Introduction into QUIC Protocol

Semester Project at DCSL, supervised by Marios Kogias
Sutandi, I Made Sanadhi

Making the internet faster with...

Quick
UDP
Internet
Connections

Project Goals

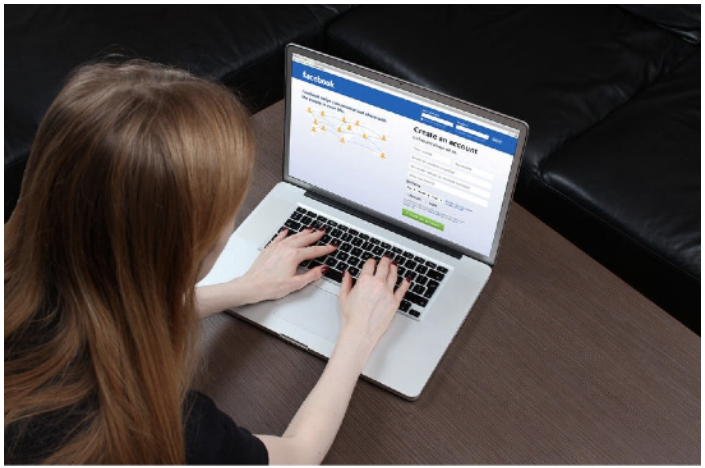
- Insight of QUIC
- Overview for QUIC
- Semantics of QUIC
- Structure of QUIC
- Benefits of QUIC

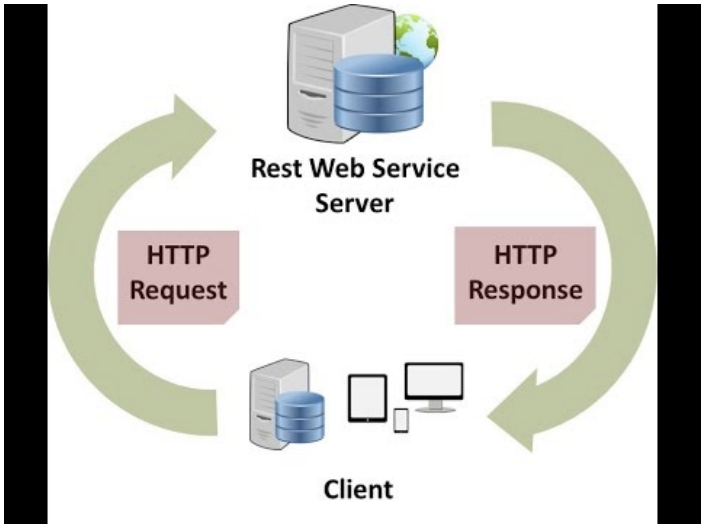
Contents

- Motivation for QUIC
- QUIC Overview
- QUIC vs TCP
- QUIC Structures/Features
- QUIC API
- QUIC Packet Exchanges
- QUIC Chromium vs QUIC IETF
- Conclusions

Current Situation

Most application desire a reliable delivery -> TCP

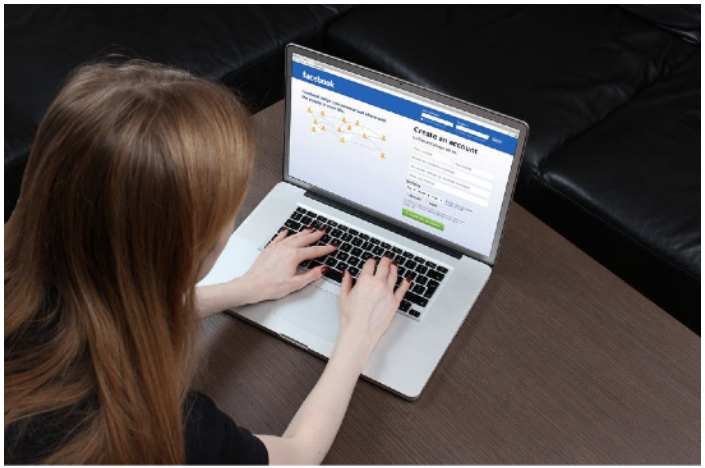


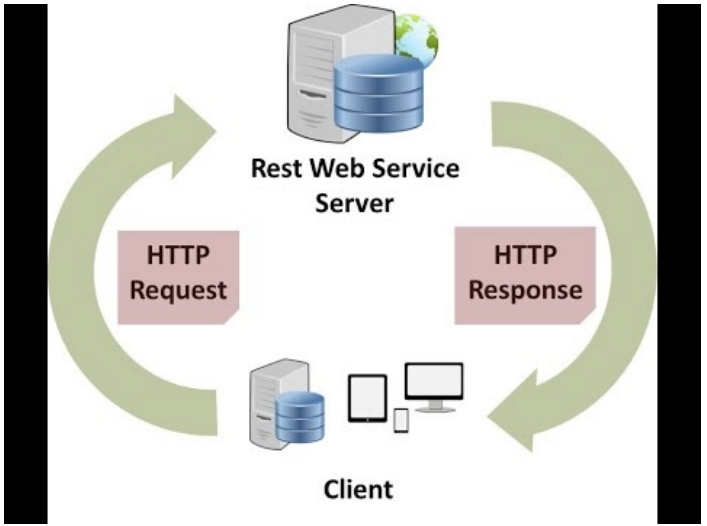


Web application (HTTP communication) is atop of TCP!

Current Situation

Most application desire a reliable delivery -> TCP



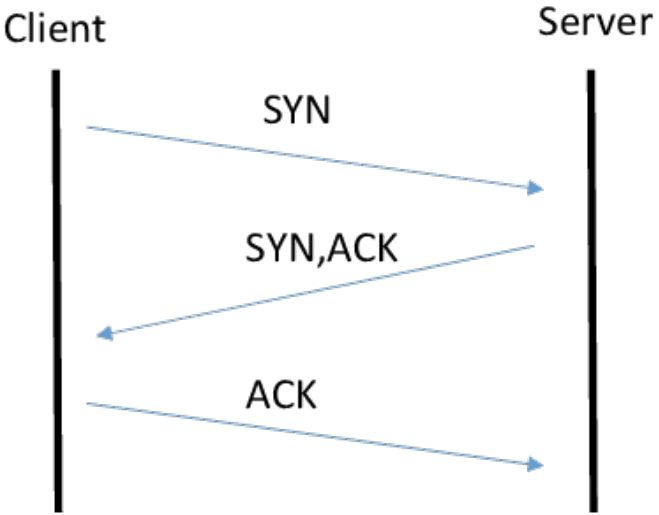


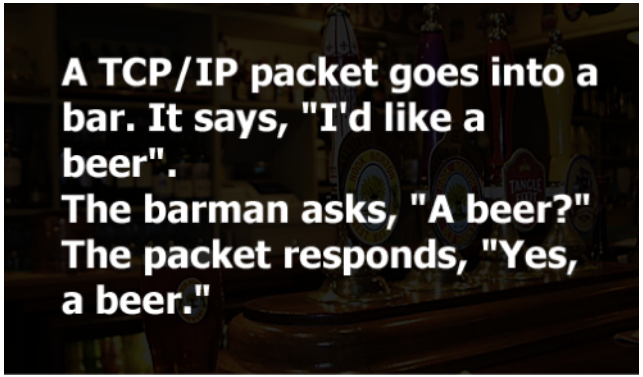
Web application (HTTP communication) is atop of TCP!

Shortcomings…?

TCP's Connection Establishment

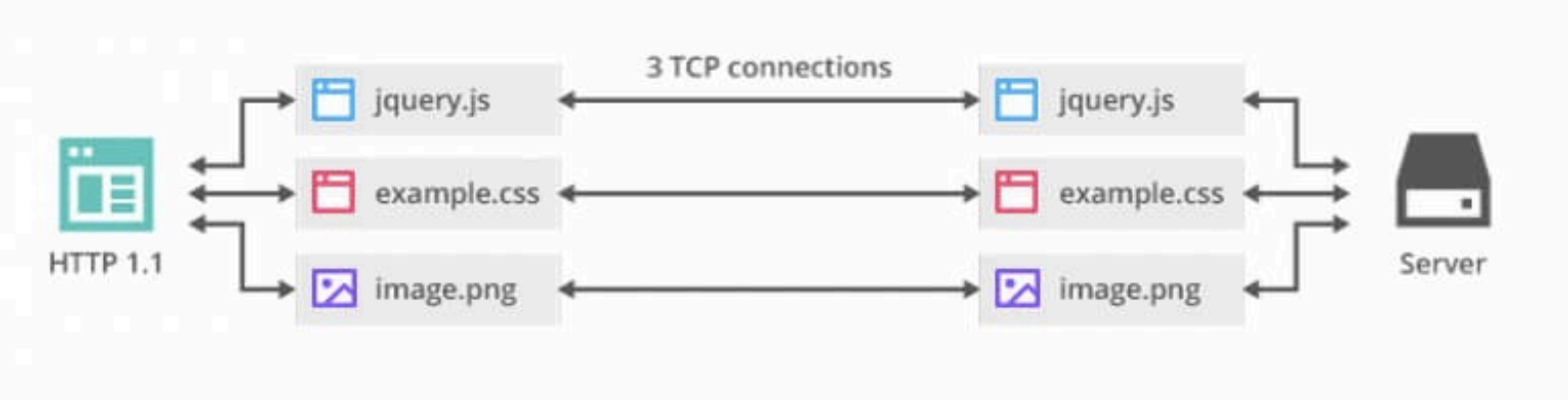
Requires at least 1-RTT!





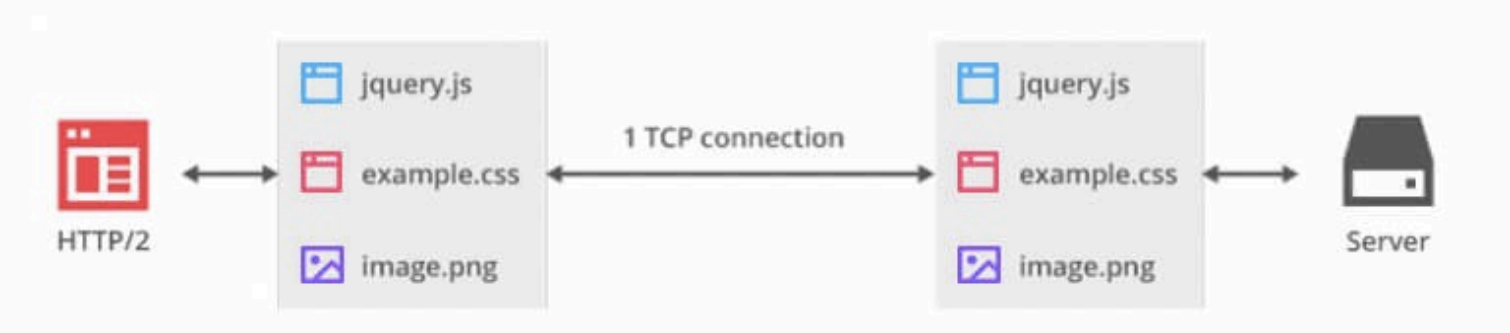
TCP in HTTP 1

One connection per object



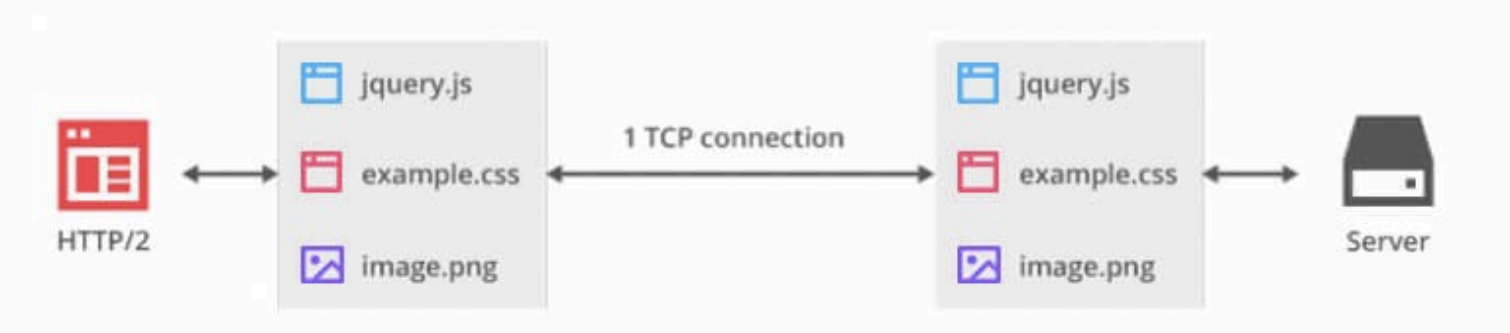
TCP in HTTP 2

One connection for multiple objects



TCP in HTTP 2

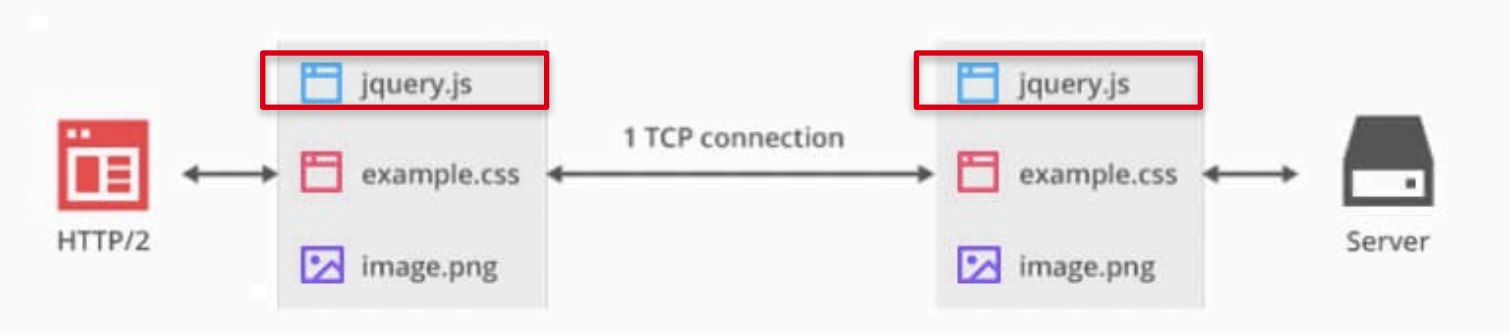
One connection for multiple objects



The catch?

TCP in HTTP 2

One connection for multiple objects



The catch?

Head-of-Line Blocking

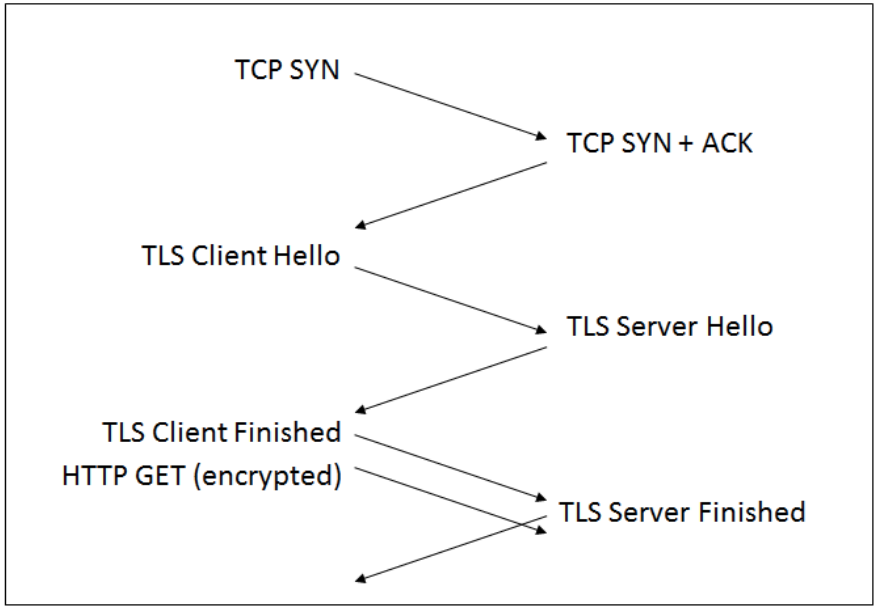
TCP for Secure Communication

- Applications need to add TLS (security layer) atop of TCP

TCP for Secure Communication

- Applications need to add TLS (security layer) atop of TCP

At least another 1-RTT for connection establishment (e.g. TCP fast open)



QUIC

Motivation for QUIC

Handshake delay

- TCP + TLS: 2 RTT
- QUIC: 0 RTT (best case)

Motivation for QUIC

Handshake delay

- TCP + TLS: 2 RTT
- QUIC: 0 RTT (best case)

Head-of-Line Blocking

- HOL delay problem is still happening in HTTP/2 over TCP

Motivation for QUIC

Handshake delay
- TCP + TLS: 2 RTT
- QUIC: 0 RTT (best case)

Head-of-Line Blocking
- HOL delay problem is still happening in HTTP/2 over TCP

Security:
- Demands for a fully encrypted, thus secured internet traffic

Motivation for QUIC

Handshake delay
- TCP + TLS: 2 RTT
- QUIC: 0 RTT (best case)

Head-of-Line Blocking
- HOL delay problem is still happening in HTTP/2 over TCP

Security:
- Demands for a fully encrypted, thus secured internet traffic

As reliable as TCP, as secure as TLS, and atop of UDP

Motivation for QUIC

Protocol Entrenchment

- Deliver a new transport protocol without major upgrades on middleboxes.

Motivation for QUIC

Protocol Entrenchment

- Deliver a new transport protocol without major upgrades on middleboxes.

Implementation Entrenchment

- Deploy changes to client rapidly and handle coupling of transport layer protocol with OS kernel.
- Networking protocol in the user space
- Deployment, testing, iteration becomes faster

QUIC Overview

- Designed for HTTP/2 communication

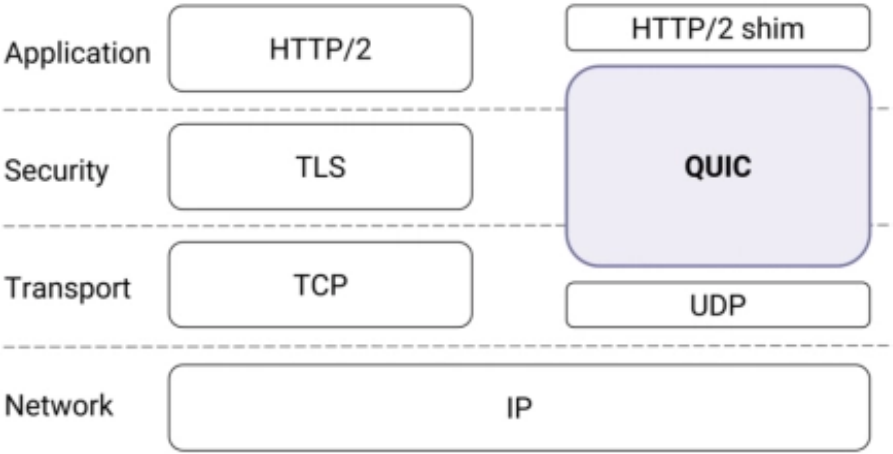


Figure 1: QUIC in the traditional HTTPS stack.

QUIC Overview

- Designed for HTTP/2 communication

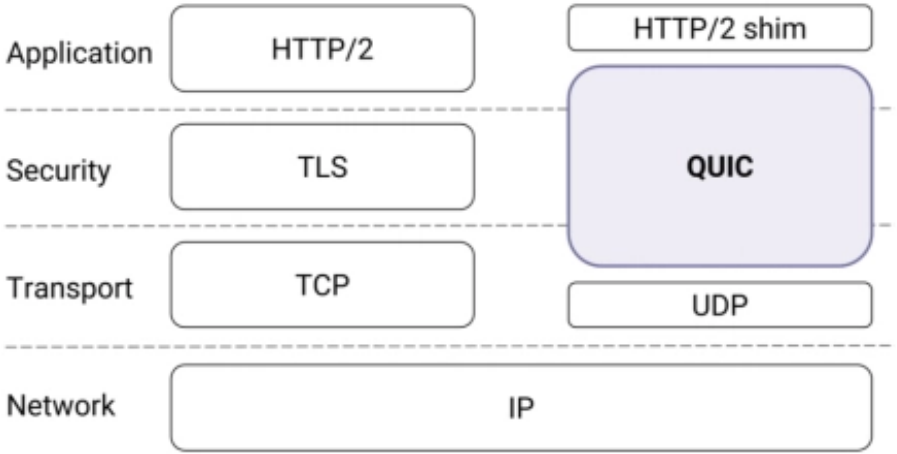


Figure 1: QUIC in the traditional HTTPS stack.

- But also other applications that require reliable communication! E.g video streaming

QUIC Overview

An interesting fact? [1]

- Chrome, Google Search App, and Youtube app make use of QUIC
- Google claims that 7% of the internet traffics are now QUIC.

QUIC Overview

Two version of QUIC:

- QUIC Chromium / GQUIC
- QUIC IETF

**Unless being specified, let's assume that the rest of explanations are about QUIC Chromium / GQUIC*

QUIC's Abstraction

1. Connection (UDP)
2. Streams

QUIC Connection

Implement “pluggable” congestion control
- e.g. Cubic (default), BBR, NewReno

Connection-level flow control.
- Limits the buffer size that receiver must maintain

QUIC: Connection establishment

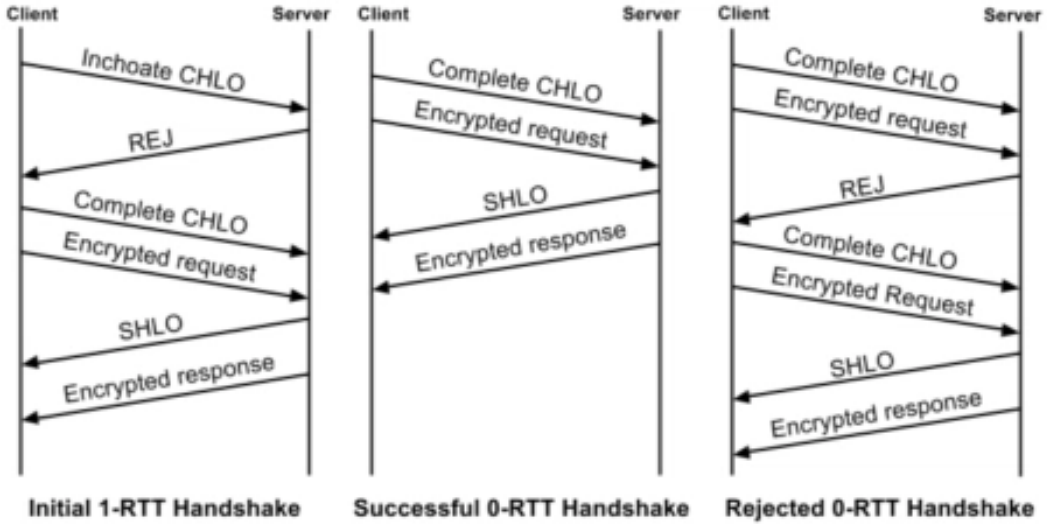


Figure 4: Timeline of QUIC's initial 1-RTT handshake, a subsequent successful 0-RTT handshake, and a failed 0-RTT handshake.

QUIC: Connection establishment

1-RTT for initial connection

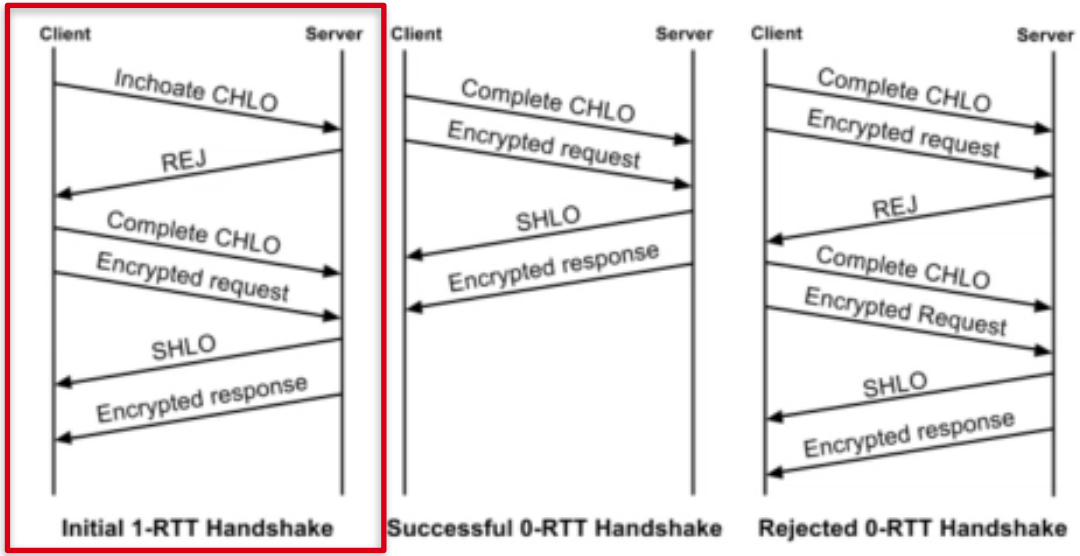


Figure 4: Timeline of QUIC's initial 1-RTT handshake, a subsequent successful 0-RTT handshake, and a failed 0-RTT handshake.

QUIC: Connection establishment

0-RTT for the best case

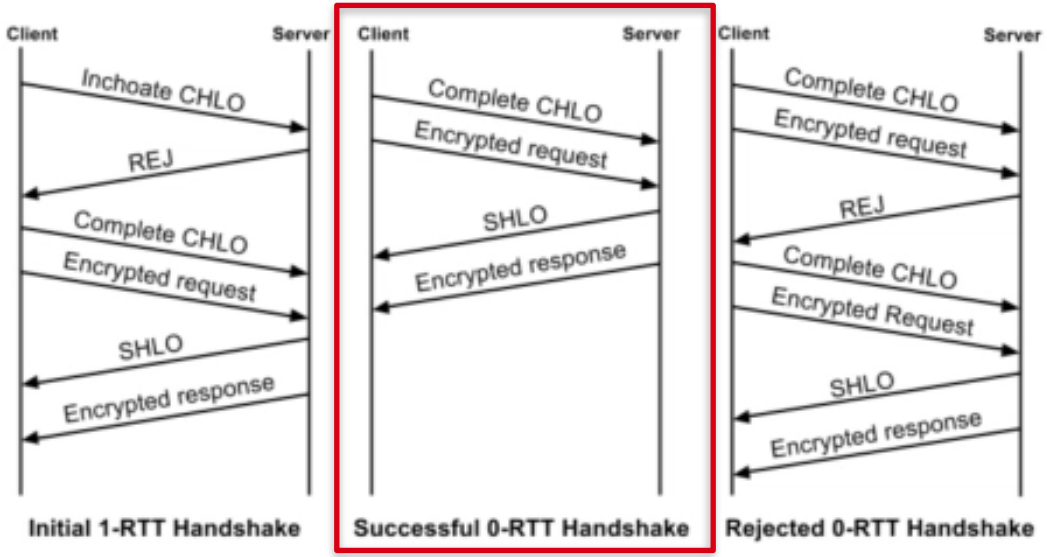


Figure 4: Timeline of QUIC's initial 1-RTT handshake, a subsequent successful 0-RTT handshake, and a failed 0-RTT handshake.

QUIC: Connection establishment

1-RTT for the worst case (rejected)

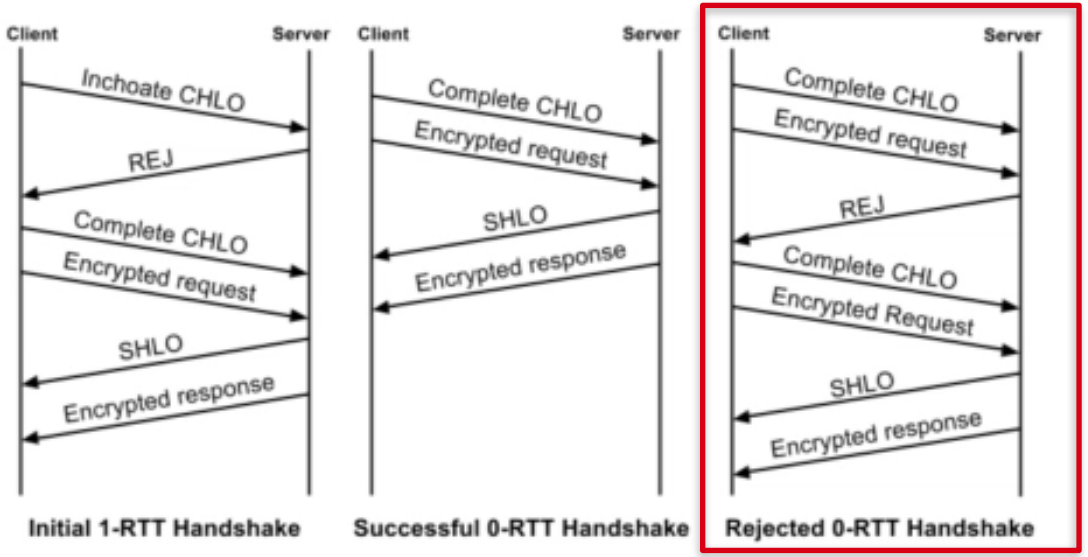


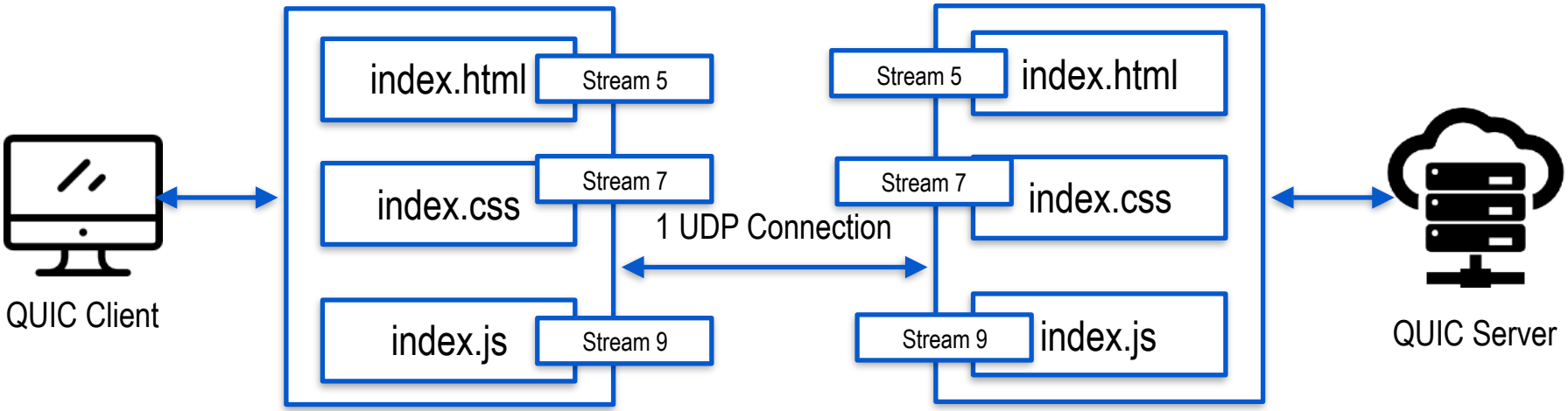
Figure 4: Timeline of QUIC's initial 1-RTT handshake, a subsequent successful 0-RTT handshake, and a failed 0-RTT handshake.

QUIC: Introducing Streams

- QUIC streams are a lightweight abstraction that provide a reliable bidirectional bytestream.
- Streams can be used for framing application messages of arbitrary size.

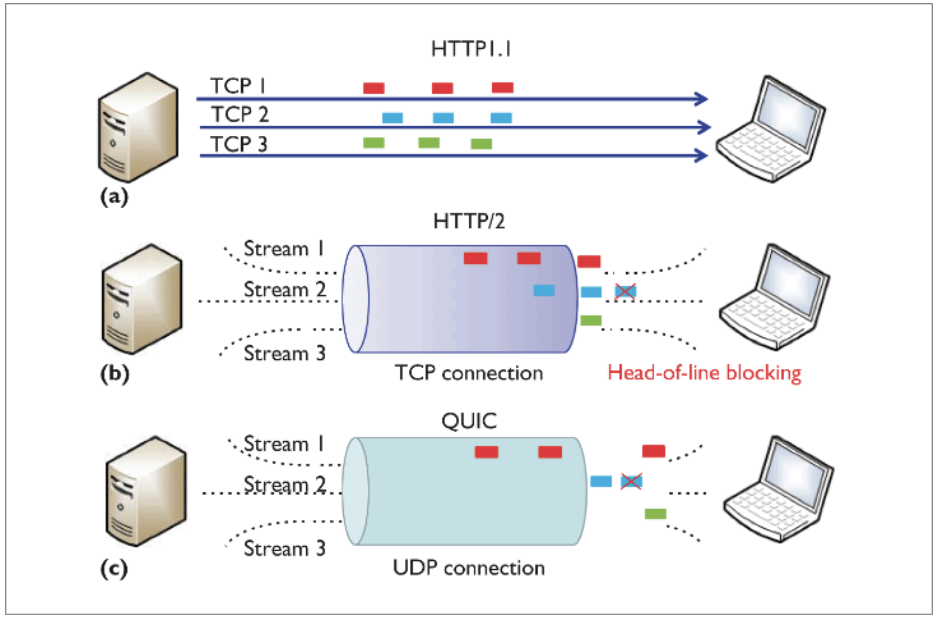
QUIC: Introducing Streams

- QUIC streams are a lightweight abstraction that provide a reliable bidirectional bytestream.
- Streams can be used for framing application messages of arbitrary size.



QUIC: Addressing HOL with Streams

- One QUIC connection contains multiple N streams
- Different objects are mapped into different streams.
- Avoid HOL, which happens in TCP's sequential delivery



QUIC Stream: Structure

Figure 5: Structure of a QUIC packet, as of version 35 of Google's QUIC implementation. Red is the authenticated but unencrypted public header, green indicates the encrypted body. This packet structure is evolving as QUIC gets standardized at the IETF [2].

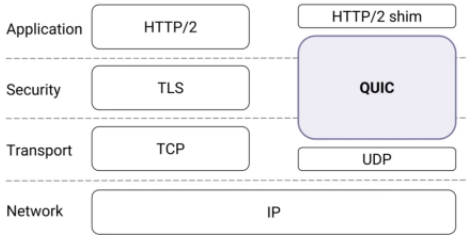


Figure 1: QUIC in the traditional HTTPS stack.

QUIC: Dedicated Stream

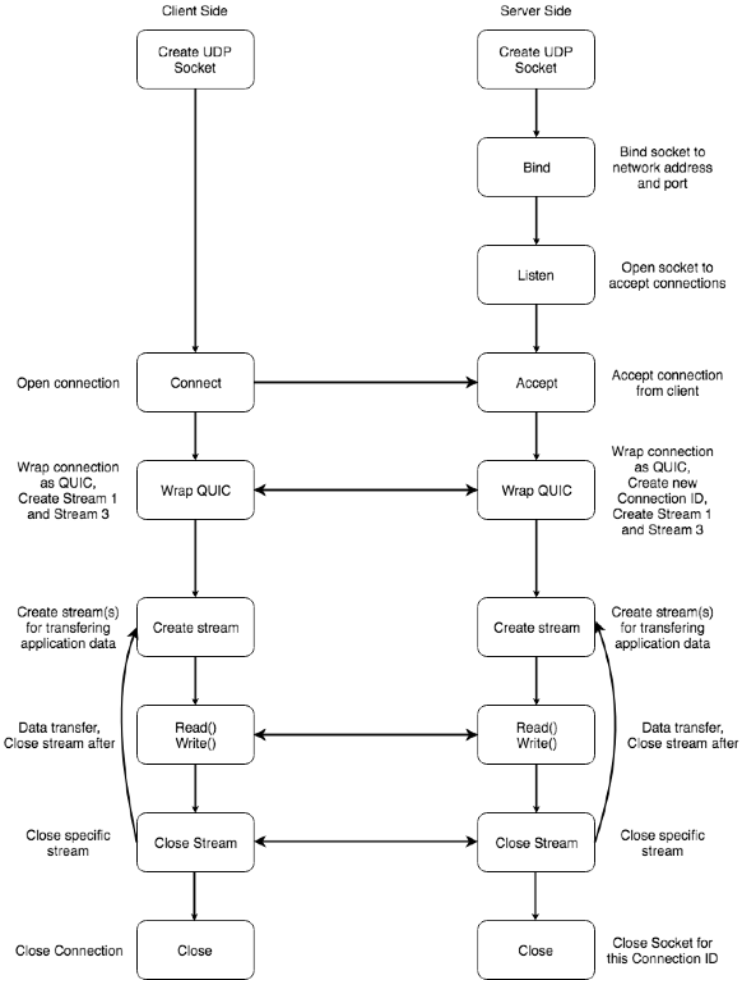
1. Cryptographic handshake: Stream 1 (Google), Stream 0 (IETF)
2. HTTP/2 Headers mapping: Stream 3 (Google only)

QUIC Streams' Flow Control

Stream-level flow control:

- Limits the buffer that a single stream can consume
- No greedy streams

QUIC API


Client Side
Server Side
Create UDP Socket
Create UDP Socket
Bind
Bind socket to network address and port
Listen
Open socket to accept connections
Open connection
Connect
Accept
Accept connection from client
Wrap connection as QUIC, Create Stream 1 and Stream 3
Wrap QUIC
Wrap QUIC
Wrap connection as QUIC, Create new Connection ID, Create Stream 1 and Stream 3
Create stream(s) for transfering application data
Create stream
Create stream
Create stream(s) for transfering application data
Data transfer, Close stream after
Read() Write()
Read() Write()
Data transfer, Close stream after
Close specific stream
Close Stream
Close Stream
Close specific stream
Close Connection
Close
Close
Close Socket for this Connection ID

QUIC: Connection Creation - Server & Client

```
connection_ = new QuicConnection(
    connection_id_, QuicSocketAddress(QuicSocketAddressImpl(peer_addr_)),
    helper_.get(), alarm_factory_.get(),
```

QUIC: Socket Creation - Server

```
int QuicSimpleServer::Listen(const IPEndPoint& address) {
  std::unique_ptr<UDPServerSocket> socket(
      new UDPServerSocket(&net_log_, NetLogSource()));

  socket->AllowAddressReuse();

  int rc = socket->Listen(address);
  if (rc < 0) {
    LOG(ERROR) << "Listen() failed: " << ErrorToString(rc);
    return rc;
  }

  // cont'd
  …
```

QUIC: Streams Creation - Server

Incoming stream:

```
QuicSpdyStream* QuicSimpleServerSession::CreateIncomingDynamicStream(
    QuicStreamId id) {
  if (!ShouldCreateIncomingDynamicStream(id)) {
    return nullptr;
  }

  QuicSpdyStream* stream =
      new QuicSimpleServerStream(id, this, response_cache_);
  ActivateStream(QuicWrapUnique(stream));
  return stream;
}
```

QUIC: Streams Creation - Server

Outgoing stream:

```
QuicSimpleServerStream* QuicSimpleServerSession::CreateOutgoingDynamicStream() {
  if (!ShouldCreateOutgoingDynamicStream()) {
    return nullptr;
  }

  QuicSimpleServerStream* stream = new QuicSimpleServerStream(
      GetNextOutgoingStreamId(), this, response_cache_);
  ActivateStream(QuicWrapUnique(stream));
  return stream;
}
```

QUIC: Streams Creation - Server

Stream validity checker:

```
// We rely on the visitor to check validity of stream_id.
bool valid_stream =
    visitor()->OnUnknownFrame(header.stream_id, raw_frame_type);
```

Stream writer:

```
if (!body.empty() || send_fin) {
  WriteOrBufferData(body, send_fin, nullptr);
}
```

QUIC: Socket Creation - Client

```
bool QuicClientMessageLooplNetworkHelper::CreateUDPSocketAndBind(
    QuicSocketAddress server_address,
    QuicIpAddress bind_to_address,
    int bind_to_port) {
  auto socket = std::make_unique<UDPClientSocket>(DatagramSocket::DEFAULT_BIND,
                                                  &net_log_, NetLogSource());

  // cont'd
  …

}
```

QUIC: Streams Creation - Client

```
void QuicSpdyClientBase::SendRequest(const SpdyHeaderBlock& headers,
                                     QuicStringPiece body,
                                     bool fin) {
  // some code here
  …

  QuicSpdyClientStream* stream = CreateClientStream();
  stream->SendRequest(headers.Clone(), body, fin);

  // Record this in case we need to resend.
  MaybeAddDataToResend(headers, body, fin);
}
```

QUIC: Streams Creation - Client

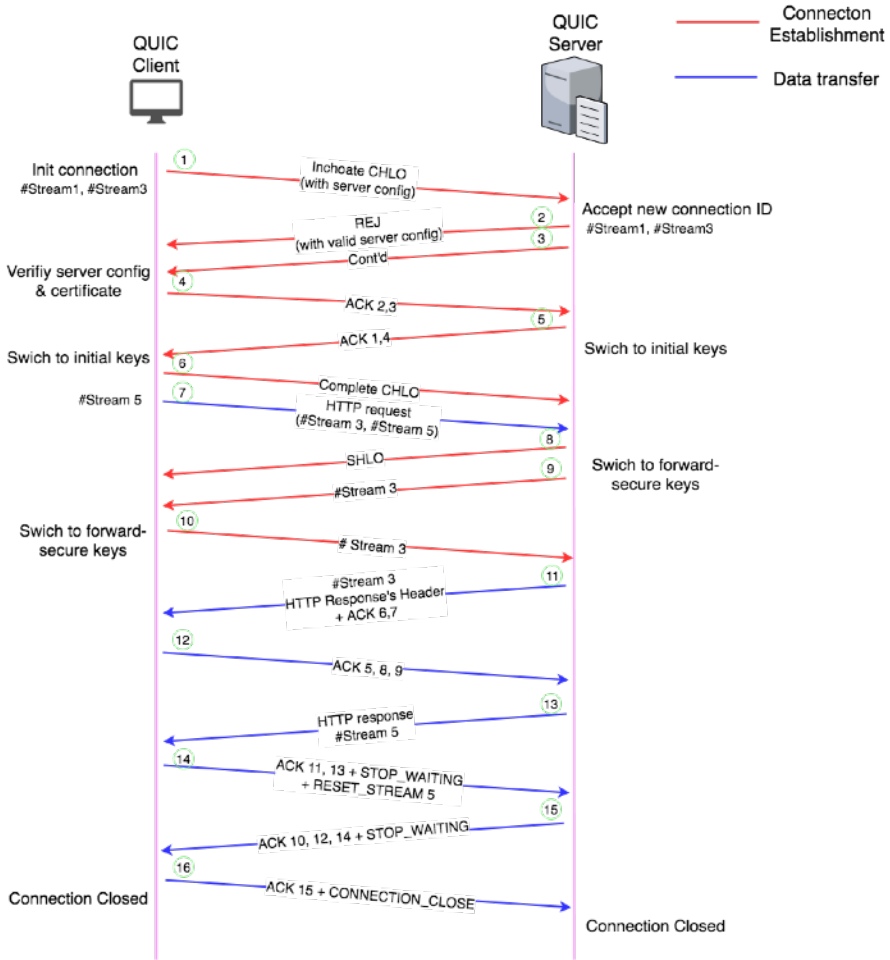
```
QuicSpdyClientStream* QuicSpdyClientBase::CreateClientStream() {
  // some code here
  …

  auto* stream = static_cast<QuicSpdyClientStream*>(
      client_session()->CreateOutgoingDynamicStream());
  if (stream) {
    stream->SetPriority(QuicStream::kDefaultPriority);
    stream->set_visitor(this);
  }
  return stream;
}
```

Client request is an outgoing stream!

QUIC Packet Exchanges

(Toy Client + Toy Server)

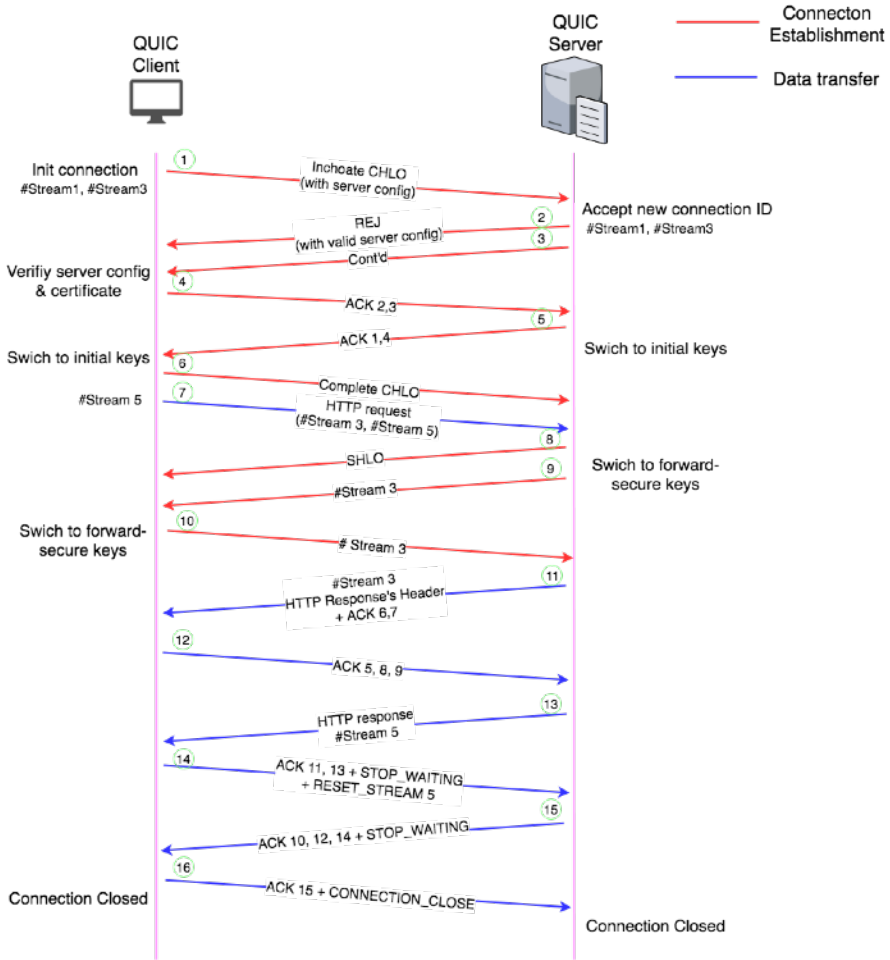


- Initial handshake: Initial keys
- Final handshake: forward-secure keys

QUIC Packet Exchanges

(Toy Client + Toy Server)

Notes:
- 4,5 are optional

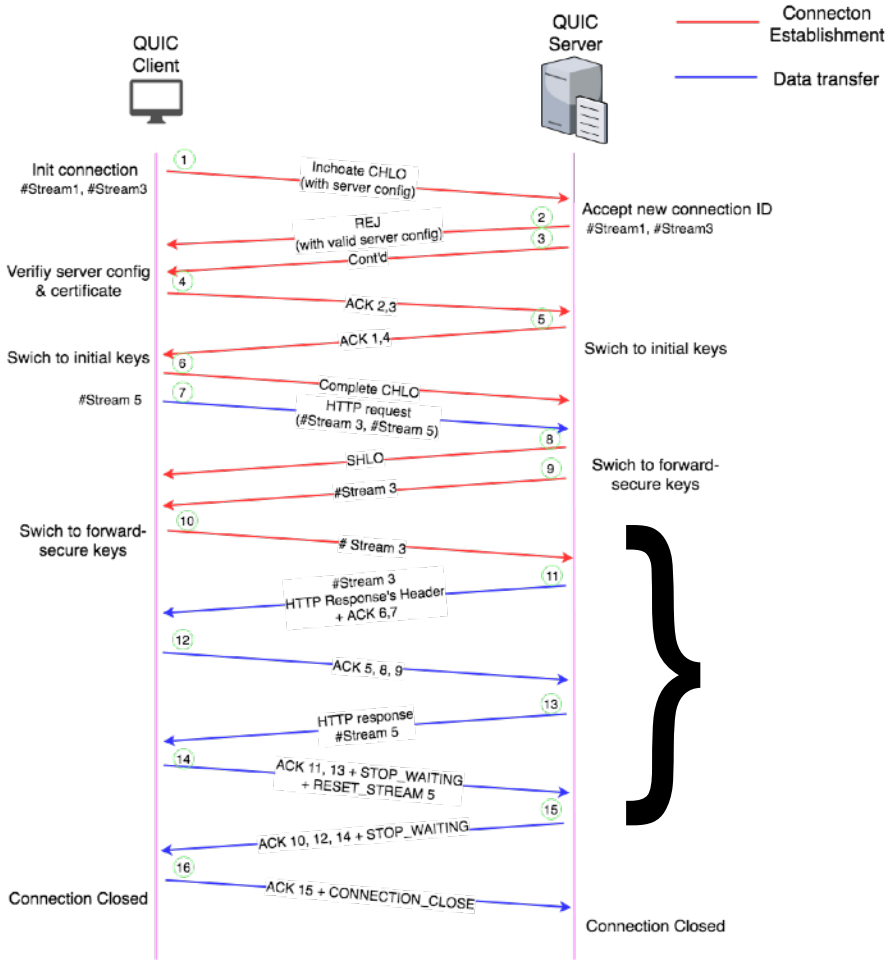


- Initial handshake: Initial keys
- Final handshake: forward-secure keys

QUIC Packet Exchanges

(Toy Client + Toy Server)

Notes:
- 4,5 are optional
- Some packets are not necessarily to be in this order!

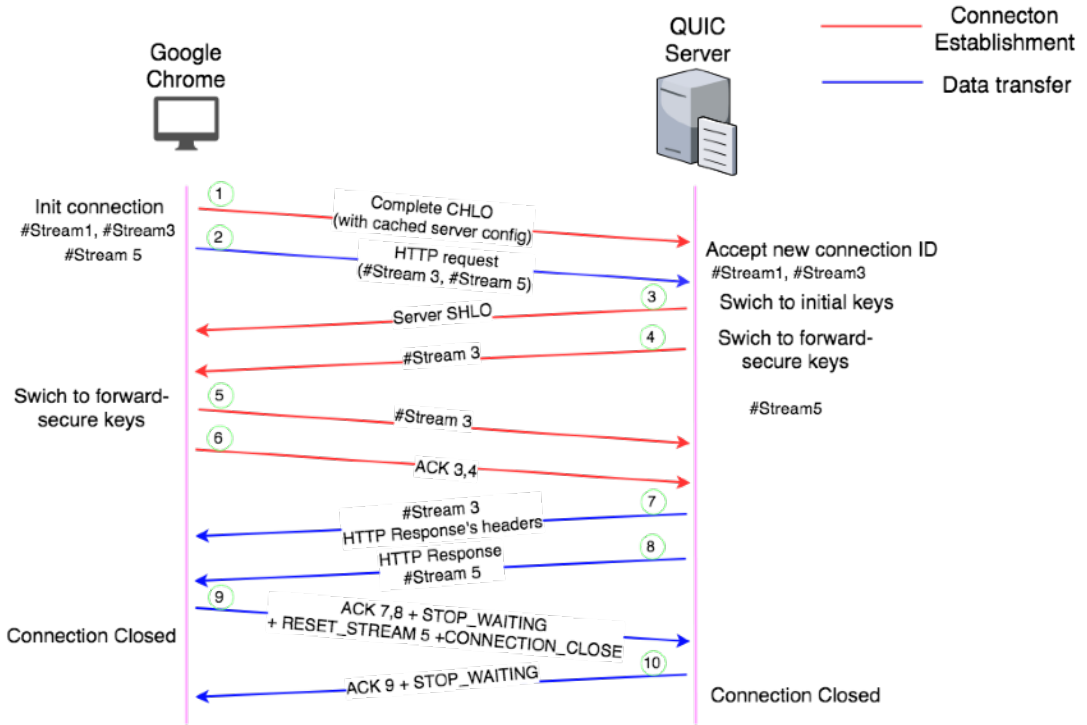


- Initial handshake: Initial keys
- Final handshake: forward-secure keys

QUIC Packet Exchanges

(Chrome + Toy Server)

- 0-RTT



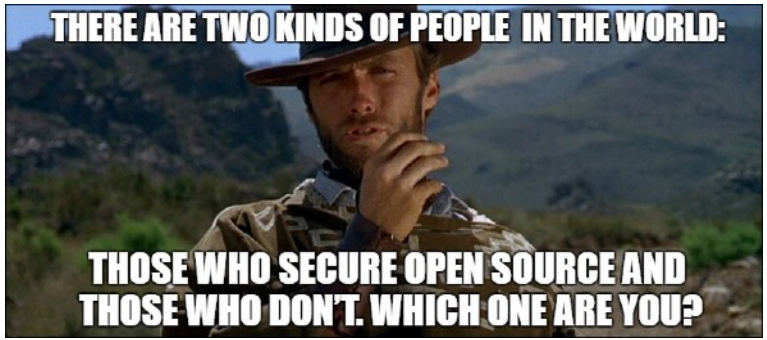
- Initial handshake: Initial keys
- Final handshake: forward-secure keys

QUIC Chromium vs QUIC IETF

- Developed and initiated by Google.
- Submitted to IETF and has been a work in progress until now.
- Open source as in Chromium project, but not really…

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	GQUIC	1382	Client Hello, PKN: 1, CID: 9518021173104961792
2	0.000295	127.0.0.1	127.0.0.1	GQUIC	346	Payload (Encrypted), PKN: 2, CID: 9518021173104961792
3	0.023538	127.0.0.1	127.0.0.1	GQUIC	1382	Rejection, PKN: 1, CID: 9518021173104961792
4	0.028676	127.0.0.1	127.0.0.1	GQUIC	1382	Client Hello, PKN: 3, CID: 9518021173104961792

No.	Time	Source	Destination	Protocol	Length	Info
3	0.340318	127.0.0.1	127.0.0.1	QUIC	1382	SH, Protected Payload (KP0), PKN: 81, CID: 0x705b9f2e9164be1b
4	0.352158	127.0.0.1	127.0.0.1	QUIC	1382	SH, Protected Payload (KP0), PKN: 1, CID: 0x705b9f2e9164be1b
5	0.353602	127.0.0.1	127.0.0.1	QUIC	1382	SH, Protected Payload (KP0), PKN: 2, CID: 0x705b9f2e9164be1b
6	0.356065	127.0.0.1	127.0.0.1	QUIC	70	SH, Protected Payload (KP0), PKN: 2, CID: 0x705b9f2e9164be1b



QUIC Chromium vs QUIC IETF

- Different HTTP request to streams mapping
- Different cryptographic function
- Different Stream Types and formats. Example:

  WINDOW_UPDATE (GQUIC) == MAX_DATA & MAX_STREAM_DATA (IETF)
  GQUIC only: GOAWAY
  IETF only: STREAM_BLOCKED, STREAM_ID_BLOCKED, MAX_STREAM_ID

- Wireshark (will) supports only QUIC IETF decryption

QUIC Chromium

- QUIC “Toy” Server and Client,
- C++
- Single-threaded, not thread safe -> blocking
- Bundled within Chromium project

- Use their own cryptographic function for TLS
- Toy client does not support 0-RTT
- By default, implement Cubic for congestion control (can be replaced by BBR)

QUIC IETF

- Immature implementations are available online.
- Its standardization is still being discussed in IETF QUICWG

- E.g. ngtcp2: https://github.com/ngtcp2/ngtcp2
- Based on 4th implementation draft
- C++
- Use openssl for TLSv1.3
- Support 0-RTT
- Has yet started to work on congestion control: TCP NewReno

Conclusions

- QUIC is designed to replace TCP + TLS stack.
- QUIC is user-space transport layer protocol.
- QUIC can reduce RTT for connection establishment and avoids HOL

However
- Currently, the available QUIC codes are still immature.
- Different structures between QUIC Chromium and IETF.

Future works

- QUIC API for other programming languages
- QUIC benchmark & testing over multiple setups

Thank you

Questions?

Main References

(1) A. Langley et al., “The QUIC Transport Protocol: Design and Internet-Scale Deployment,” in Proceedings of the Conference of the ACM Special Interest Group on Data Communication, New York, NY, USA, 2017, pp. 183–196.

(2) A. M. Kakhki, S. Jero, D. Choffnes, C. Nita-Rotaru, and A. Mislove, “Taking a Long Look at QUIC: An Approach for Rigorous Evaluation of Rapidly Evolving Transport Protocols,” in Proceedings of the 2017 Internet Measurement Conference, New York, NY, USA, 2017, pp. 290–303.