SharkFest '17 Europe

20 QUIC Dissection

Using Wireshark to Understand QUIC Quickly

Megumi Takeshita

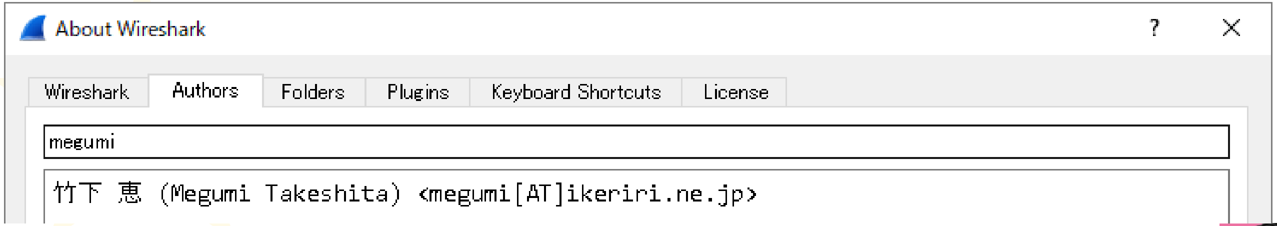
ikeriri network service

ParkSuite Classroom
11 November 2017
11:15am-12:30pm

supplimental files
http://www.ikeriri.ne.jp/sharkfest

Megumi Takeshita, ikeriri network service

- Founder, ikeriri network service co.,ltd
- Wrote 10+ books about Wireshark
- Reseller of Riverbed Technology (former CACE technologies) in Japan
- Attending all Sharkfest
- Translator of QT Wireshark into Japanese



20 QUIC: Using Wireshark to Understand QUIC Quickly

In this presentation, Megumi explains the details of QUIC, and shows you how to understand the protocol and mechanisms involved.
Using sample trace files,Megumi will show how to inspect and visualize QUIC traffic and explain the advantage of QUIC in comparison with other protocols too.

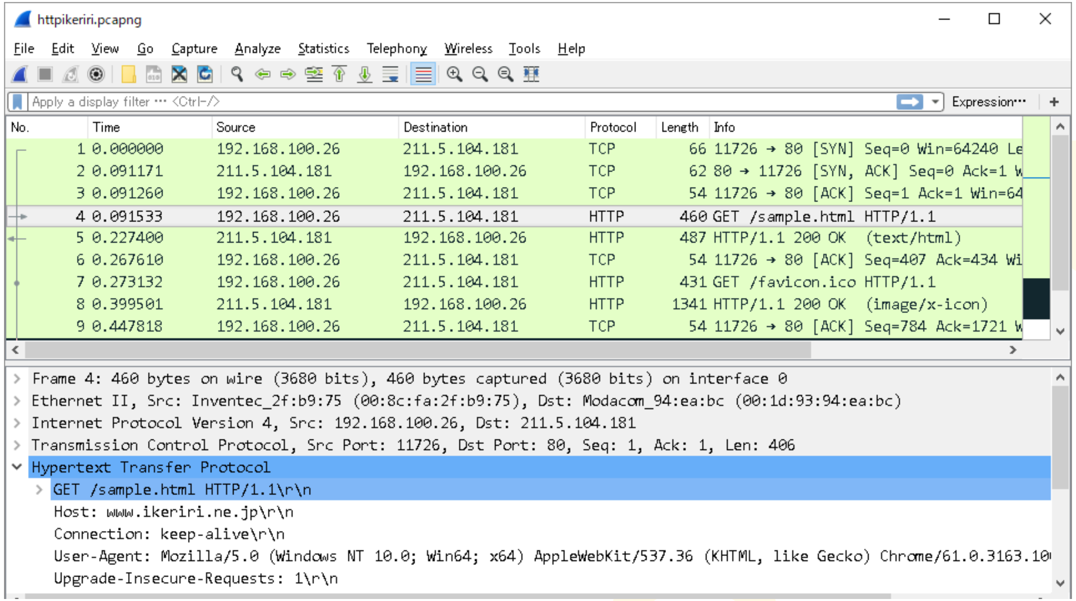
NOTE: IQUIC(IETF QUIC) is Internet-Draft and now standardizing, so some specification may be changed and the sample trace file is not adequate

Set up environment

- For QUIC dissection, we need nightly build version of Wireshark (this time I use 2.5.0-1547-gbe625b9b development version)
- All supplemental files of this presentation is below http://www.ikeriri.ne.jp/sharkfest (temporal)

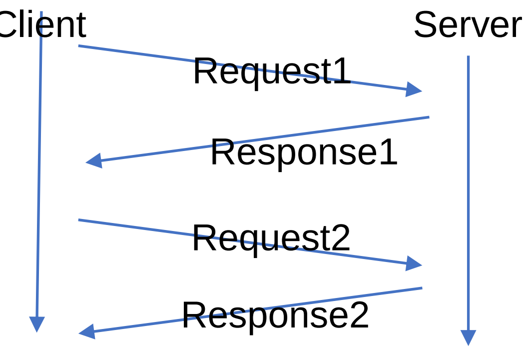
Open simple HTTP/1.1

- open httpikeriri.pcapng of simple HTTP/1.1 packet,

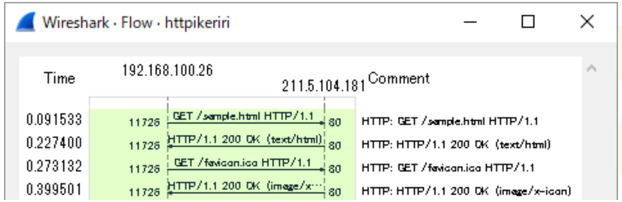


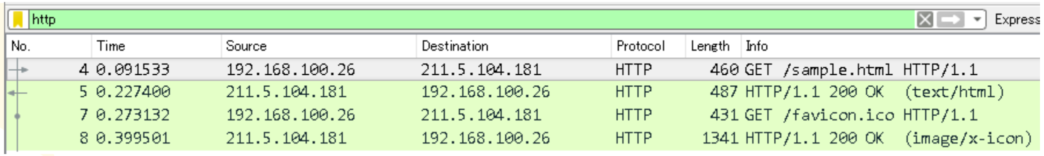
- HTTP/1.1 request response loop
- Head of Line blocking
- Rich application needs many TCP connection (AJAX)

HTTP/1.1 is difficult to speed up



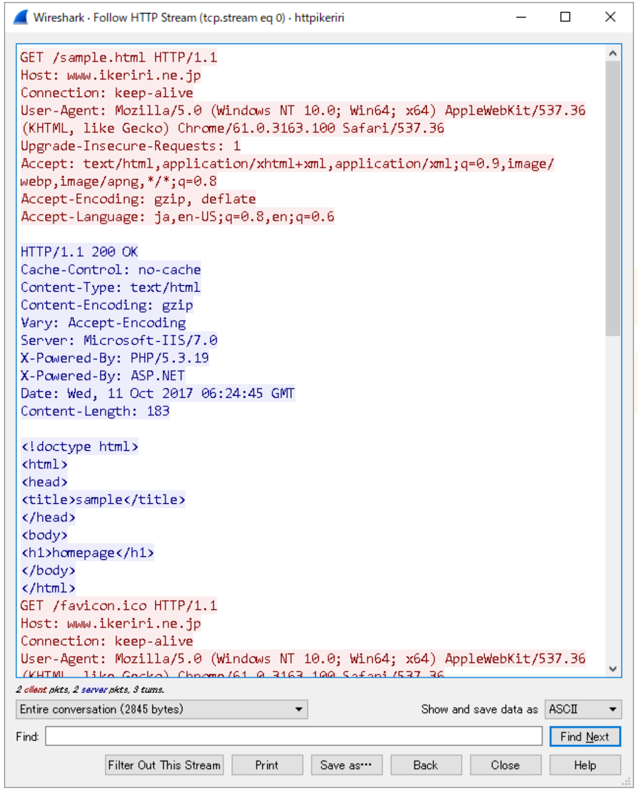
- HTTP request have to send after previous response has been received.
- Please input display filter in Wireshark “http.next_request_in” (Next request in frame in HTTP request)
- HTTP request is always waiting in one connection. (head line blocking)
- Display filter “http” and Statistics>Flow Graph





HTTP/1.1 is text based, not efficient protocol

- Right click HTTP header and “follow http stream”
- HTTP is text-based application protocol, easy to read,
  but not efficient, ambiguous, and redundant
- HTTP messages are clear texts so they uses more data and CPU power for dissection.
- Many connections are separated by each other TCP connection, they work their own TCP rules without HTTP.

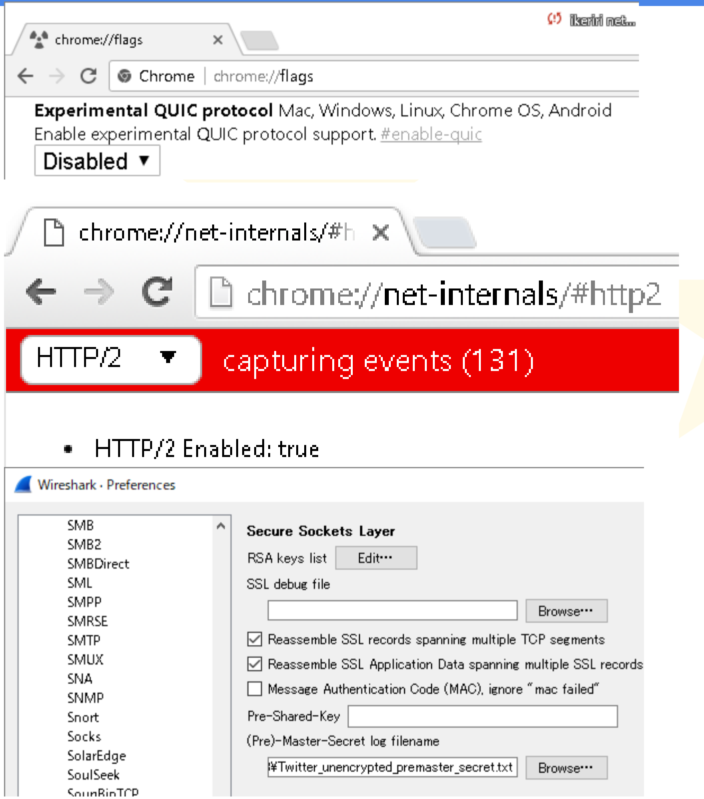


Accellerate Web service

- Wider bandwidth, Faster computing in todays internet, then what is the protocol ?
  HTTP/1.0 (RFC1945-,1996)
  HTTP/1.1 (RFC2068-,1997)
- New generation of web protocol comes
  HTTP/2.0 (RFC7540-,2015) former SPDY
  Google, Facebook, Twitter, Yahoo, and major website using Chrome, Edge, Safari and major browser

www.twitter.com with HTTP/2.0

- Set SSLKEYLOGFILE variable to decrypt SSL/TLS
- Open Chrome URL "chrome://flags/" and disable QUIC protocol in list box, now Chrome prefer to use HTTP2
- Start capture and open www.twitter.com, type chrome://net-internals/#http2 you can see the HTTP/2 sessions
- This time open twitter.pcapng and set (Pre)-Master-Secret log filename Twitter_unencrypted_premaster_secret.txt in SSL preference

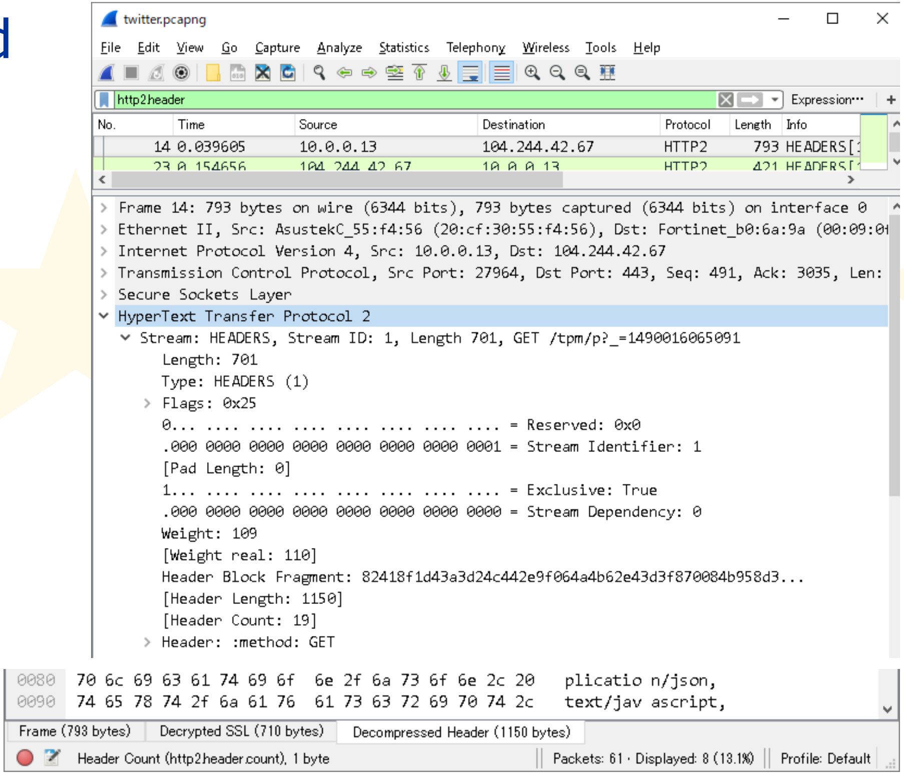


HTTP/2.0 uses binary frame with Huffman coding compression in a SSL/TLS connection

- Set “http2.header” in display filter and check the #14
- The packet contains EthernetII, IPv4, TCP, SSL, and HTTP2 header

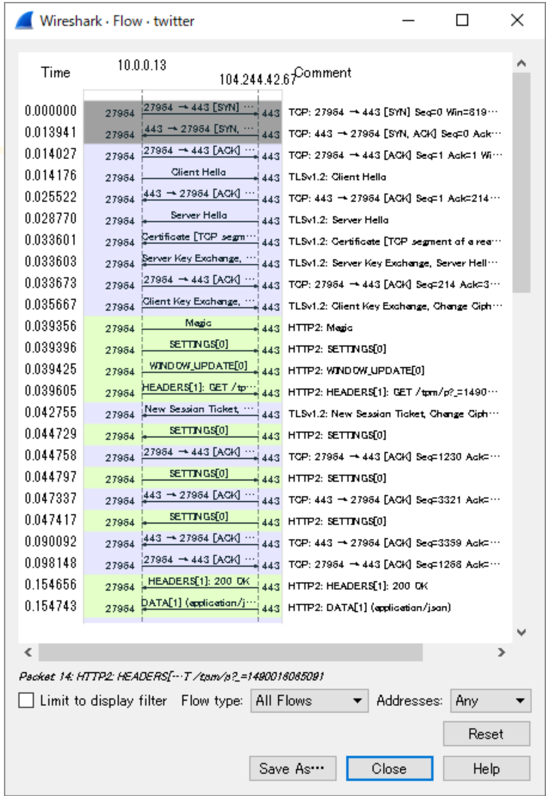
Application	HTTP/1.1 semantics
Session	HTTP/2.0
Session	SSL/TLS
Transport	TCP

- HTTP/2.0 uses binary frame with Huffman coding, check packet bytes pane



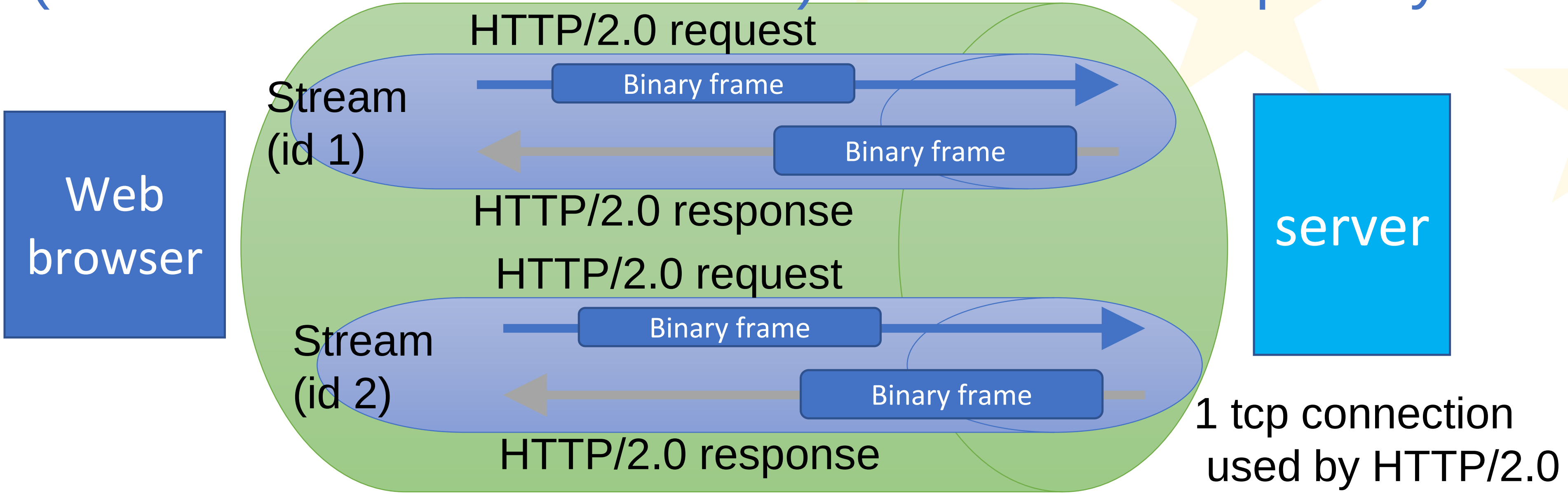
Connection process of HTTP/2.0

- Click Statistics > Flow Graph and check connection process of HTTP/2.0
- HTTP/2.0 needs TCP 3 way handshake that contains 1 RTT(round trip time) SYN-SYN/ACK-ACK from Client side
- HTTP/2.0 needs SSL/TLS connection that contains 2 RTT(round trip time) from Client side Client Hello/Server Hello-Certificate-Server Key Exchange-Server Hello Done/Client Key Exchange -New Session Ticket(TLS)-Change Cipher Spec-Finished at the first time
- We need TCP 1 and SSL/TLS 2 RTT at the first time



HTTP/2.0 Stream mechanism

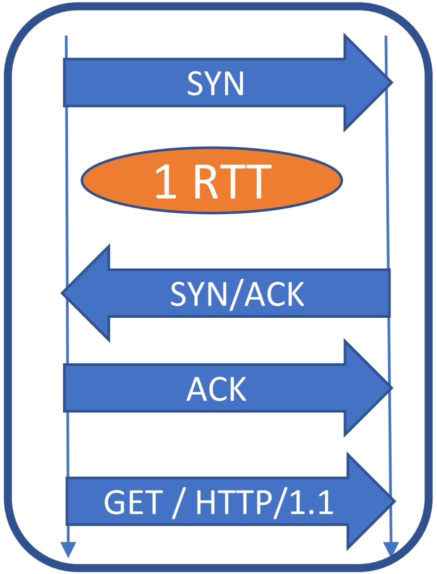
HTTP/2.0 uses 1 tcp connection and many Stream (virtual connection channel) that has id and priority



Multiple HTTP connections (at the first time)

HTTP/1.1

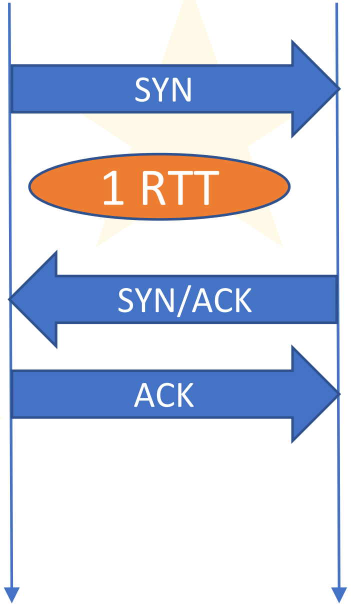
TCP connection



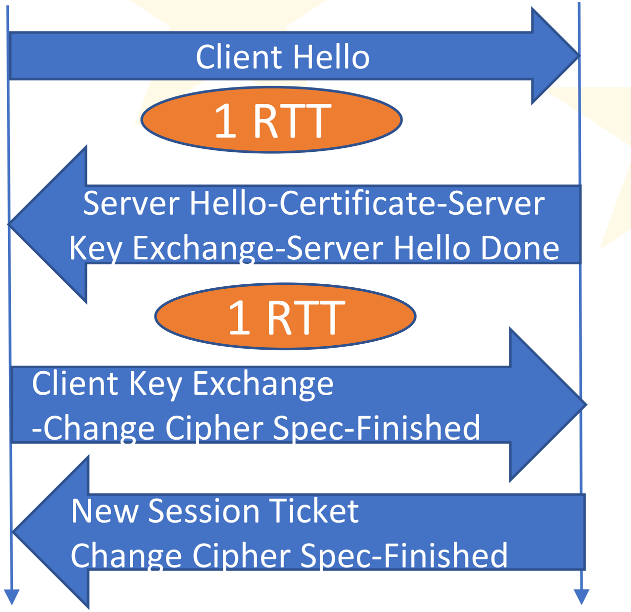
X many times

HTTP/2.0

TCP connection



SSL/TLS1.0 connection

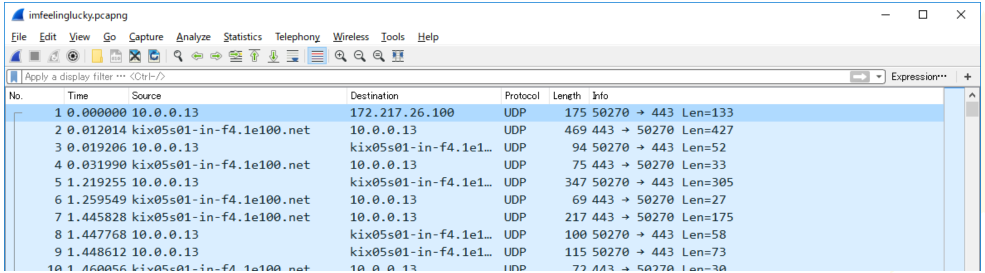


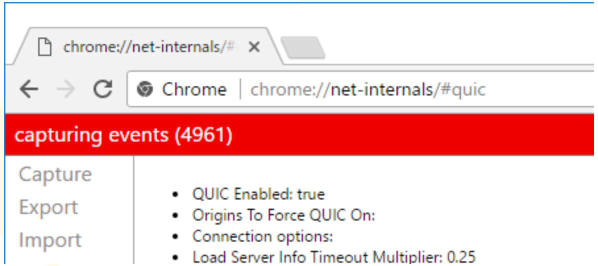
GQUIC

- Google creates proprietary protocol, QUIC (Quick UDP Internet Connection) (a.k.a. GQUIC)
- GQUIC omits TCP, SSL/TLS and HTTP/2.0 and provides a monolithic mechanism of TCP + SSL/TLS authentication and encryption + HTTP/2 multiplexing and compression in UDP stream
- Already used in Google service (Gmail, Youtube,…)
- QUIC needs just 1-RTT at the first time, and no RTT (0-RTT) when we connect again (if resumption successes)

imfeelinglucky.pcapng

- Open imfeelinglucky.pcapng, it is the packet that just I pushed I’m feeling lucky button at google using Chrome





- At this time we just see some UDP streams of QUIC
- Open the Chrome and type chrome://net-internals/#quic you can see current QUIC sessions

Host	Version	Peer address	Connection UID	Active stream count	Active streams	Total stream count	Packets Sent	Packets Lost	Packets Received	Connected
apis.google.com:443 ogs.google.com:443	QUIC_VERSION_35	216.58.197.142:443	8126651583460386461	0	None	1	7	0	7	true
clients4.google.com:443	QUIC_VERSION_35	172.217.26.46:443	7635153523791641412	0	None	1	8	0	7	true
clients5.google.com:443	QUIC_VERSION_35	172.217.26.46:443	3964807834305814218	0	None	0	3	0	3	true
fonts.gstatic.com:443 ssl.gstatic.com:443	QUIC_VERSION_35	172.217.26.35:443	3002020916089671232	0	None	0	3	0	3	true
lh3.googleusercontent.com:443	QUIC_VERSION_35	216.58.197.129:443	1788128156380661810	0	None	0	4	0	4	true

Check GQUIC packets

- Check header encapsulation (Ethernet II, IP, UDP, and QUIC) and payloads are encrypted

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.0.0.13	172.217.26.100	QUIC	175	SH, Protected Payload (KP0), PKN: 205
2	0.012014	172.217.26.100	10.0.0.13	QUIC	469	SH, Protected Payload (KP0), PKN: 7
3	0.019206	10.0.0.13	172.217.26.100	QUIC	94	SH, Protected Payload (KP0), PKN: 205

```
> Frame 1: 175 bytes on wire (1400 bits), 175 bytes captured (1400 bits) on interface 0
> Ethernet II, Src: AsustekC_55:f4:56 (20:cf:30:55:f4:56), Dst: Fortinet_b0:6a:9a (00:09:0f:b0:6a:9
> Internet Protocol Version 4, Src: 10.0.0.13, Dst: 172.217.26.100
> User Datagram Protocol, Src Port: 50270, Dst Port: 443
v QUIC (Quick UDP Internet Connections) IETF
    0... .... = Header Form: Short Header (0)
    .0.. .... = Connection ID Flag: False
    ..0. .... = Key Phase Bit: False
    ...0 1100 = Packet Type: Unknown (12)
    Packet Number: 205
    Protected Payload: d7df8e5231d78606c86a83da1f6e65a50765b9f2cc2b8aca...
```

Packet Number	Info
205	SH, Protected Payload (KP0), PKN: 205
7	SH, Protected Payload (KP0), PKN: 7
205	SH, Protected Payload (KP0), PKN: 205
8	SH, Protected Payload (KP0), PKN: 8
205	SH, Protected Payload (KP0), PKN: 205
9	SH, Protected Payload (KP0), PKN: 9
10	SH, Protected Payload (KP0), PKN: 10
205	SH, Protected Payload (KP0), PKN: 205
205	SH, Protected Payload (KP0), PKN: 205
11	SH, Protected Payload (KP0), PKN: 11
12	SH, Protected Payload (KP0), PKN: 12

- This is not a first connection, so it immediately starts data transaction (0-RTT) because we can see SH(Short Header) at Header Form field.
- 64-bit packet number is used as a part of nonce. Each endpoint uses a separate packet number, that is increasing.

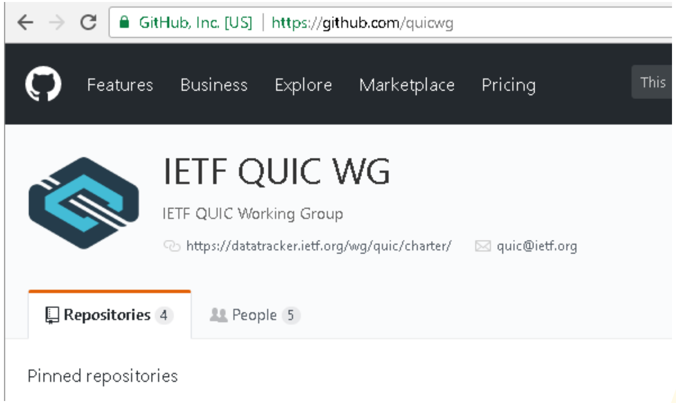
IQUIC (IETF Quick UDP Internet Connection)

- Now IETF standardize IETF QUIC (a.k.a. IQUIC)
- IQUIC also provides a monolithic mechanism of TCP reliable transport + SSL/TLS1.3 authentication and encryption + HTTP/2 multiplexing and compression
- Now Internet-Draft (October13, 2017) https://tools.ietf.org/html/draft-ietf-quic-transport-07
- Data tracker (IETF) https://datatracker.ietf.org/wg/quic

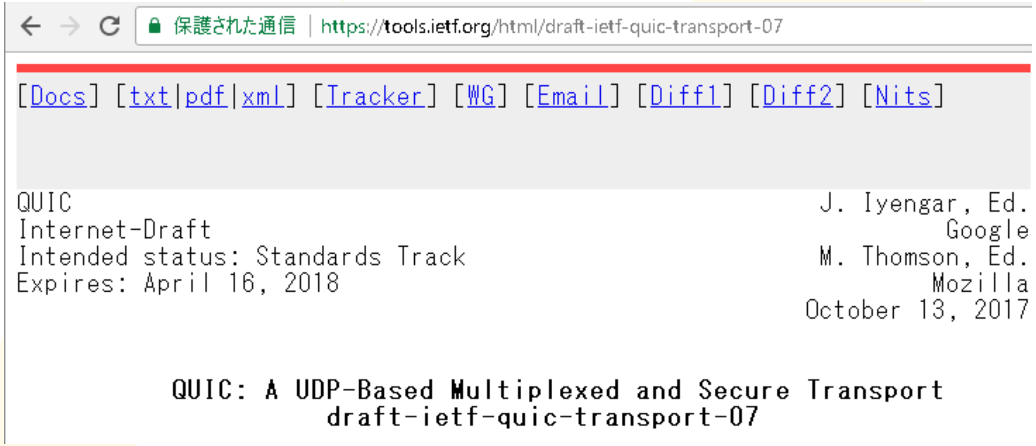
TCP+SSL/TLS+HTTP/2.0=QUIC

IETF QUIC standards

- Working Group
  https://github.com/quicwg



- Internet-Draft (October, 2017)
  https://tools.ietf.org/html/draft-ietf-quic-transport-07



Core specification

IETF QUIC standards

- QUIC-TLS (October, 2017)
  https://tools.ietf.org/html/draft-ietf-quic-tls-07

保護された通信 | https://tools.ietf.org/html/draft-ietf-quic-tls-07

[Docs] [txt|pdf|xml|html] [Tracker] [WG] [Email] [Diff1] [Diff2] [Nits]

Versions: (draft-thomson-quic-tls) 00 01 02
03 04 05 06 07

QUIC M. Thomson, Ed.
Internet-Draft Mozilla
Intended status: Standards Track S. Turner, Ed.
Expires: April 16, 2018 sn3rd
October 13, 2017

Using Transport Layer Security (TLS) to Secure QUIC
draft-ietf-quic-tls-07

Using TLS in QUIC

Open sample packets of IETF QUIC

- Open quic_ietf_draft05_ngtcp2.pcapng using Wireshark (Thank you Alexis-san for dissector and sample pcap file)
- View> Coloring rules..., new rule name: UDP source port 443, set filter udp.srcport==443, and set pink color at background

Wireshark · Coloring Rules · Default

Name	Filter
☑ UDP source port 443	udp.srcport==443

- Blue color is from Client and Pink is from Server

1	0.000000000	12…	12…	QUIC	1294	LH, Client Initial, PKN: 558625387, CID: 0x8ee4cfaf7e9f5d9c
2	0.037343527	12…	12…	QUIC	1283	LH, Server Cleartext, PKN: 726976297, CID: 0x5ab56b082f4e162c
3	0.037665201	12…	12…	QUIC	221	LH, Server Cleartext, PKN: 726976298, CID: 0x5ab56b082f4e162c
4	0.038324823	12…	12…	QUIC	159	LH, Client Cleartext, PKN: 558625388, CID: 0x5ab56b082f4e162c
5	0.038578488	12…	12…	QUIC	83	SH, Protected Payload (KP0), PKN: 726976299, CID: 6536248117095700012
6	4.459336855	12…	12…	QUIC	91	SH, Protected Payload (KP0), PKN: 558625389, CID: 6536248117095700012
7	4.459571939	12…	12…	QUIC	106	SH, Protected Payload (KP0), PKN: 726976300, CID: 6536248117095700012

Long header of QUIC

- Click #1 packet and check QUIC header format

```
User Datagram Protocol, Src Port: 39916, Dst Port: 443
QUIC (Quick UDP Internet Connections) IETF
    1... .... = Header Form: Long Header (1)
    .000 0010 = Packet Type: Client Initial (2)
    Connection ID: 0x8ee4cfaf7e9f5d9c
    Packet Number: 558625387
    Version: draft-05 (0xff000005)
  v STREAM Stream ID: 0
    v Frame Type: STREAM (0xc1)
          11.. .... = Stream: 0x3
          ..0. .... = Fin(F): False
          ...0 0... = Stream Length (SS): 1 Byte (0)
          .... .00. = Offset Length (OO): 0 Byte (0)
          .... ...1 = Data Length (D): 2 Bytes
      Stream ID: 0 (Cryptographic handshake)
      Data Length: 274
      Stream Data: 160301010d010001090303a29b79ac62f9
    > Secure Sockets Layer
  > PADDING Length: 948
    Hash: 4e8c7f5146059fb9
```

64-bit random connection ID from the client

Long headers are used for negotiation and establishment of 1-RTT keys
Once both conditions are met, a sender switches to send short header

Packet type indicates the frame type of QUIC

64-bit packet number is used as pa art of nonce. Each endpoint uses a separate packet number, that is increasing.

Stream is the same mechanism of HTTP/2.0 stream, and Stream ID 0 is reserved for cryptographic handshake (TLS1.3)

Packet Type (October, 2017)

type	Name	Explanation
0x01	Version Negotiation	Server sends this type packet for not supporting client's version (Long header)
0x02	Client Initial	Client sends this type packet for initializing handshake (Long Header)
0x03	Server Stateless Retry	Server sends this type packet as cryptographic handshake message and ACK for requiring a new Client Initial packet (Long Header)
0x04	Server Cleartext	Server sends this type packet as cryptographic handshake message and ACK that contains server chosen connection ID and randomized packet number with with STREAM, PADDING, ACK. (Long Header)
0x05	Client Cleartext	Client sends this type packet as the receipt of Server Cleartext message, Client Cleartext contains Server selected connection ID and incremented packet number of Client Initial with STREAM, PADDING, ACK. (Long Header)
0x06	0-RTT Protected	Packets that are protected with 0-RTT keys are sent with Long Header; all packets protected with 1-RTT keys are sent with Short Header. Packets protected with 0−RTT keys use a type value of 0x06. The connection ID field for a 0−RTT packet is selected by the client.

connection ID / packet number

- Click connection ID field, right click and "Apply as column" (same as packet number) in #1 packet, and check the changes of both
- Server set 64-bit the random connection ID in #2 packet, Client updates the connection ID as the same number
- Packet number is set randomly (0 and 2^31-1) and used as a part of nonce. Each endpoint uses a separate packet number, that is increasing

	Time	Source	Destina	Protocol	Length	Connection ID	Packet Number	Info
1	0.00…	127…	127…	QUIC	1294	0x8ee4cfaf7e9f5d9c	558625387	LH, Client Initial, PKN: 558625387, CID: 0x8ee4cfaf7e9f5d9c
2	0.03…	127…	127…	QUIC	1283	0x5ab56b082f4e162c	726976297	LH, Server Cleartext, PKN: 726976297, CID: 0x5ab56b082f4e162c
3	0.03…	127…	127…	QUIC	221	0x5ab56b082f4e162c	726976298	LH, Server Cleartext, PKN: 726976298, CID: 0x5ab56b082f4e162c
4	0.03…	127…	127…	QUIC	159	0x5ab56b082f4e162c	558625388	LH, Client Cleartext, PKN: 558625388, CID: 0x5ab56b082f4e162c
5	0.03…	127…	127…	QUIC	83	0x5ab56b082f4e162c	726976299	SH, Protected Payload (KP0), PKN: 726976299, CID: 6536248117095700012
6	4.45…	127…	127…	QUIC	91	0x5ab56b082f4e162c	558625389	SH, Protected Payload (KP0), PKN: 558625389, CID: 6536248117095700012
7	4.45…	127…	127…	QUIC	106	0x5ab56b082f4e162c	726976300	SH, Protected Payload (KP0), PKN: 726976300, CID: 6536248117095700012
8	4.45…	127…	127…	QUIC	83	0x5ab56b082f4e162c	726976301	SH, Protected Payload (KP0), PKN: 726976301, CID: 6536248117095700012
9	4.45…	127…	127…	QUIC	83	0x5ab56b082f4e162c	558625390	SH, Protected Payload (KP0), PKN: 558625390, CID: 6536248117095700012

Stream ID (encrypted in Short Header)

- IQUIC packet has a 32-bit STREAM id for multiplexing many data connections.
- Clients use odd-number, Server use even-number, 0 is reserved for cryptographic Handshake (usually TLS connection)
- IQUIC stream mechanism is almost the same as HTTP/2.0(also as TCP)
- Stream change the state, Many streams in a UDP connection

```
                        +--------+
                        |        |
                        |  idle  |
                        |        |
                        +--------+
                             |
                 send/recv STREAM/RST
                      recv MSD/SB
                             |
                             v
        recv FIN/       +--------+       send FIN/
        recv RST        |        |       send RST
          ,-------------|  open  |-------------.
         /              |        |              ¥
        v               +--------+               v
  +----------+                             +----------+
  |   half   |                             |   half   |
  |  closed  |                             |  closed  |
  | (remote) |                             |  (local) |
  +----------+                             +----------+
      |                                          |
      |   send FIN/     +--------+     recv FIN/ |
      ¥   send RST      |        |     recv RST  /
       `--------------->| closed |<------------'
                        |        |
                        +--------+

send:    endpoint sends this frame
recv:    endpoint receives this frame

STREAM:  a STREAM frame
FIN:     FIN flag in a STREAM frame
RST:     RST_STREAM frame
MSD:     MAX_STREAM_DATA frame
SB:      STREAM_BLOCKED frame
```

Short header of QUIC

- Click #5 packet and check QUIC IETF header

```
Frame 5: 83 bytes on wire (664 bits), 83 bytes captured (664 bits) on interface 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
User Datagram Protocol, Src Port: 443, Dst Port: 39916
QUIC (Quick UDP Internet Connections) IETF
    0... .... = Header Form: Short Header (0)
    .1.. .... = Connection ID Flag: True
    ..0. .... = Key Phase Bit: False
    ...0 0011 = Packet Type: 4 octet (3)
    Connection ID: 0x5ab56b082f4e162c
    Packet Number: 726976299
    Protected Payload: a48cd45b995a5391...6448d31ac728...
```

Set 0 : Connection ID field is omitted
Set 1 : Connection ID field is present

Every time that a new set of keys is used for protecting outbound packets, the KEY_PHASE bit in the public flags is toggled.

64-bit random
Server chosen
connection ID

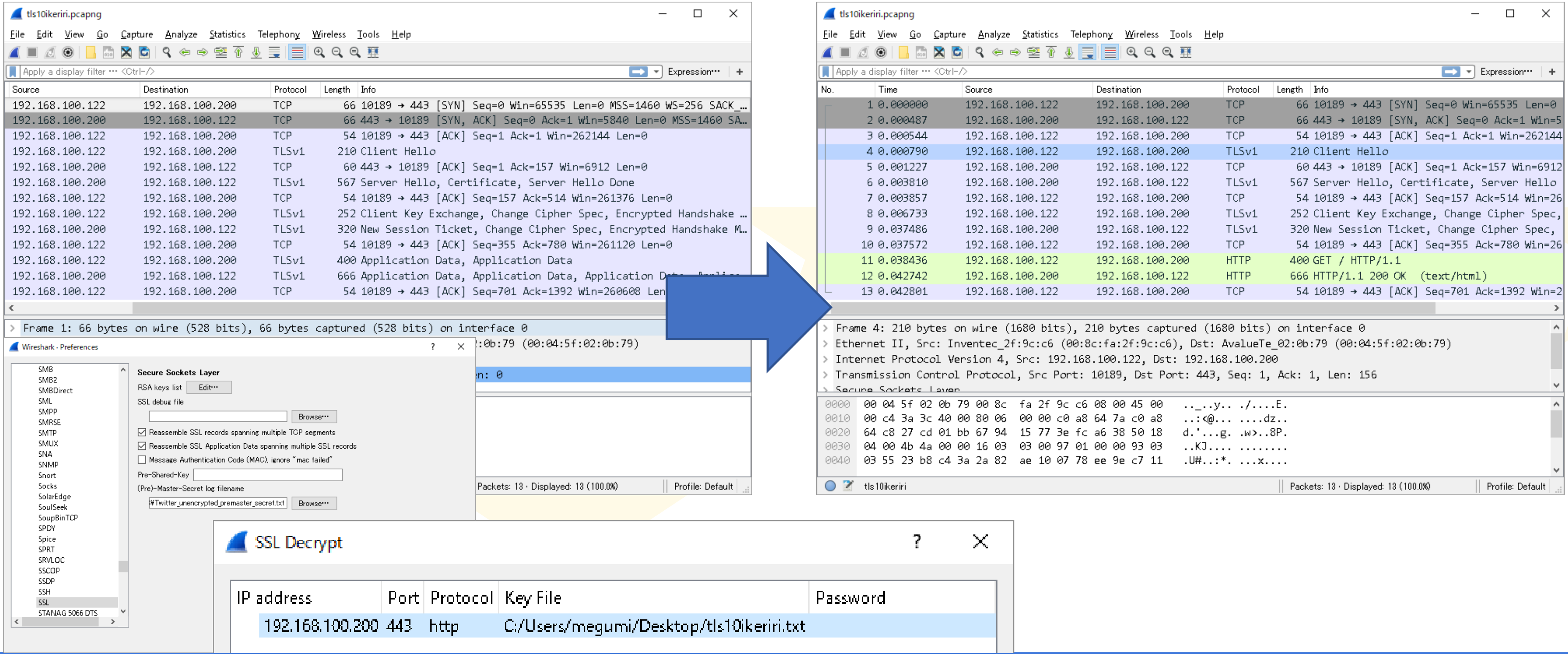
64-bit packet number is used as pa art of nonce. Each endpoint uses a separate packet number, that is increasing.

The short header can be used after the version and 1-RTT keys are negotiated.

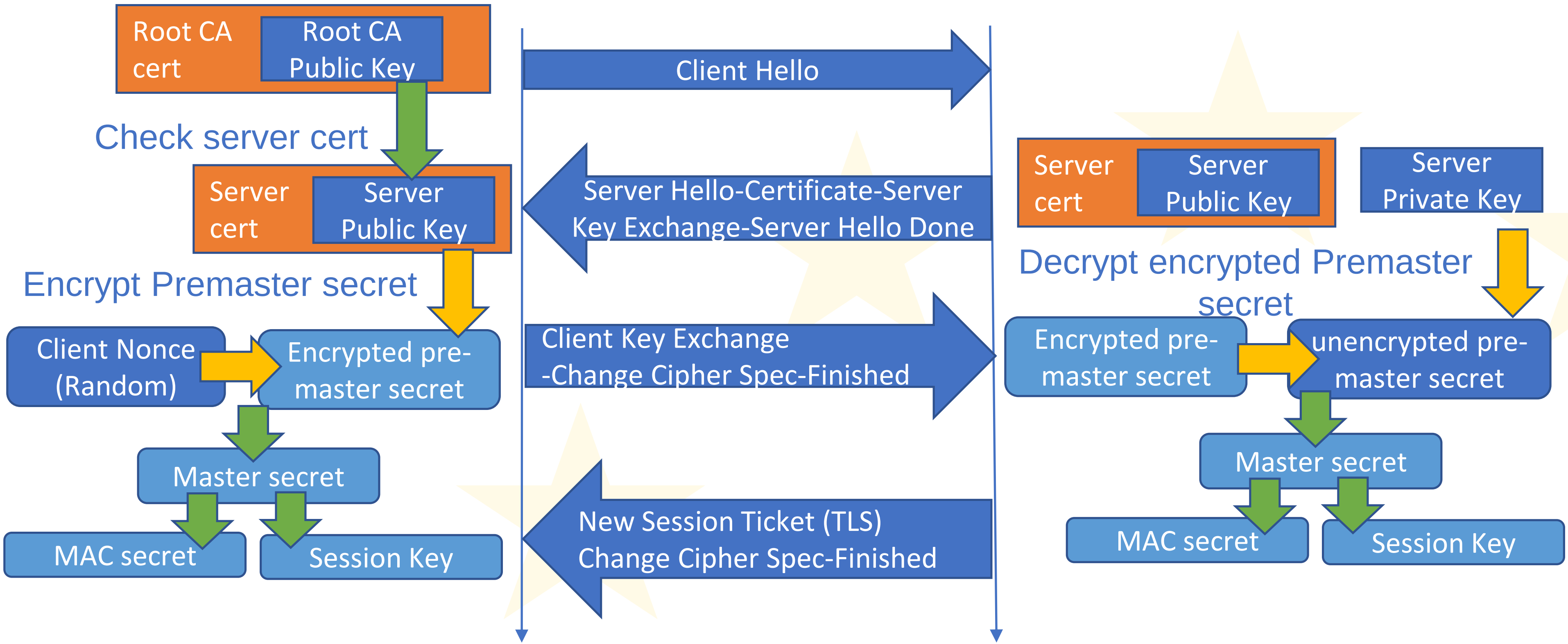
How to negotiate and install session key in IQUIC

- IQUIC is learned from SSL/TLS to install session key, but how do QUIC install session key at the first time (1-RTT) and at resumption (0-RTT)
- Open tls10ikeriri.pcapng to remember how to negotiate and install session key in TLS1.0
- tls10ikeriri.txt is a PEM format certification file with server's private key
- Set RSA key list in SSL preference of Wireshark

Open tls10ikeriri.pcapng and set RSA key list (tls10ikeriri.txt)

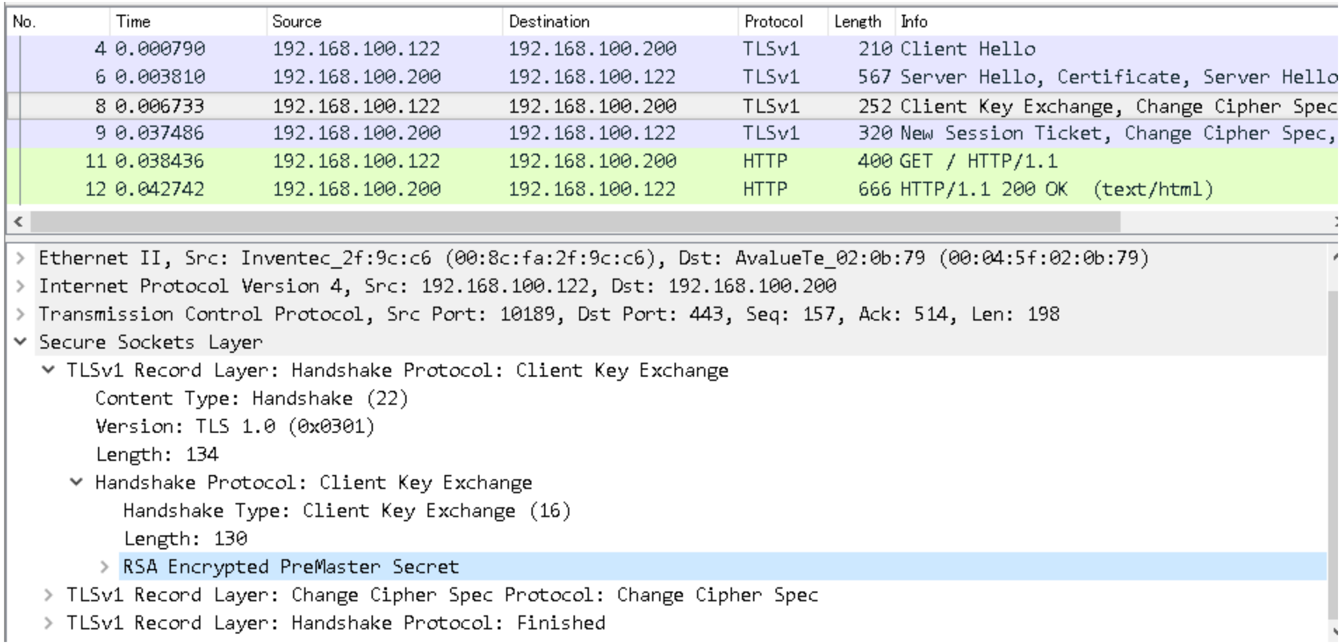


Key creation process of TLS1.0



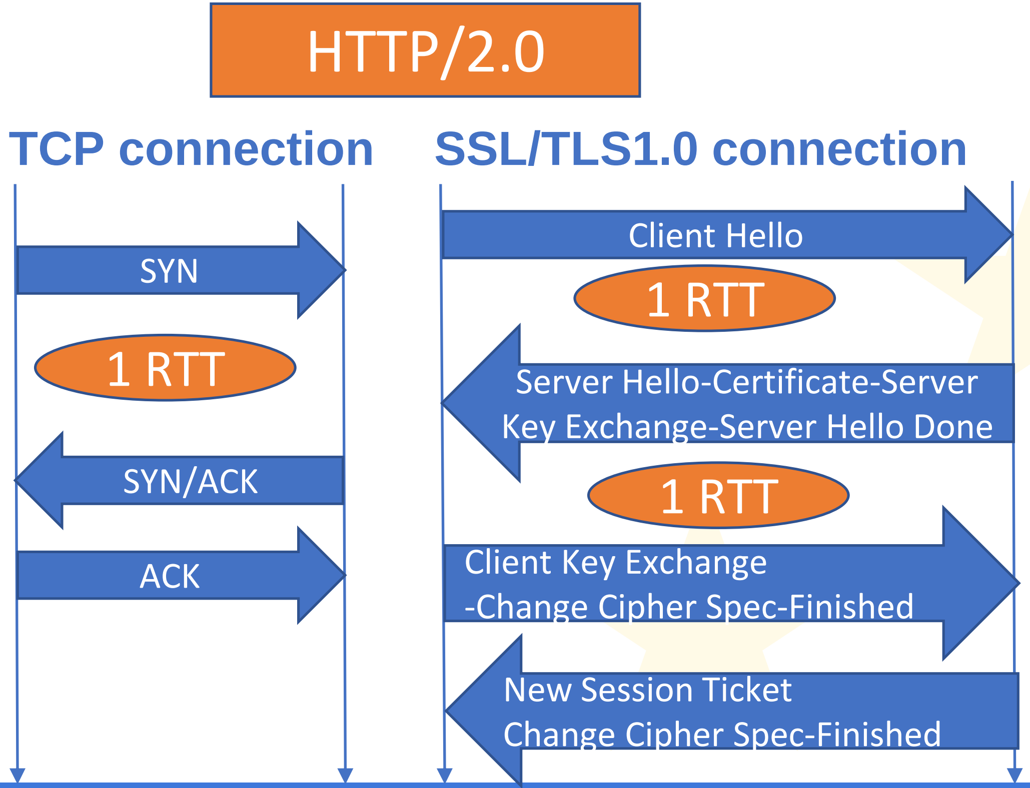
Filter “ssl” and check the each TLS packet

- Check packet #6 and expand Client Key Exchange



Client send Encrypted PreMaster Secret after negotiated with Server

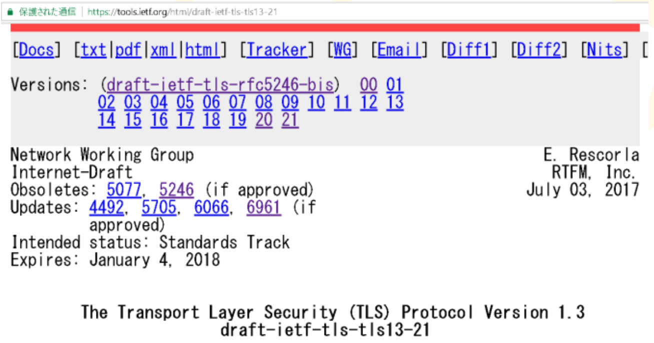
TLS1.0/1.2 needs 2 RTT at the first connection



- Old TLS needs 2 RTT at the fist connection
- It is not use for QUIC 1RTT connection
- Another way to negotiate and install session key….

TLS1.3 Internet Draft 21

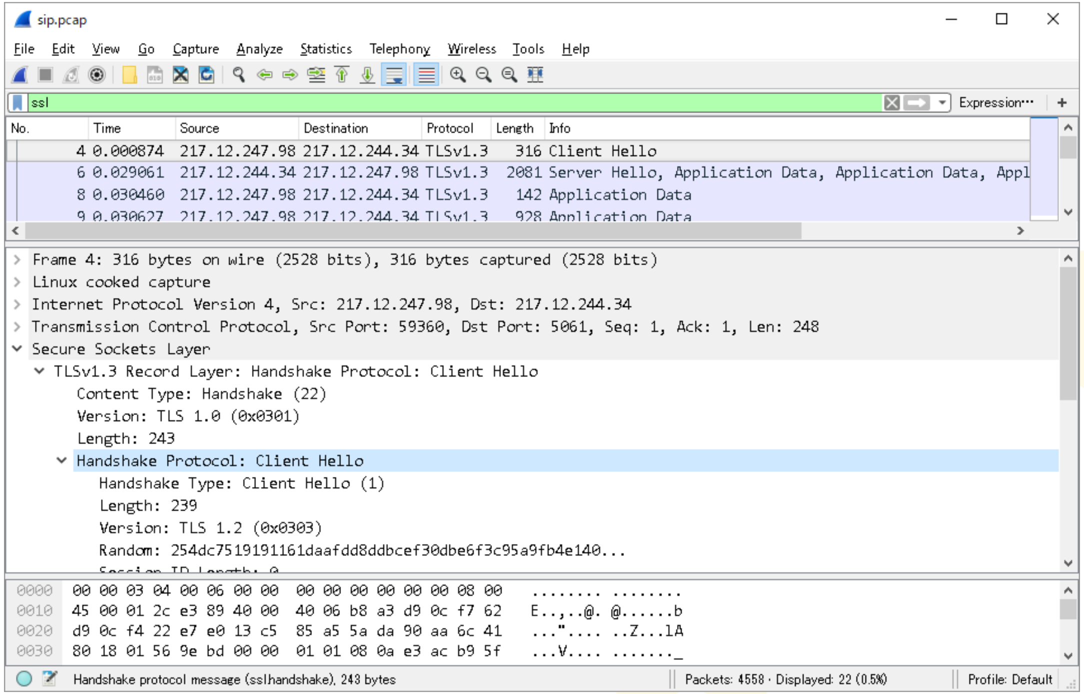
- New TLS protocol since 2014 now Internet-Drafts
  https://tools.ietf.org/html/draft-ietf-tls-tls13-21
- Stronger (few cleartext) and Faster (few packet)
- New encryption / authentication
- No SessionID, No Ticket, use PSK
  No Change Cipher Spec,
  No Client Key Exchange,
- 1-RTT at first time, 0-RTT when we connect again

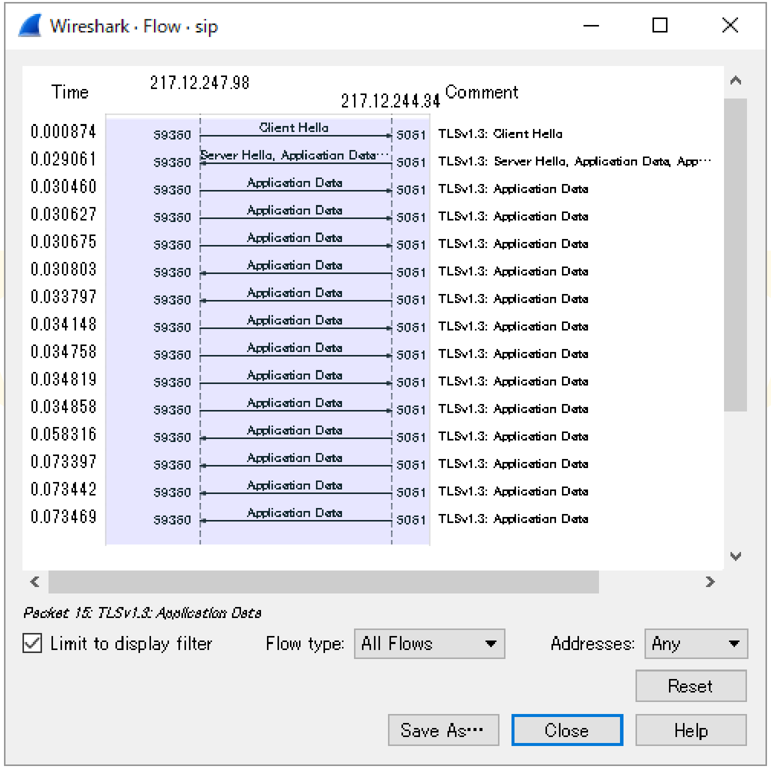

```
保護された通信 | https://tools.ietf.org/html/draft-ietf-tls-tls13-21

[Docs] [txt|pdf|xml|html] [Tracker] [WG] [Email] [Diff1] [Diff2] [Nits] [

Versions: (draft-ietf-tls-rfc5246-bis)  00 01
          02 03 04 05 06 07 08 09 10 11 12 13
          14 15 16 17 18 19 20 21

Network Working Group                                    E. Rescorla
Internet-Draft                                            RTFM, Inc.
Obsoletes: 5077, 5246 (if approved)                    July 03, 2017
Updates: 4492, 5705, 6066, 6961 (if
         approved)
Intended status: Standards Track
Expires: January 4, 2018


     The Transport Layer Security (TLS) Protocol Version 1.3
                   draft-ietf-tls-tls13-21
```

Sample trace of TLS1.3

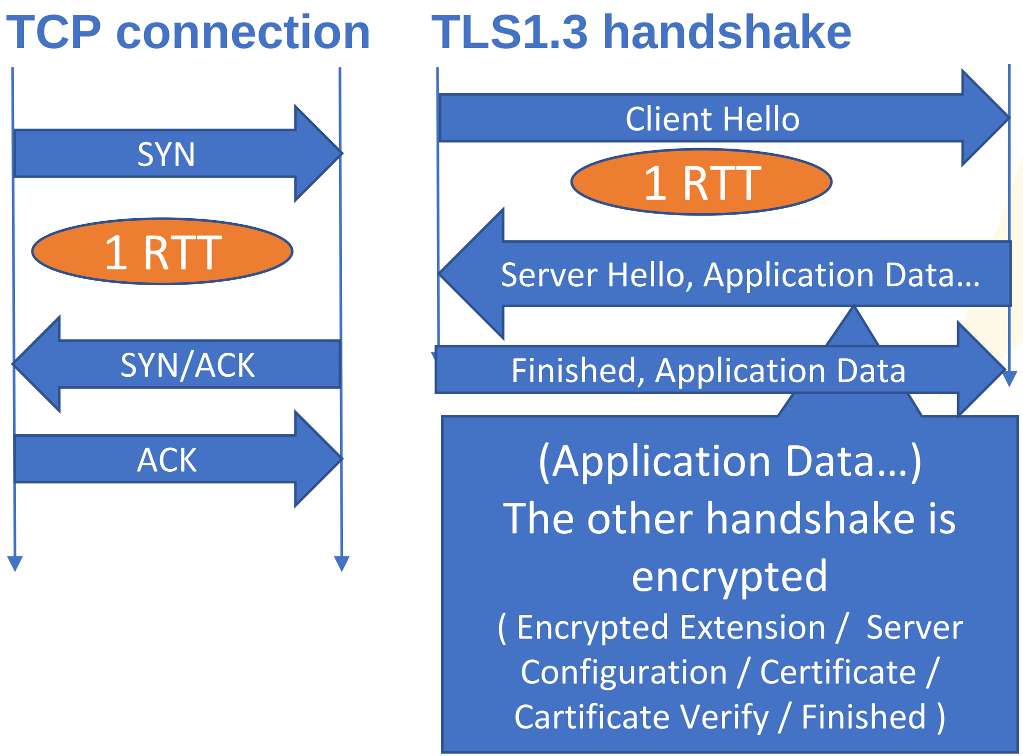
- Open sample trace file sip.pcap from Wireshark Wiki sip-tls-1.3-and-rtcp.zip SIP call over TLS 1.3 transport with enabled RTCP. Used openssl 1.1.1 prerelease version (https://wiki.wireshark.org/SampleCaptures)
- Open sip.pcap and filter ssl in Display Filter
- Statistics > Flow Graph and set Displayed Packet to see the 1-RTT full handshake of TLS1.3

Open sip.pcapng and filter ssl and create Flow Graph





TLS1.3 1-RTT handshake



There are no Cleint Key Exchange, no Change Cipher Spec packet, and the encryption starts after Server Hello

The other handshake is encrypted using PSK (Pre Shard Key).

Client send Application data after receiving Server packet

It needs just 1 Round trip time from Client side

Client Hello

(matches Client Key Exchange, Change Cipher Spec)

Extension: psk_key_exchange_modes (len=2)

Type: psk_key_exchange_modes (45) Length: 2

PSK Key Exchange Modes Length: 1

PSK Key Exchange Mode: PSK with (EC)DHE key establishment (psk_dhe_ke) (1)

Set PSK Key Exchange Mode

Extension: key_share (len=71)

Type: key_share (40) Length: 71

Key Share extension

Client Key Share Length: 69

Key Share Entry: Group: secp256r1, Key Exchange length: 65

Set Key Share settings

Group: secp256r1 (23)

Key Exchange Length: 65

Key Exchange: 04f145e0e15072f4983d04be08c7886c598af98607204dd0…

Send Key Exchange Data

Extension: certificate_authorities (len=40)

Type: certificate_authorities (47) Length: 40

Distinguished Names Length: 38

Distinguished Names (38 bytes)

Send certificate authorities

```
Secure Sockets Layer
  TLSv1.3 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 243
    Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 239
      Version: TLS 1.2 (0x0303)
      Random: 254dc7519191161daafdd8ddbcef30dbe6f3c95a9fb4e140...
      Session ID Length: 0
      Cipher Suites Length: 12
      Cipher Suites (6 suites)
      Compression Methods Length: 1
      Compression Methods (1 method)
        Compression Method: null (0)
      Extensions Length: 186
      Extension: ec_point_formats (len=4)
      Extension: supported_groups (len=4)
      Extension: SessionTicket TLS (len=0)
      Extension: signature_algorithms (len=22)
      Extension: encrypt_then_mac (len=0)
      Extension: extended_master_secret (len=0)
      Extension: supported_versions (len=3)
      Extension: psk_key_exchange_modes (len=2)
        Type: psk_key_exchange_modes (45)
        Length: 2
        PSK Key Exchange Modes Length: 1
        PSK Key Exchange Mode: PSK with (EC)DHE key establishment (psk_dhe_ke) (1)
      Extension: key_share (len=71)
        Type: key_share (40)
        Length: 71
        Key Share extension
          Client Key Share Length: 69
          Key Share Entry: Group: secp256r1, Key Exchange length: 65
            Group: secp256r1 (23)
            Key Exchange Length: 65
            Key Exchange: 04f145e0e15072f4983d04be08c7886c598af98607204dd0...
      Extension: certificate_authorities (len=40)
```

Send Client Nonce

Server Hello
(matches former Change Cipher Spec)

Handshake Protocol: Server Hello

Handshake Type: Server Hello (2)

Length: 111

Version: TLS 1.3 (draft 21) (0x7f15)

Random: 8f3a63a080b3c1ae2b3192c76574d4f28afdb1f123a68f81...

Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)

Extensions Length: 73

Extension: key_share (len=69)

Type: key_share (40)

Length: 69

Key Share extension

Key Share Entry: Group: secp256r1, Key Exchange length: 65

Group: secp256r1 (23)

Key Exchange Length: 65

Key Exchange: 04110e96ae58d23b968ebb7fd9075d83348733a622013785...

TLSv1.3 Record Layer: Application Data Protocol: sip.tcp

Determined Auth/Encryption

Set Key Share settings

The others are encrypted

Send Key Exchange Data

Send Server Nonce

```
Secure Sockets Layer
  TLSv1.3 Record Layer: Handshake Protocol: Server Hello
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 115
    Handshake Protocol: Server Hello
      Handshake Type: Server Hello (2)
      Length: 111
      Version: TLS 1.3 (draft 21) (0x7f15)
      Random: 8f3a63a080b3c1ae2b3192c76574d4f28afdb1f123a68f81...
      Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
      Extensions Length: 73
      Extension: key_share (len=69)
        Type: key_share (40)
        Length: 69
        Key Share extension
          Key Share Entry: Group: secp256r1, Key Exchange length: 65
            Group: secp256r1 (23)
            Key Exchange Length: 65
            Key Exchange: 04110e96ae58d23b968ebb7fd9075d83348733a622
  TLSv1.3 Record Layer: Application Data Protocol: sip.tcp
  TLSv1.3 Record Layer: Application Data Protocol: sip.tcp
  TLSv1.3 Record Layer: Application Data Protocol: sip.tcp
  TLSv1.3 Record Layer: Application Data Protocol: sip.tcp
```

TLS1.3 in IETF QUIC

- Let's go back to quic_ietf_draft05_ngtcp2.pcapng
- Check #1 packet of Client Initial (including Client Hello)
  Extension: quic_transport_parameters
  Extension: psk_key_exchange_modes
  Extension: key_share
- Check #2 packet of Server Cleartext (including Server Hello)
  Extension: key_share
- #3 (Server Cleartext) and #4(Client Cleartext) is encrypted with application data (http-over-tls)

0x04	Server Cleartext	Server sends this type packet as cryptographic handshake message and ACK that contains server chosen connection ID and randomized packet number with with STREAM, PADDING, ACK. (Long Header)
0x05	Client Cleartext	Client sends this type packet as the receipt of Server Cleartext message, Client Cleartext contains Server selected connection ID and incremented packet number of Client Initial with STREAM, PADDING, ACK. (Long Header)

Client Hello/ Server Hello of IQUIC

```
Secure Sockets Layer
  TLSv1.3 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 269
    Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 265
      Version: TLS 1.2 (0x0303)
      Random: a29b79ac62f9f87056f6322460f9cd7d116ec034526ff42d...
      Session ID Length: 0
      Cipher Suites Length: 8
      Cipher Suites (4 suites)
      Compression Methods Length: 1
      Compression Methods (1 method)
      Extensions Length: 216
      Extension: quic_transports_parameters (len=40)
      Extension: server_name (len=14)
      Extension: ec_point_formats (len=4)
      Extension: supported_groups (len=4)
      Extension: SessionTicket TLS (len=0)
      Extension: signature_algorithms (len=22)
      Extension: application_layer_protocol_negotiati  (len=
      Extension: encrypt_then_mac (len=0)
      Extension: extended_master_secret (len=0)
      Extension: supported_versions (len=3)
      Extension: psk_key_exchange_modes (len=2)
        Type: psk_key_exchange_modes (45)
        Length: 2
        PSK Key Exchange Modes Length: 1
        PSK Key Exchange Mode: PSK with (EC)DHE key establishment (psk_dhe_ke) (1)
      Extension: key_share (len=7
        Type: key_share (40)
        Length: 71
        Key Share extension
          Client Key Share Length: 69
          Key Share Entry: Group: secp256r1, Key Exchange length: 65
```

Send Client Nonce

Set PSK Key Exchange Mode

Set Key Share settings

Send Key Exchange Data

```
Secure Sockets Layer
  TLSv1.3 Record Layer: Handshake Protocol: Server Hello
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 115
    Handshake Protocol: Server Hello
      Handshake Type: Server Hello (2)
      Length: 111
      Version: TLS 1.3 (draft 21) (0x7f15)
      Random: 2affcdc53d1a1eda315150c35fff9b4461f7022dcbaed072...
      Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
      Extensions Length: 73
      Extension: key_share (len=69)
        Type: key_share (40)
        Length: 69
        Key Share extension
          Key Share Entry: Group: secp256r1, Key Exchange length: 65
  TL  .3 Record Layer: Application Data Protocol: http-over-tls
  TLSv1.3 Record Layer: Application Data Protocol: http-over-tls
Hash: 48682463dc278f1c
```

Send Server Nonce

The others are encrypted

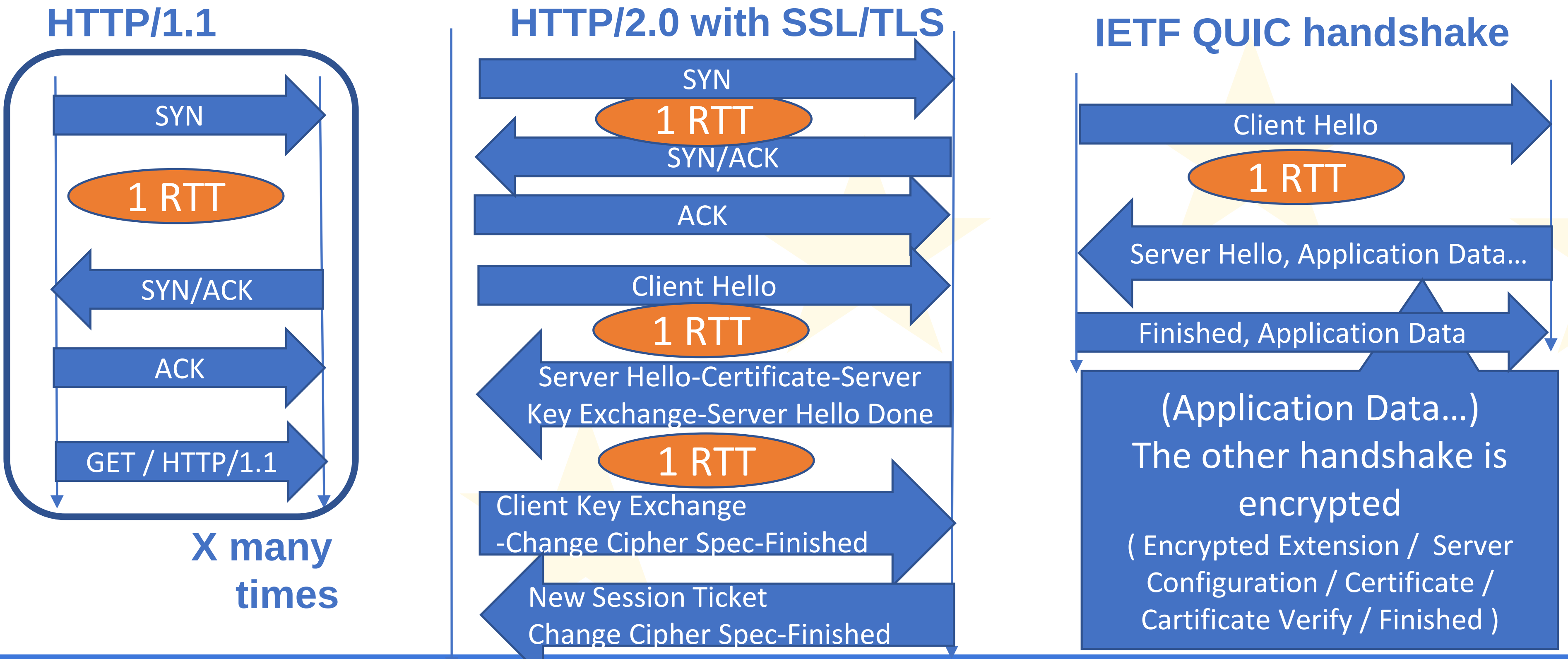
Short header transaction

- Check #5- packets with Short Header of IQUIC
- The short header can be used after the version and 1-RTT keys are negotiated.

```
5 0.038578488 12… 12… QUIC    83 SH, Protected Payload (KP0), PKN: 726976299, CID: 6536248117095700012
6 4.459336855 12… 12… QUIC    91 SH, Protected Payload (KP0), PKN: 558625389, CID: 6536248117095700012
7 4.459571939 12… 12… QUIC   106 SH, Protected Payload (KP0), PKN: 726976300, CID: 6536248117095700012
8 4.459603388 12… 12… QUIC    83 SH, Protected Payload (KP0), PKN: 726976301, CID: 6536248117095700012
9 4.459767458 12… 12… QUIC    83 SH, Protected Payload (KP0), PKN: 558625390, CID: 6536248117095700012

> Frame 5: 83 bytes on wire (664 bits), 83 bytes captured (664 bits) on interface 0
> Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 443, Dst Port: 39916
v QUIC (Quick UDP Internet Connections) IETF
    0... .... = Header Form: Short Header (0)
    .1.. .... = Connection ID Flag: True
    ..0. .... = Key Phase Bit: False
    ...0 0011 = Packet Type: 4 octet (3)
    Connection ID: 0x5ab56b082f4e162c
    Packet Number: 726976299
    Protected Payload: a48cd45b995a53917486448d31ac728254b21de8c7ad4c27...
```

- Transactions are independent and based on IP/UDP
- Next time Client try to use 0-RTT way.

Comparizon between HTTP/1.1 HTTP/2.0 and IETF QUIC



Use Wireshark !

Thank you very much !!

どうもありがとうございました！