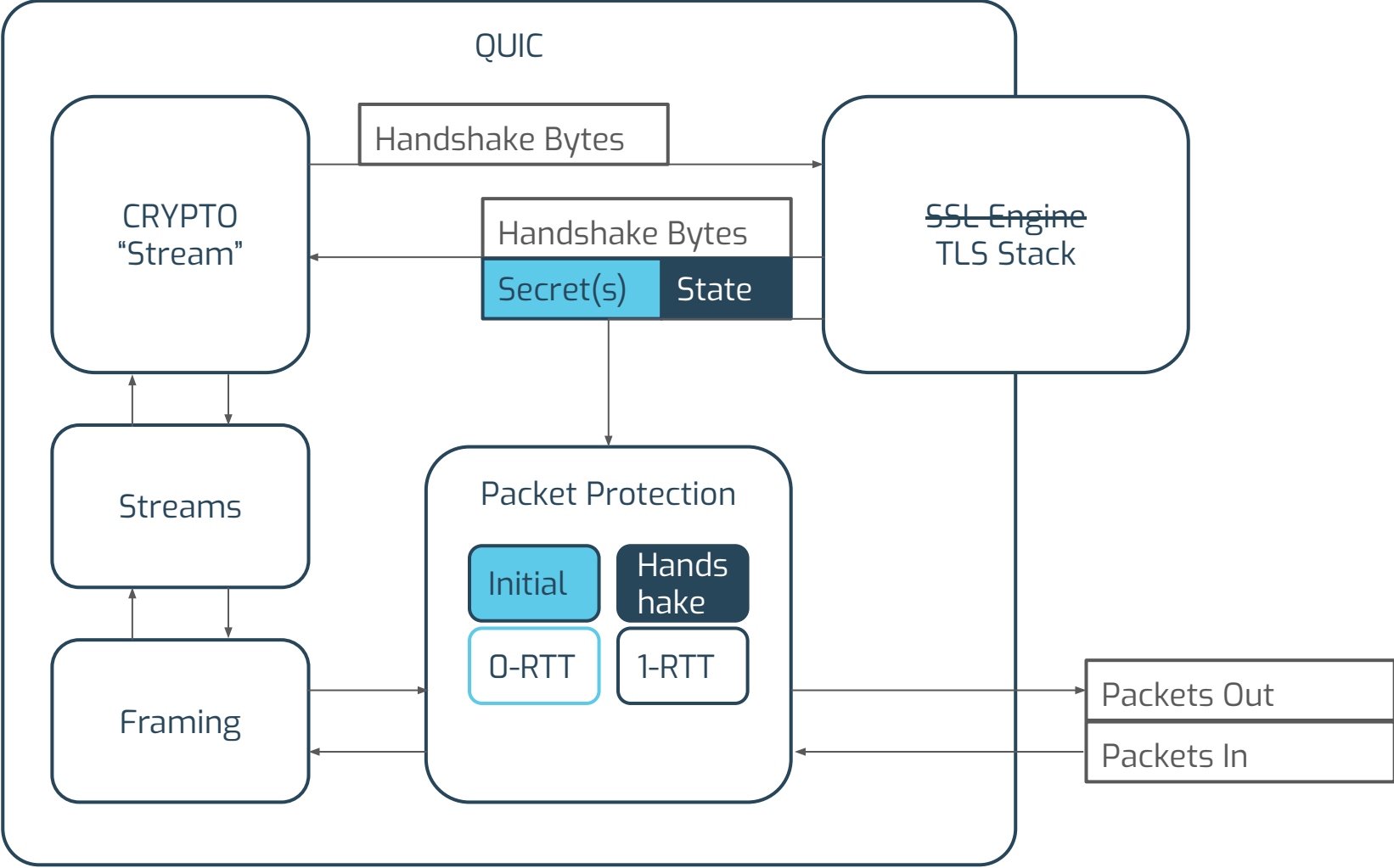
QUIC Handshake - Simplified



QUIC Handshake - With ACKs



“Layering”



DoS mitigation

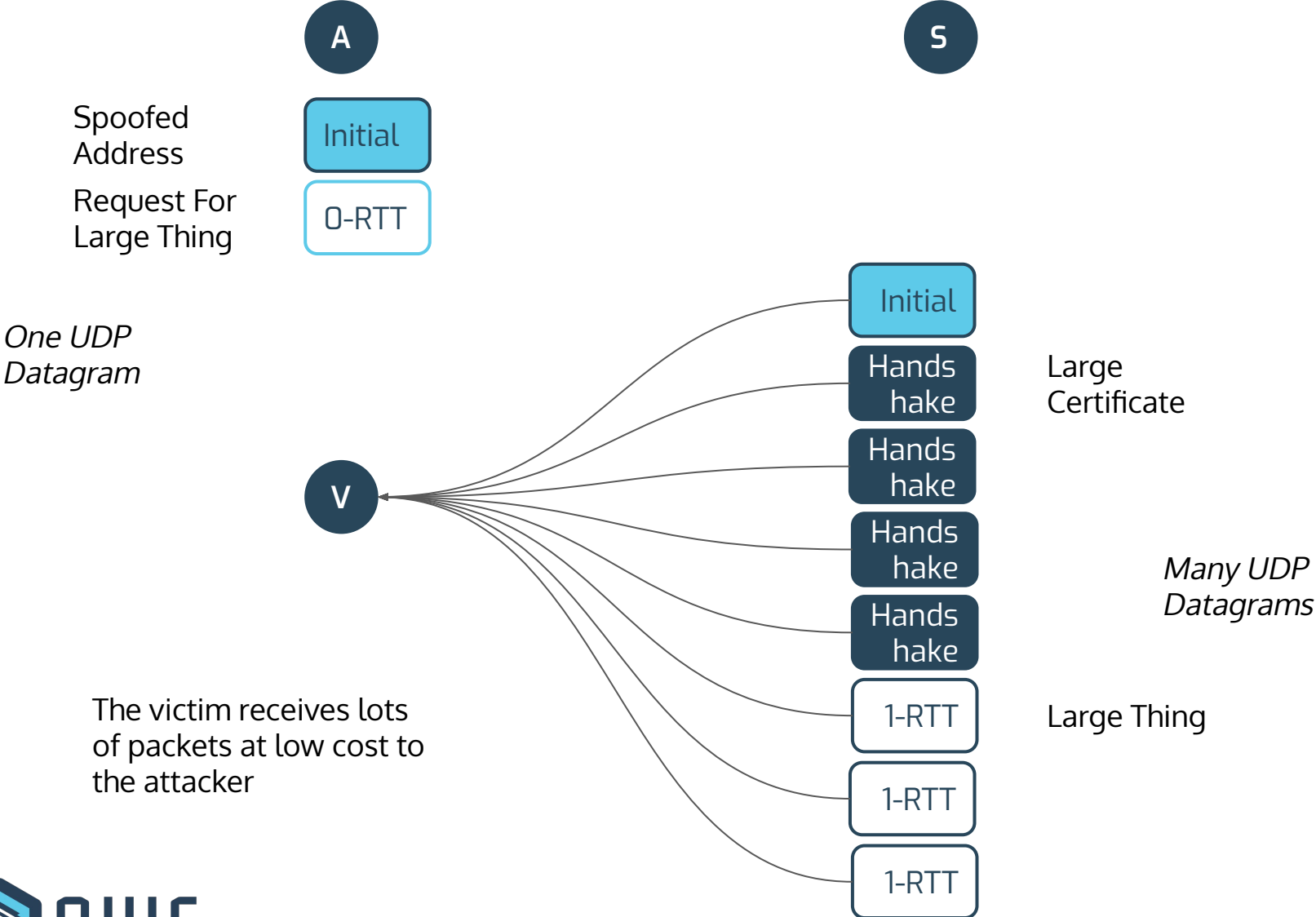
Basic rule:

Don't send more than 3x what you get
from any unvalidated address

Address validation is sending something to an address
and getting a response that proves that they got it

Address validation establishes that the peer
understands what is going on (i.e., QUIC) and
consents to be part of that

Handshake Amplification Attack



TCP Handshake

TCP also has a 3 way handshake: SYN, SYN+ACK, ACK

For TLS/TCP, this adds 1 RT to setup

TCP confirms willingness to communicate

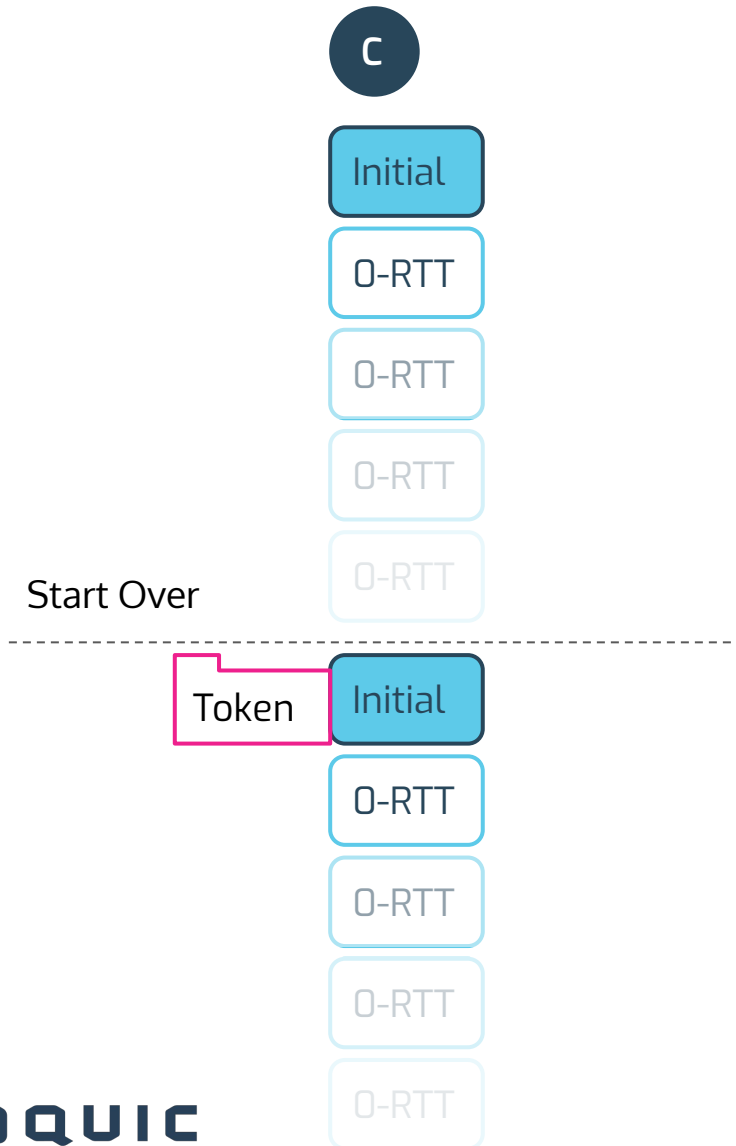
... importantly: before spending CPU on crypto

The basic QUIC handshake does not

so we added Retry

Retry adds a round trip for address validation

QUIC Handshake - Retry



Echoing the token forces the client to prove that it can receive packets from the server.

Communication Consent w/o Retry

Retry is best for extreme cases

- Server is under stress and this might be an attack

- Server thinks the source IP (or port) is dodgy

A round trip to confirm that a client is live is expensive

In other cases, packet protection is used to the same effect

Client Proves that it saw Server Initial

Once all Initial packets are exchanged
endpoints have shared keys
derived from unpredictable key exchange messages

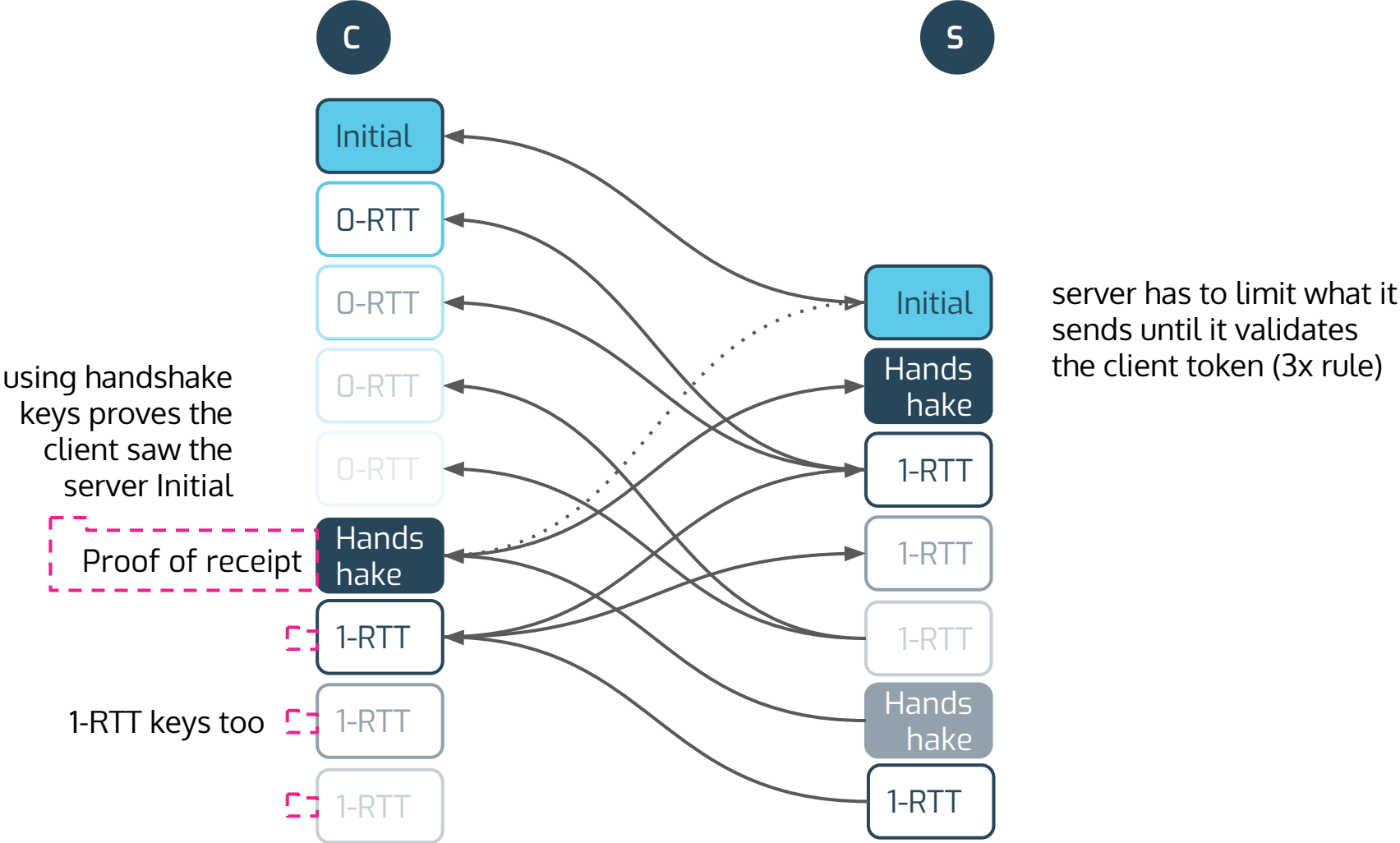
These keys are used to protect Handshake packets

So if the client produces a valid Handshake packet
(or a token that was sent in a Retry)
the server can be sure that the client has seen its packets

Until then, the server has to limit the data it sends

Rule: no more than 3 bytes out for every byte received

Client Implicit Token



Server Proves that it saw Client Initial

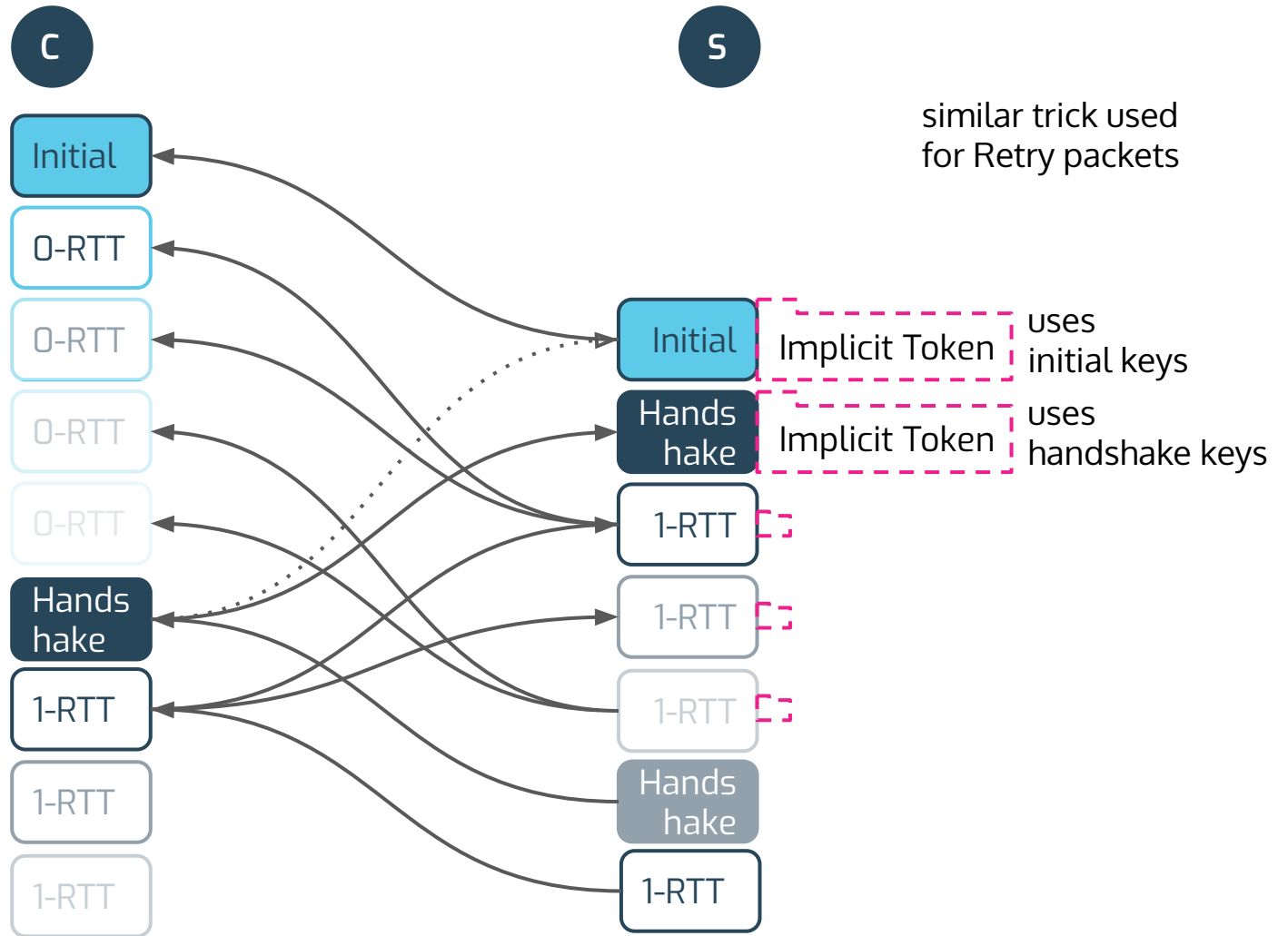
The client encrypts their Initial packet

The key is derived from an unpredictable value
the "Destination Connection ID"

The server proves that it saw this by
using the same value to encrypt its Initial packet

Retry includes the value in its integrity check to prove receipt
the value is not transmitted

Server Implicit Token



Lessons Learned

Getting this far was **hard**

Lots of deadlocks discovered in the process

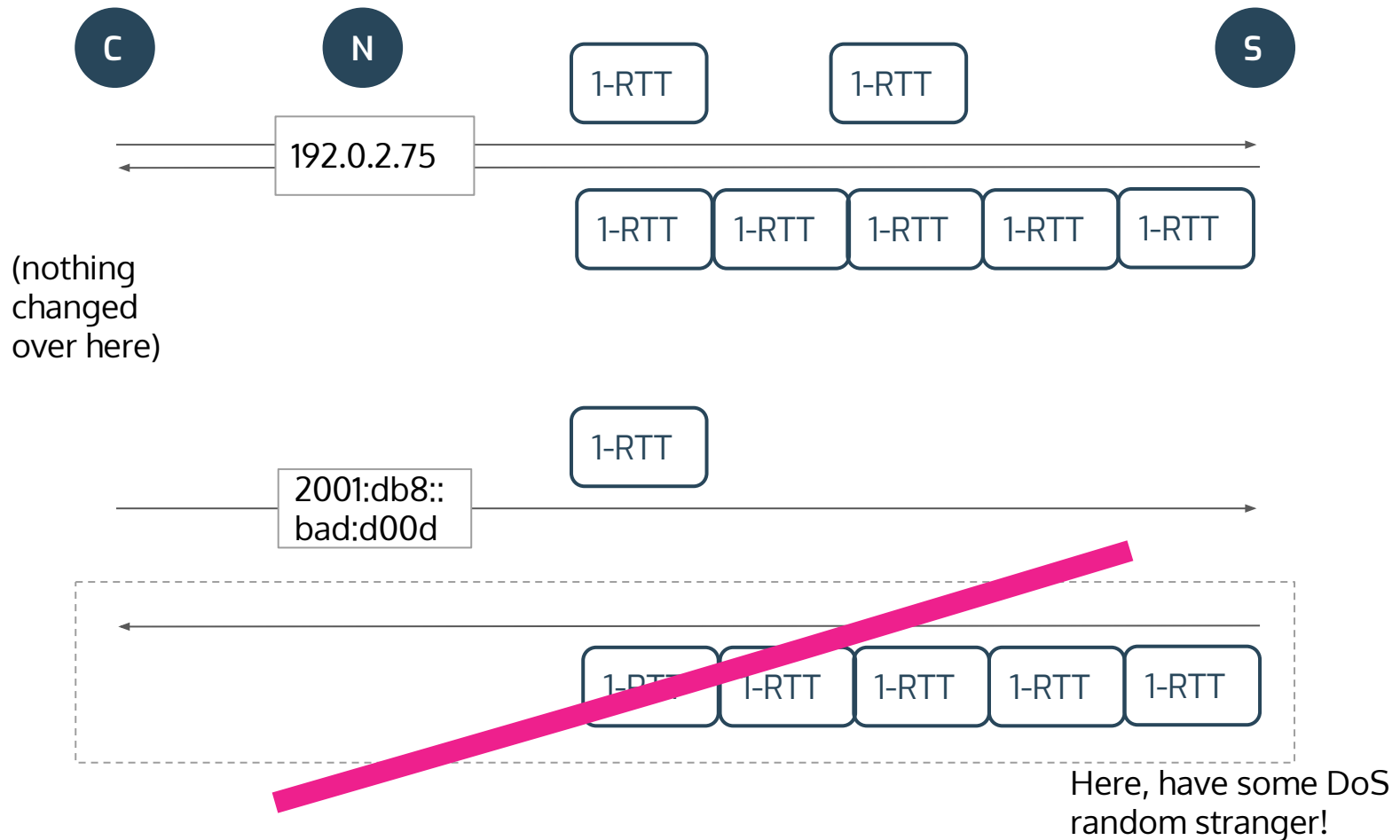
The protocol state machines are complex

No formal verification to support correctness

Version negotiation added more complexity

initially not confident in it either functionality or security
deferred decision to add downgrade prevention

Migration



Migration

Desire to deal with NAT binding changes

Desire to allow moving connection to new paths

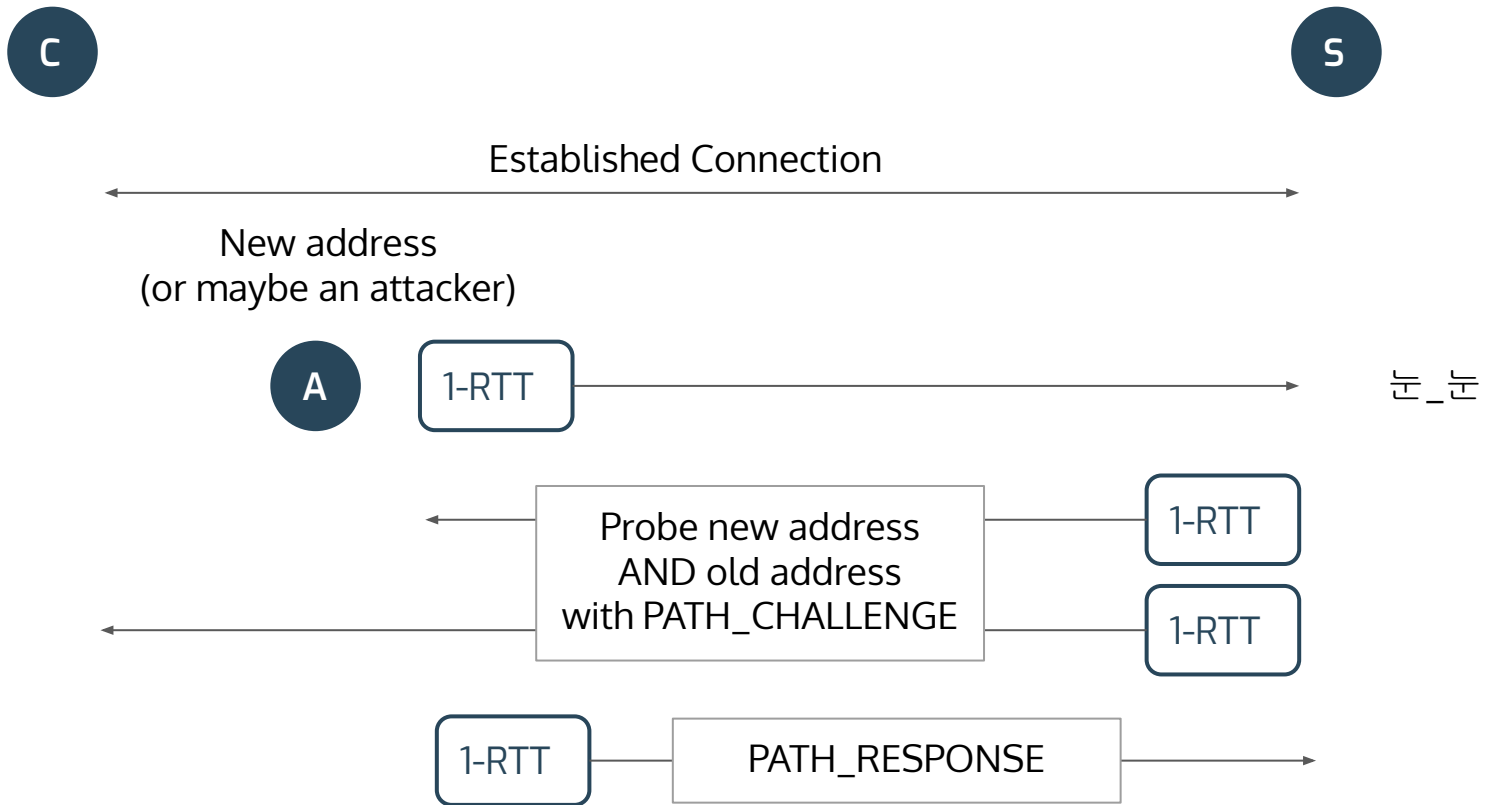
Desire to prevent attacker from forcing unwanted migration

Desire to prevent attacker from blocking migration

Dolev-Yao model says attacker can rewrite addresses

We can't integrity protect or QUIC wouldn't deploy (NAT!)

Address Validation for Migration



Migration Rules

Only clients can migrate

(Servers can ask clients to migrate, once)

To migrate:

Send to the address that sent the last valid packet

3x rule for DoS:

Don't send more than 3x what you receive

...until the path is validated

Migration Design

Anti-amplification applies to any new remote address
new paths require validation to remove limits
PATH_CHALLENGE to test; PATH_RESPONSE to confirm

Also validate old path
in case old path is good and attacker is forwarding

Limit deliberate migrations to client only
special case: server can request migration
to a preferred address after handshake completion

New paths need new connection IDs to avoid linkability

Lots of Other Stuff

QUIC has a bunch of other security-relevant design features

- Packet and packet header protection

- Key rotation

- Stateless reset (secure equivalent of TCP RST)

- Version negotiation (draft-ietf-quic-version-negotiation)

Not enough time to cover everything

QUIC Handshake Performance Degradation

- +1 RTT if version negotiation if the QUIC version is wrong*
- +1 RTT if the server wants to validate the client with Retry
- +1 RTT for TLS HelloRetryRequest for a new client key share
- +n RTTs if data needs to exceed anti-amplification limits (PQ)
 - if client data exceeds initial congestion limit
 - if server handshake exceeds 3x client data

Servers do the first two statelessly

Packet Protection

Everything in QUIC is protected, except

- Version Negotiation - has to be version independent

- Retry - integrity protection only, with fixed keys

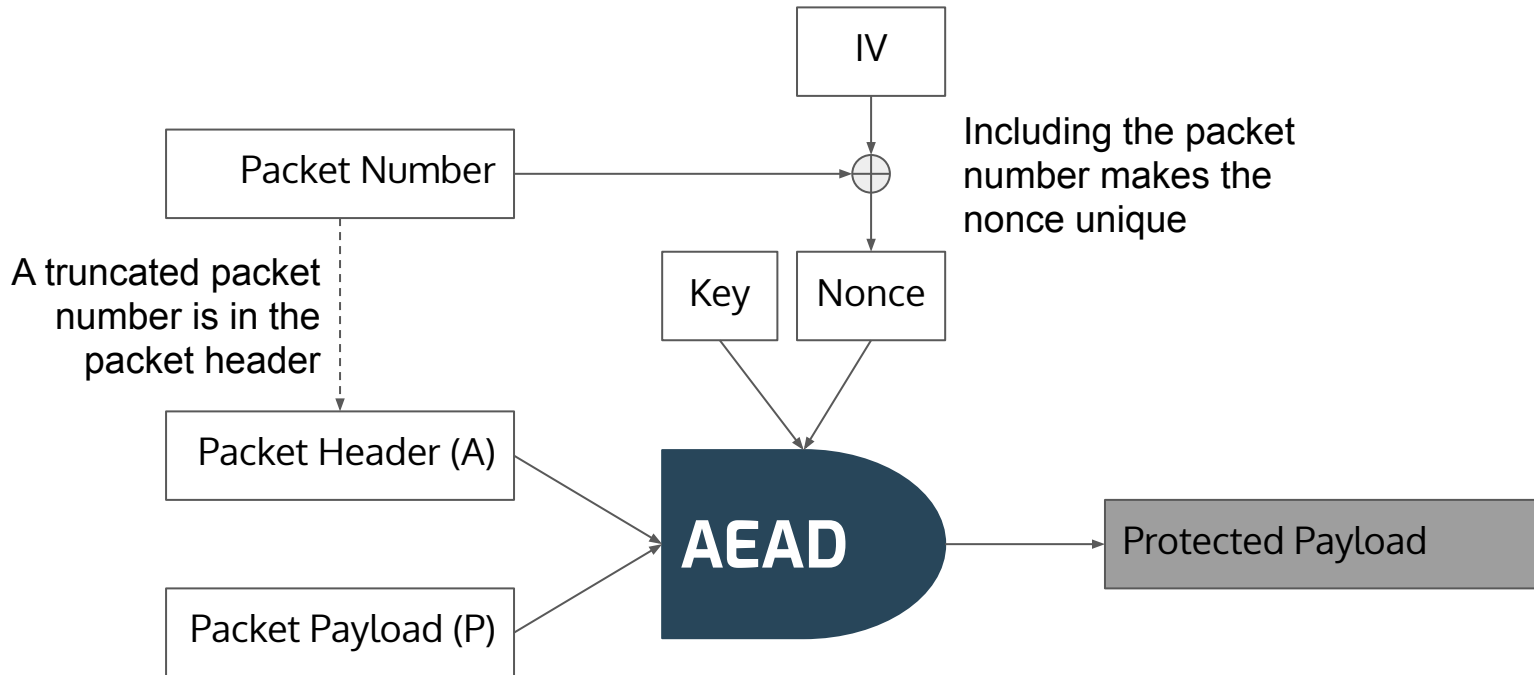
- Initial - protected, but with fixed keys

- 0-RTT - protected, but without replay protection

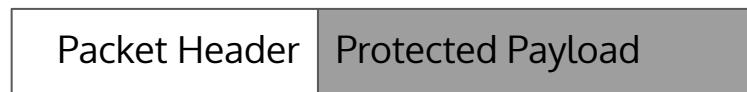
- Handshake - protected, but not fully authenticated

Protected packets all use the same basic scheme

QUIC Packet Protection



We could just send this now...



... but we're not done yet...

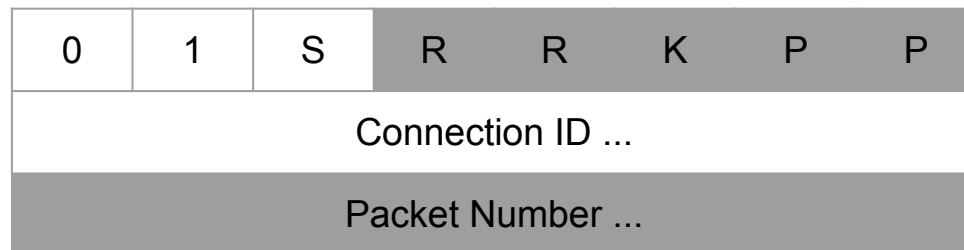
Header Linkability

The QUIC packet header contains linkable fields

connection ID

key phase

packet number



If packets from the same connection are sent on two network paths, these could be used to match those packets

These could allow people to be tracked as they move

Header Protection Lessons

For linkability with Connection IDs we use new values

We tried that for packet numbers, but it didn't work well

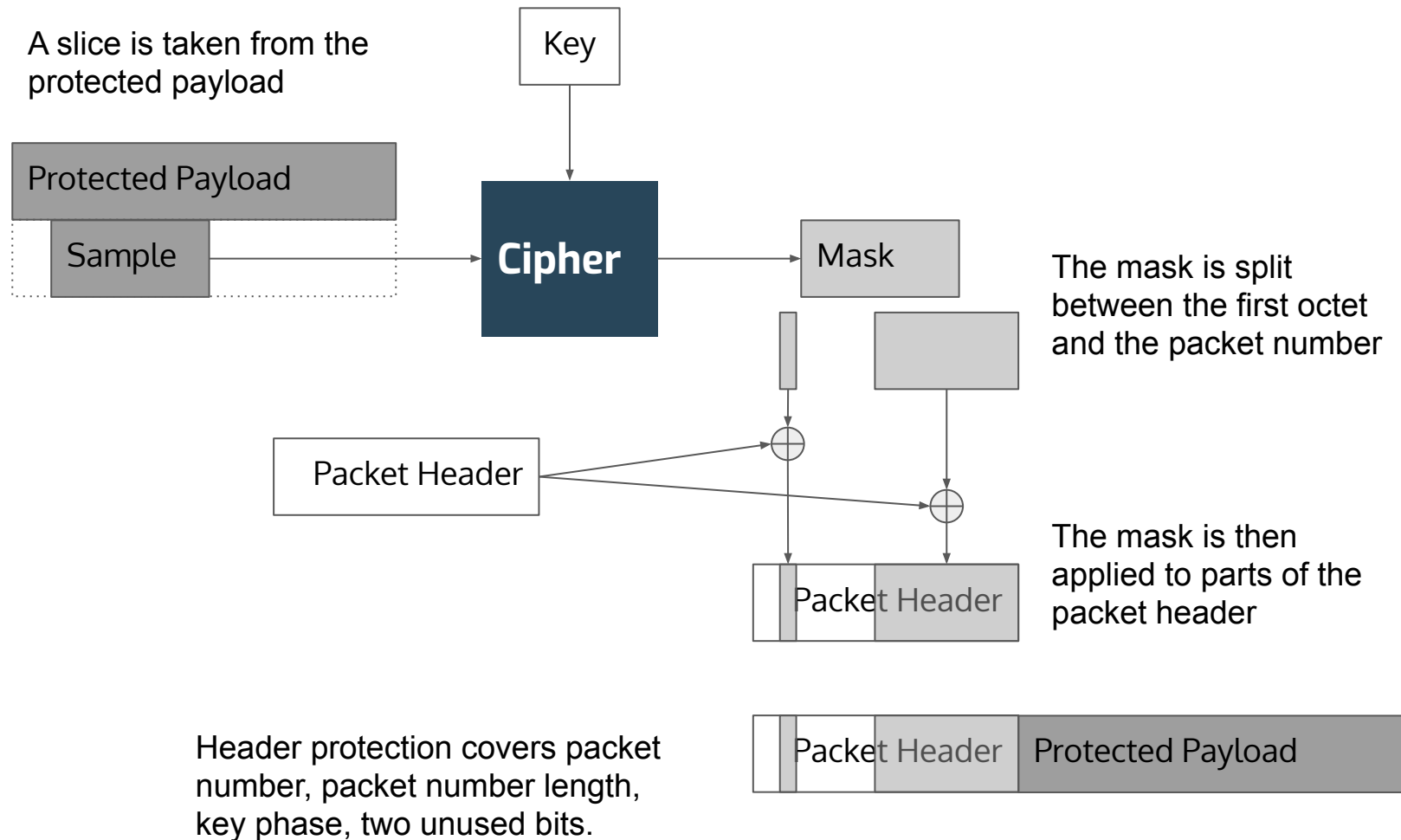
it always ended up looking like bad encryption

Idea: the protected packet payload is essentially random

use that as a nonce and just encrypt the packet number

now we just cite Nonces are Noticed

QUIC Header Protection



Cleartext Parts of QUIC Packets

The first two bits (0b11 = long, 0b01 = short)

Version, Type, and Length (long header)

Destination Connection ID

Source Connection ID and connection ID lengths (long)

Spin bit (short)

Address validation token and its length (Initial packets)

All of Retry and Version Negotiation packets

Key Update

Lock-step protocol for updating keys

- 1-bit signal indicates which keys are used
- updates blocked until peer confirms update

Keying based on TLS design:

- separate read and write secrets are run through HKDF
- new AEAD key and nonce derived from each
- discard old secrets

Header protection keys aren't updated

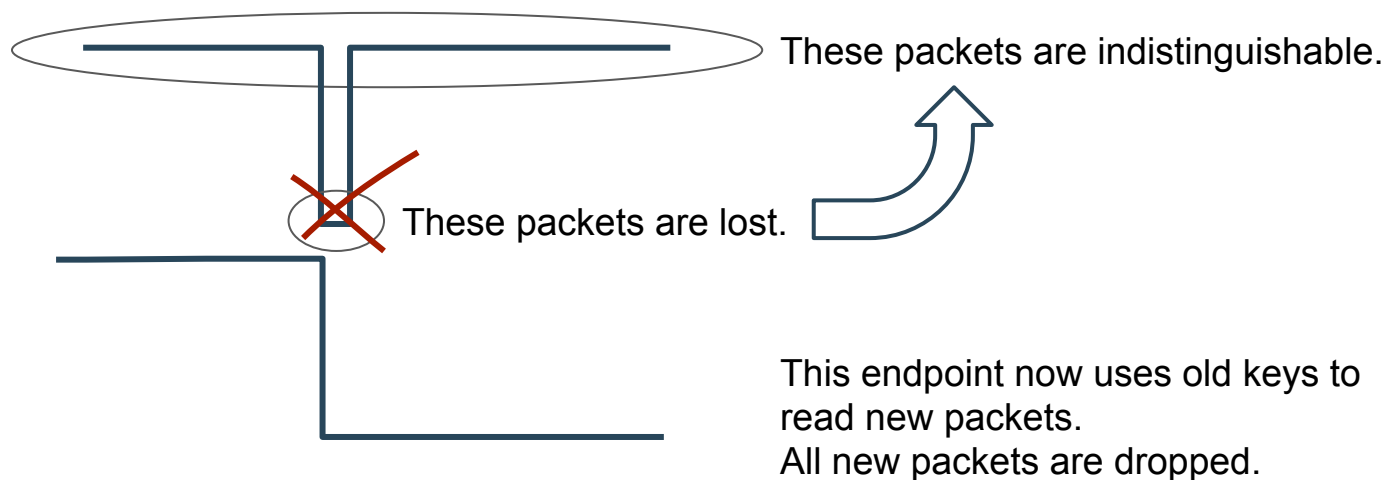
- these are used to protect the 1-bit signal

Key Update Deadlocks

Original protocol had deadlocks

Either side can update at any time.

packets from one endpoint are unreadable



Fix: require acknowledgment of an update, not use of keys

Terminating Connections Safely

Use case: server reboots and loses state
connections will eventually time out
but that isn't efficient

TCP provides RST for this case

but TCP RST can be sent by any on-path element
that is a denial of service risk

QUIC provides a secure stateless reset

QUIC Stateless Reset

Every connection ID has an associated stateless reset token

This token is shared secretly (a frame in a protected packet)

Sending a packet containing this secret kills the connection

These can't be generated by path elements

- they don't know the secret

These can't be detected by path elements

- the packet looks like a regular packet, mostly

Generating a Stateless Reset

How does a server generate a stateless reset?

it might have just rebooted and lost connection state

Keep a secret that is semi-permanent

all server instances have the same secret

The stateless reset token is $KDF(\text{secret}, \text{connection ID})$

The connection ID is taken from the incoming packet

Need to ensure that connection IDs aren't reused

Stateless Reset

C

Initial

S

Server is configured with a static secret.

Initial

Handshake

Stateless reset token is sent to the client under encryption:
 $\text{token} = f(\text{secret}, \text{cid})$

Server Reboots

1-RTT

Server doesn't know about this connection!

Reset

Recalculate:
 $\text{token} = f(\text{secret}, \text{cid})$

The packet looks normal but can't be decrypted.
However it contains the token.
The client abandons the connection.

Stateless Reset Challenges

Stateless reset oracles in clusters

- misrouting might cause node to reset live connections
- creates coupling between routing logic and node config

Key rotation challenges

- limited space to embed information in connection ID

Stateless resets aren't truly indistinguishable from normal

- can only really create uncertainty for middleboxes

Version Negotiation (tentative)

Two modes compatible and incompatible

Compatible can be performed with no additional round trips

Client sends Initial[vX] containing a signal that vY is OK

Server knows **F : Initial[vX] → Initial[vY]**

Server completes handshake with vY

Incompatible requires a rejection using Version Negotiation packet

Client might start over with a different version

Version Negotiation Properties

Server picks between compatible versions

Client picks between incompatible versions

Authenticate final choice by describing choices and outcomes in TLS handshake and authenticating that

Other Potential Sources

HTTP/3 compression is interesting

Flow control deadlocks might have security implications

Effect of network-controlled signals on internal state

- path MTU discovery

- packet loss

- ECN-CE marking