



# SharkFest '17 Europe

## 20 QUIC Dissection

Using Wireshark to  
Understand QUIC  
Quickly

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

Megumi Takeshita

ikeriri network service

supplemental files  
<http://www.ikeriri.ne.jp/sharkfest>

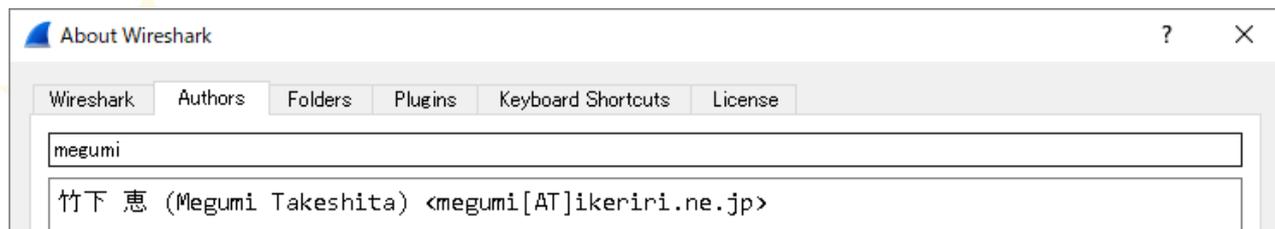
#sf17eu • Estoril, Portugal • 7-10 november 2017



# 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,

httpikeriri.pcapng

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-F> Expression...

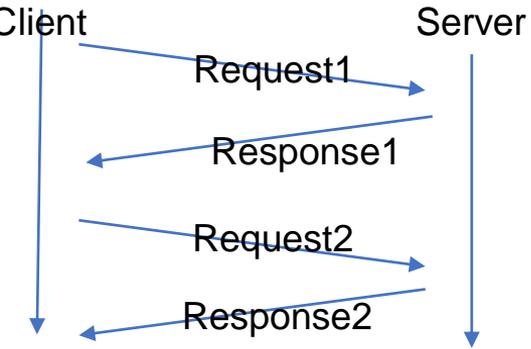
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.100.26	211.5.104.181	TCP	66	11726 → 80 [SYN] Seq=0 Win=64240 Le
2	0.091171	211.5.104.181	192.168.100.26	TCP	62	80 → 11726 [SYN, ACK] Seq=0 Ack=1 W
3	0.091260	192.168.100.26	211.5.104.181	TCP	54	11726 → 80 [ACK] Seq=1 Ack=1 Win=64
4	0.091533	192.168.100.26	211.5.104.181	HTTP	460	GET /sample.html HTTP/1.1
5	0.227400	211.5.104.181	192.168.100.26	HTTP	487	HTTP/1.1 200 OK (text/html)
6	0.267610	192.168.100.26	211.5.104.181	TCP	54	11726 → 80 [ACK] Seq=407 Ack=434 Wi
7	0.273132	192.168.100.26	211.5.104.181	HTTP	431	GET /favicon.ico HTTP/1.1
8	0.399501	211.5.104.181	192.168.100.26	HTTP	1341	HTTP/1.1 200 OK (image/x-icon)
9	0.447818	192.168.100.26	211.5.104.181	TCP	54	11726 → 80 [ACK] Seq=784 Ack=1721 W

> Frame 4: 460 bytes on wire (3680 bits), 460 bytes captured (3680 bits) on interface 0  
> Ethernet II, Src: Inventec\_2f:b9:75 (00:8c:fa:2f:b9:75), Dst: Modacom\_94:ea:bc (00:1d:93:94:ea:bc)  
> Internet Protocol Version 4, Src: 192.168.100.26, Dst: 211.5.104.181  
> Transmission Control Protocol, Src Port: 11726, Dst Port: 80, Seq: 1, Ack: 1, Len: 406  
▼ Hypertext Transfer Protocol  
 > GET /sample.html HTTP/1.1\r\n Host: www.ikeriri.ne.jp\r\n Connection: keep-alive\r\n User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.10\r\n Upgrade-Insecure-Requests: 1\r\n

- 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

Time	192.168.100.26	211.5.104.181	Comment
0.091533	11726	GET /sample.html HTTP/1.1	80 HTTP: GET /sample.html HTTP/1.1
0.227400	11726	HTTP/1.1 200 OK (text/html)	80 HTTP: HTTP/1.1 200 OK (text/html)
0.273132	11726	GET /favicon.ico HTTP/1.1	80 HTTP: GET /favicon.ico HTTP/1.1
0.399501	11726	HTTP/1.1 200 OK (image/x-icon)	80 HTTP: HTTP/1.1 200 OK (image/x-icon)

No.	Time	Source	Destination	Protocol	Length	Info
4	0.091533	192.168.100.26	211.5.104.181	HTTP	460	GET /sample.html HTTP/1.1
5	0.227400	211.5.104.181	192.168.100.26	HTTP	487	HTTP/1.1 200 OK (text/html)
7	0.273132	192.168.100.26	211.5.104.181	HTTP	431	GET /favicon.ico HTTP/1.1
8	0.399501	211.5.104.181	192.168.100.26	HTTP	1341	HTTP/1.1 200 OK (image/x-icon)

# 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 connection is separated by each other TCP connection, they work their own TCP rules without HTTP.



```
Wireshark - Follow HTTP Stream (tcp.stream eq 0) - httpikeriri
GET /sample.html HTTP/1.1
Host: www.ikeriri.ne.jp
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36
Upgrade-Insecure-Requests: 1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: ja,en-US;q=0.8,en;q=0.6

HTTP/1.1 200 OK
Cache-Control: no-cache
Content-Type: text/html
Content-Encoding: gzip
Vary: Accept-Encoding
Server: Microsoft-IIS/7.0
X-Powered-By: PHP/5.3.19
X-Powered-By: ASP.NET
Date: Wed, 11 Oct 2017 06:24:45 GMT
Content-Length: 183

<doctype html>
<html>
<head>
<title>sample</title>
</head>
<body>
<h1>homepage</h1>
</body>
</html>

GET /favicon.ico HTTP/1.1
Host: www.ikeriri.ne.jp
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36

2 client pkts, 2 server pkts, 3 sums.
Entire conversation (2845 bytes) Show and save data as ASCII
Find: Find Next
Filter Out This Stream Print Save as... Back Close Help
```





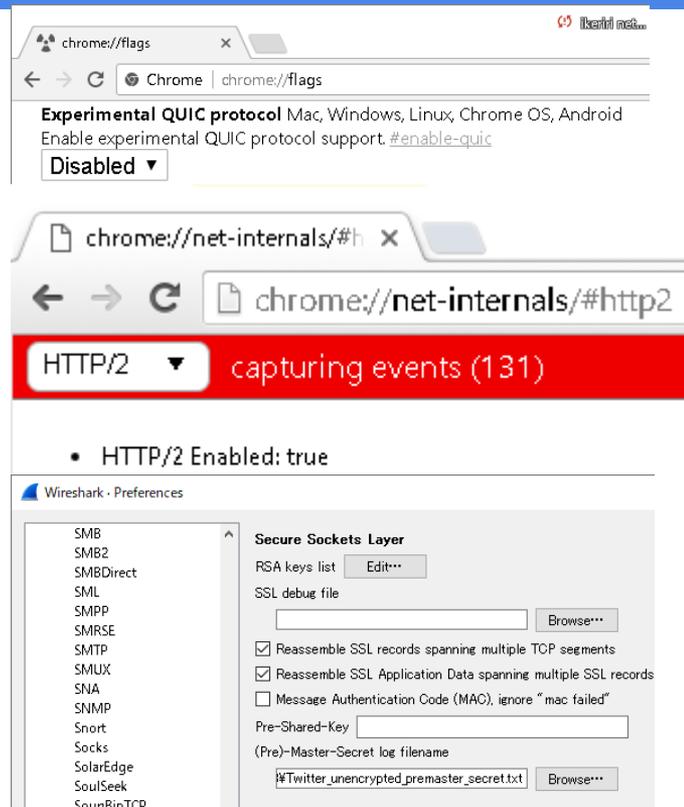
# Accelerate Web service

- 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](http://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



The image shows two screenshots. The top one is a Chrome browser window at `chrome://flags` with the 'Experimental QUIC protocol' flag set to 'Disabled'. The bottom one is a Chrome window at `chrome://net-internals/#http2` showing 'HTTP/2' as the selected protocol and 'capturing events (131)'. Below that is a Wireshark 'Preferences' dialog box, specifically the 'Secure Sockets Layer' section, where the '(Pre)-Master-Secret log filename' is set to `#Twitter_unencrypted_premaster_secret.txt`.



# 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

twitter.pcapng

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

http2.header

No.	Time	Source	Destination	Protocol	Length	Info
14	0.039605	10.0.0.13	104.244.42.67	HTTP2	793	HEADERS [ ... ]
23	0.154656	104.244.42.67	10.0.0.13	HTTP2	471	HEADERS [ ... ]

> Frame 14: 793 bytes on wire (6344 bits), 793 bytes captured (6344 bits) on interface 0

> Ethernet II, Src: AsustekC\_55:f4:56 (20:cf:30:55:f4:56), Dst: Fortinet\_b0:6a:9a (00:09:00:00:00:00)

> Internet Protocol Version 4, Src: 10.0.0.13, Dst: 104.244.42.67

> Transmission Control Protocol, Src Port: 27964, Dst Port: 443, Seq: 491, Ack: 3035, Len: 793

> Secure Sockets Layer

> HyperText Transfer Protocol 2

- Stream: HEADERS, Stream ID: 1, Length 701, GET /tpm/p?\_=1490016065091
- Length: 701
- Type: HEADERS (1)
- Flags: 0x25
- 0... = Reserved: 0x0
- .000 0000 0000 0000 0000 0000 0000 0001 = Stream Identifier: 1
- [Pad Length: 0]
- 1... = Exclusive: True
- .000 0000 0000 0000 0000 0000 0000 0000 = Stream Dependency: 0
- Weight: 109
- [Weight real: 110]
- Header Block Fragment: 82418f1d43a3d24c442e9f064a4b62e43d3f870084b958d3...
- [Header Length: 1150]
- [Header Count: 19]
- > Header: :method: GET

0080 70 6c 69 63 61 74 69 6f 6e 2f 6a 73 6f 6e 2c 20 application/json,  
0090 74 65 78 74 2f 6a 61 76 61 73 63 72 69 70 74 2c text/javascript,

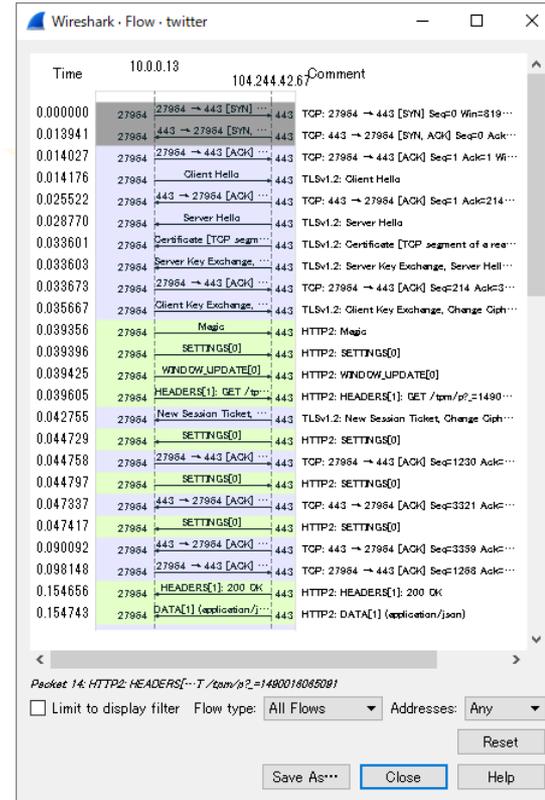
Frame (793 bytes) | Decrypted SSL (710 bytes) | Decompressed Header (1150 bytes)

Header Count (http2.header.count), 1 byte | Packets: 61 · Displayed: 8 (13.1%) | Profile: Default



# Connection process of HTTP/2.0

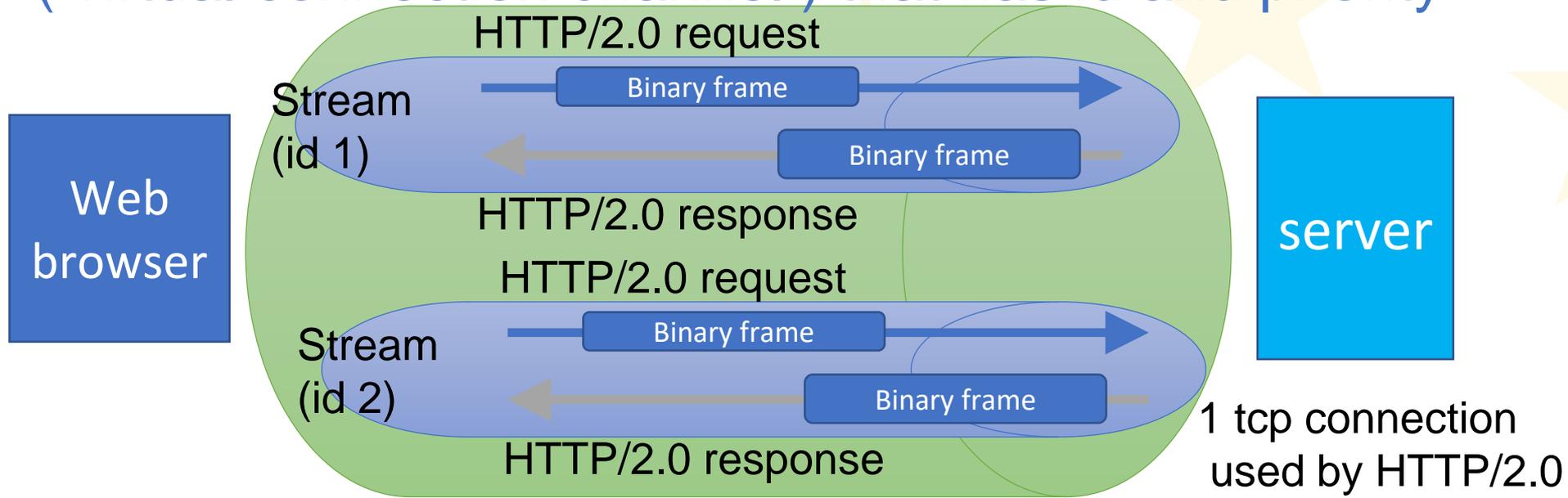
- Click Statistics > Flow Graph and check connection process of HTTP/2.0
- HTTP/2.0 needs TCP 3 way handshake that contains 1 RTT(round trip time) SYN-SYN/ACK-ACK from Client side
- HTTP/2.0 needs SSL/TLS connection that contains 2 RTT(round trip time) from Client side Client Hello/Server Hello-Certificate-Server 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

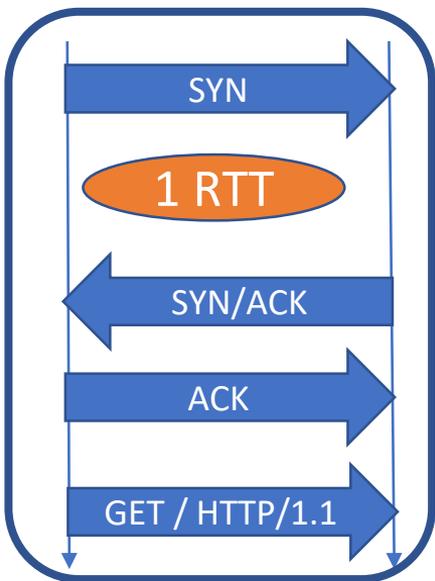




# Multiple HTTP connections (at the first time)

HTTP/1.1

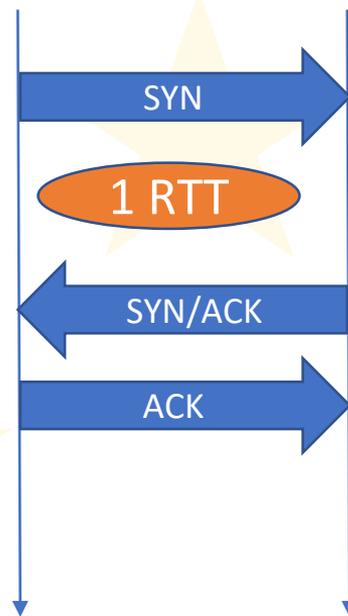
TCP connection



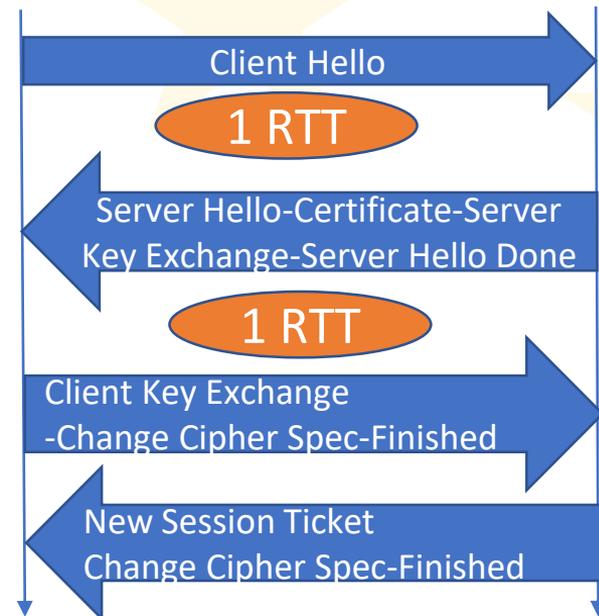
X many times

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

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.0.0.13	172.217.26.100	UDP	175	50270 → 443 Len=133
2	0.012014	kix05s01-in-f4.1e100.net	10.0.0.13	UDP	469	443 → 50270 Len=427
3	0.019206	10.0.0.13	kix05s01-in-f4.1e100.net	UDP	94	50270 → 443 Len=52
4	0.031990	kix05s01-in-f4.1e100.net	10.0.0.13	UDP	75	443 → 50270 Len=33
5	1.219255	10.0.0.13	kix05s01-in-f4.1e100.net	UDP	347	50270 → 443 Len=305
6	1.259549	kix05s01-in-f4.1e100.net	10.0.0.13	UDP	69	443 → 50270 Len=27
7	1.445828	kix05s01-in-f4.1e100.net	10.0.0.13	UDP	217	443 → 50270 Len=175
8	1.447768	10.0.0.13	kix05s01-in-f4.1e100.net	UDP	100	50270 → 443 Len=58
9	1.448612	10.0.0.13	kix05s01-in-f4.1e100.net	UDP	115	50270 → 443 Len=73
10	1.460056	kix05s01-in-f4.1e100.net	10.0.0.13	UDP	72	443 → 50270 Len=30

chrome://net-internals/# quic

Chrome | chrome://net-internals/#quic

**capturing events (4961)**

Capture  
Export  
Import

- QUIC Enabled: true
- Origins To Force QUIC On:
- Connection options:
- Load Server Info Timeout Multiplier: 0.25

- 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	<a href="#">8126651583460386461</a>	0	None	1	7	0	7	true
clients4.google.com:443		QUIC_VERSION_35	172.217.26.46:443	<a href="#">7635153523791641412</a>	0	None	1	8	0	7	true
clients5.google.com:443		QUIC_VERSION_35	172.217.26.46:443	<a href="#">3964807834305814218</a>	0	None	0	3	0	3	true
fonts.gstatic.com:443	ssl.gstatic.com:443	QUIC_VERSION_35	172.217.26.35:443	<a href="#">3002020916089671232</a>	0	None	0	3	0	3	true
lh3.googleusercontent.com:443		QUIC_VERSION_35	216.58.197.129:443	<a href="#">1788128156380661810</a>	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:9a)
> Internet Protocol Version 4, Src: 10.0.0.13, Dst: 172.217.26.100
> User Datagram Protocol, Src Port: 50270, Dst Port: 443
> 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)

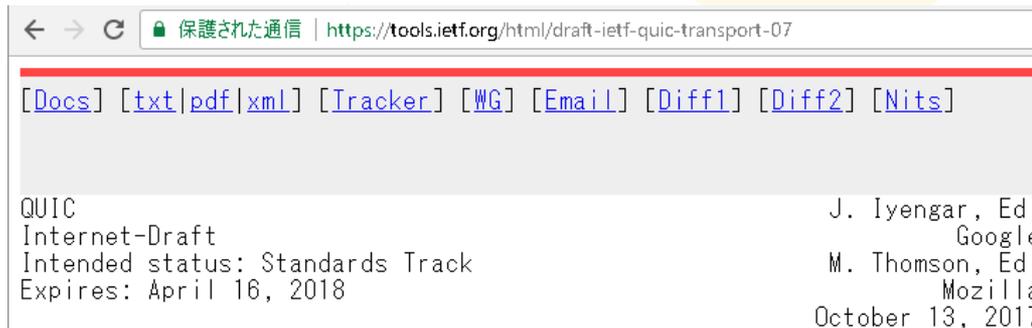
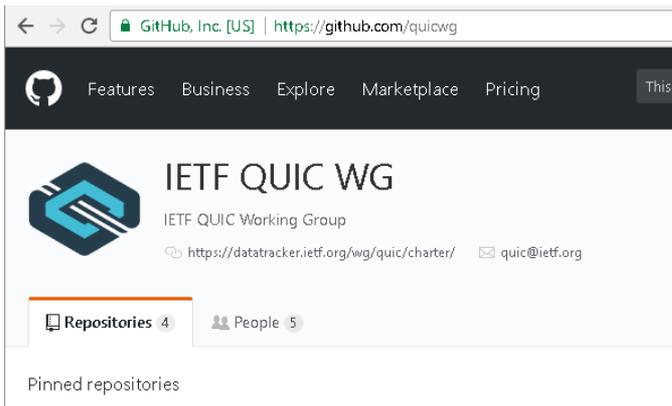
- Now IETF standardize IETF QUIC (a.k.a. IQUIC)
- IQUIC also provides a monolithic mechanism of TCP reliable transport + SSL/TLS1.3 authentication and encryption + HTTP/2 multiplexing and compression
- Now Internet-Draft (October 13, 2017)  
<https://tools.ietf.org/html/draft-ietf-quic-transport-07>
- 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>



QUIC: A UDP-Based Multiplexed and Secure Transport  
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  
Internet-Draft  
Intended status: Standards Track  
Expires: April 16, 2018

M. Thomson, Ed.  
Mozilla  
S. Turner, Ed.  
sn3rd  
October 13, 2017

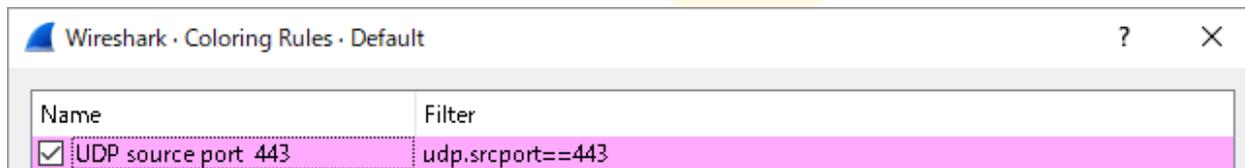
Using Transport Layer Security (TLS) to Secure QUIC  
draft-ietf-quic-tls-07

## 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



- 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







# 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 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

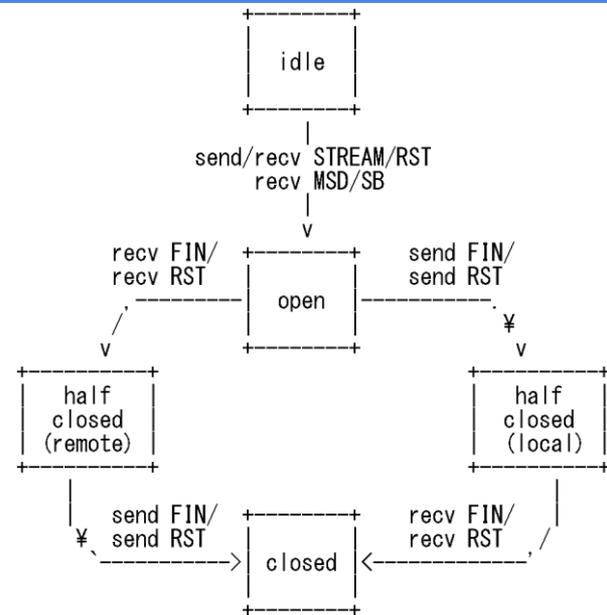
Time	Source	Destine	Protocol	Length	Connection ID	Packet Number	Info
1 0.00...	127...	127...	QUIC	1294	0x8ee4cfaf7e9f5d9c	558625387	LH, Client Initial, PKN: 558625387, CID: 0x8ee4cfaf7e9f5d9c
2 0.03...	127...	127...	QUIC	1283	0x5ab56b082f4e162c	726976297	LH, Server Cleartext, PKN: 726976297, CID: 0x5ab56b082f4e162c
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)

- IQIC 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 )
- IQIC stream mechanism is almost the same as HTTP/2.0(also as TCP)
- Stream change the state, Many streams in a UDP connection



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: a48cd45b995a53917486448d31ac728254b21de8c7ed4c27
```

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 part 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 )

Wireshark - Preferences

Secure Sockets Layer

RSA keys list Edit...

SSL debug file

Reassemble SSL records spanning multiple TCP segments

Reassemble SSL Application Data spanning multiple SSL records

Message Authentication Code (MAC), ignore "mac failed"

Pre-Shared-Key

(Pre)-Master-Secret log filename

File: C:\Users\megumi\Desktop\tls10ikeriri.txt

Source	Destination	Protocol	Length	Info
192.168.100.122	192.168.100.200	TCP	66	10189 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK...
192.168.100.200	192.168.100.122	TCP	66	443 → 10189 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 SA...
192.168.100.122	192.168.100.200	TCP	54	10189 → 443 [ACK] Seq=1 Ack=1 Win=262144 Len=0
192.168.100.122	192.168.100.200	TLSv1	210	Client Hello
192.168.100.200	192.168.100.122	TCP	60	443 → 10189 [ACK] Seq=1 Ack=157 Win=6912 Len=0
192.168.100.200	192.168.100.122	TLSv1	567	Server Hello, Certificate, Server Hello Done
192.168.100.122	192.168.100.200	TCP	54	10189 → 443 [ACK] Seq=157 Ack=514 Win=261376 Len=0
192.168.100.122	192.168.100.200	TLSv1	252	Client Key Exchange, Change Cipher Spec, Encrypted Handshake ...
192.168.100.200	192.168.100.122	TLSv1	320	New Session Ticket, Change Cipher Spec, Encrypted Handshake M...
192.168.100.122	192.168.100.200	TCP	54	10189 → 443 [ACK] Seq=355 Ack=780 Win=261120 Len=0
192.168.100.122	192.168.100.200	TLSv1	400	Application Data, Application Data
192.168.100.200	192.168.100.122	TLSv1	666	Application Data, Application Data, Application Data
192.168.100.122	192.168.100.200	TCP	54	10189 → 443 [ACK] Seq=701 Ack=1392 Win=260608 Len=0

Wireshark - Preferences

Secure Sockets Layer

RSA keys list Edit...

SSL debug file

Reassemble SSL records spanning multiple TCP segments

Reassemble SSL Application Data spanning multiple SSL records

Message Authentication Code (MAC), ignore "mac failed"

Pre-Shared-Key

(Pre)-Master-Secret log filename

File: C:\Users\megumi\Desktop\tls10ikeriri.txt

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.100.122	192.168.100.200	TCP	66	10189 → 443 [SYN] Seq=0 Win=65535 Len=0
2	0.000487	192.168.100.200	192.168.100.122	TCP	66	443 → 10189 [SYN, ACK] Seq=0 Ack=1 Win=5
3	0.000544	192.168.100.122	192.168.100.200	TCP	54	10189 → 443 [ACK] Seq=1 Ack=1 Win=262144
4	0.000790	192.168.100.122	192.168.100.200	TLSv1	210	Client Hello
5	0.001227	192.168.100.200	192.168.100.122	TCP	60	443 → 10189 [ACK] Seq=1 Ack=157 Win=6912
6	0.003810	192.168.100.200	192.168.100.122	TLSv1	567	Server Hello, Certificate, Server Hello
7	0.003857	192.168.100.122	192.168.100.200	TCP	54	10189 → 443 [ACK] Seq=157 Ack=514 Win=26
8	0.006733	192.168.100.122	192.168.100.200	TLSv1	252	Client Key Exchange, Change Cipher Spec,
9	0.037486	192.168.100.200	192.168.100.122	TLSv1	320	New Session Ticket, Change Cipher Spec,
10	0.037572	192.168.100.122	192.168.100.200	TCP	54	10189 → 443 [ACK] Seq=355 Ack=780 Win=26
11	0.038436	192.168.100.122	192.168.100.200	HTTP	400	GET / HTTP/1.1
12	0.042742	192.168.100.200	192.168.100.122	HTTP	666	HTTP/1.1 200 OK (text/html)
13	0.042801	192.168.100.122	192.168.100.200	TCP	54	10189 → 443 [ACK] Seq=701 Ack=1392 Win=2



Wireshark - Preferences

Secure Sockets Layer

RSA keys list Edit...

SSL debug file

Reassemble SSL records spanning multiple TCP segments

Reassemble SSL Application Data spanning multiple SSL records

Message Authentication Code (MAC), ignore "mac failed"

Pre-Shared-Key

(Pre)-Master-Secret log filename

File: C:\Users\megumi\Desktop\tls10ikeriri.txt

Wireshark - Preferences

Secure Sockets Layer

RSA keys list Edit...

SSL debug file

Reassemble SSL records spanning multiple TCP segments

Reassemble SSL Application Data spanning multiple SSL records

Message Authentication Code (MAC), ignore "mac failed"

Pre-Shared-Key

(Pre)-Master-Secret log filename

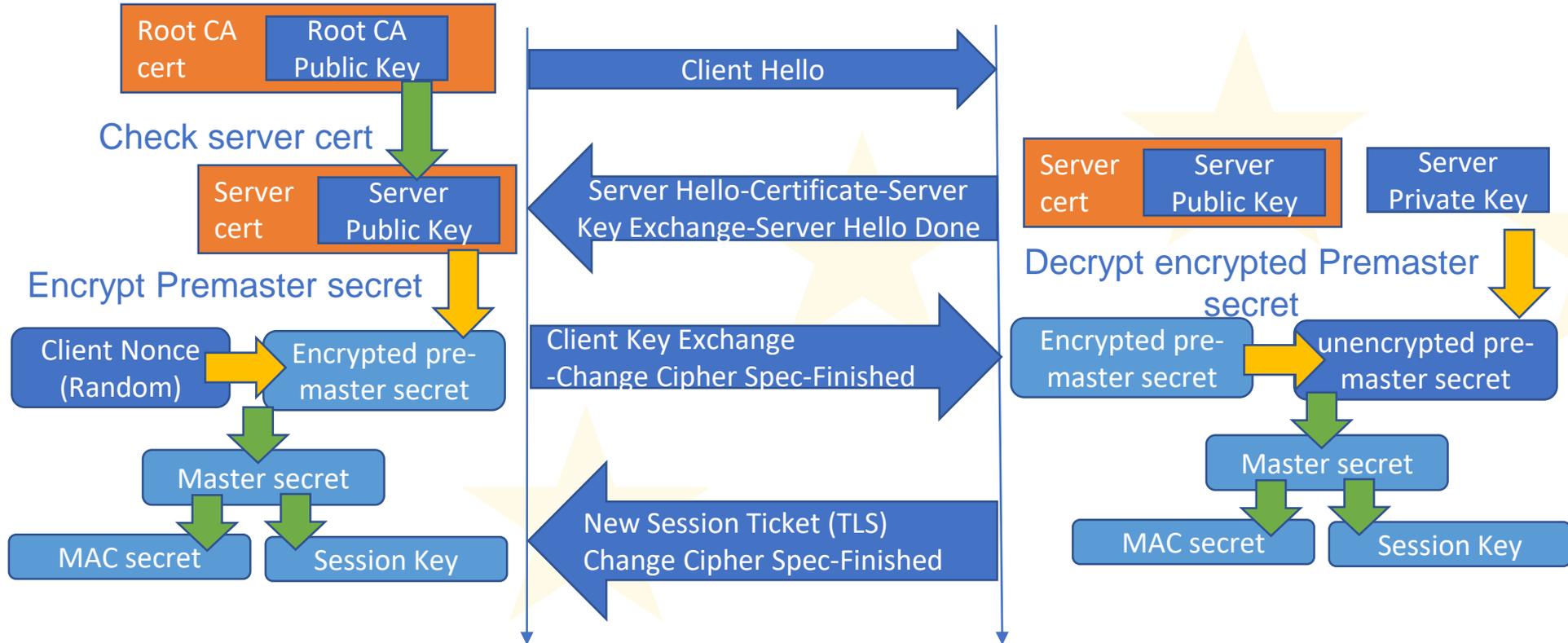
File: C:\Users\megumi\Desktop\tls10ikeriri.txt

SSL Decrypt

IP address	Port	Protocol	Key File	Password
192.168.100.200	443	http	C:/Users/megumi/Desktop/tls10ikeriri.txt	



# Key creation process of TLS1.0





# Filter "ssl" and check the each TLS packet

- Check packet #6 and expand Client Key Exchange

No.	Time	Source	Destination	Protocol	Length	Info
4	0.000790	192.168.100.122	192.168.100.200	TLSv1	210	Client Hello
6	0.003810	192.168.100.200	192.168.100.122	TLSv1	567	Server Hello, Certificate, Server Hello
8	0.006733	192.168.100.122	192.168.100.200	TLSv1	252	Client Key Exchange, Change Cipher Spec
9	0.037486	192.168.100.200	192.168.100.122	TLSv1	320	New Session Ticket, Change Cipher Spec,
11	0.038436	192.168.100.122	192.168.100.200	HTTP	400	GET / HTTP/1.1
12	0.042742	192.168.100.200	192.168.100.122	HTTP	666	HTTP/1.1 200 OK (text/html)

> Ethernet II, Src: Inventec\_2f:9c:c6 (00:8c:fa:2f:9c:c6), Dst: AvalueTe\_02:0b:79 (00:04:5f:02:0b:79)  
> Internet Protocol Version 4, Src: 192.168.100.122, Dst: 192.168.100.200  
> Transmission Control Protocol, Src Port: 10189, Dst Port: 443, Seq: 157, Ack: 514, Len: 198  
v Secure Sockets Layer  
  v TLSv1 Record Layer: Handshake Protocol: Client Key Exchange  
    Content Type: Handshake (22)  
    Version: TLS 1.0 (0x0301)  
    Length: 134  
      v Handshake Protocol: Client Key Exchange  
        Handshake Type: Client Key Exchange (16)  
        Length: 130  
          > RSA Encrypted PreMaster Secret  
      > TLSv1 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec  
      > TLSv1 Record Layer: Handshake Protocol: Finished

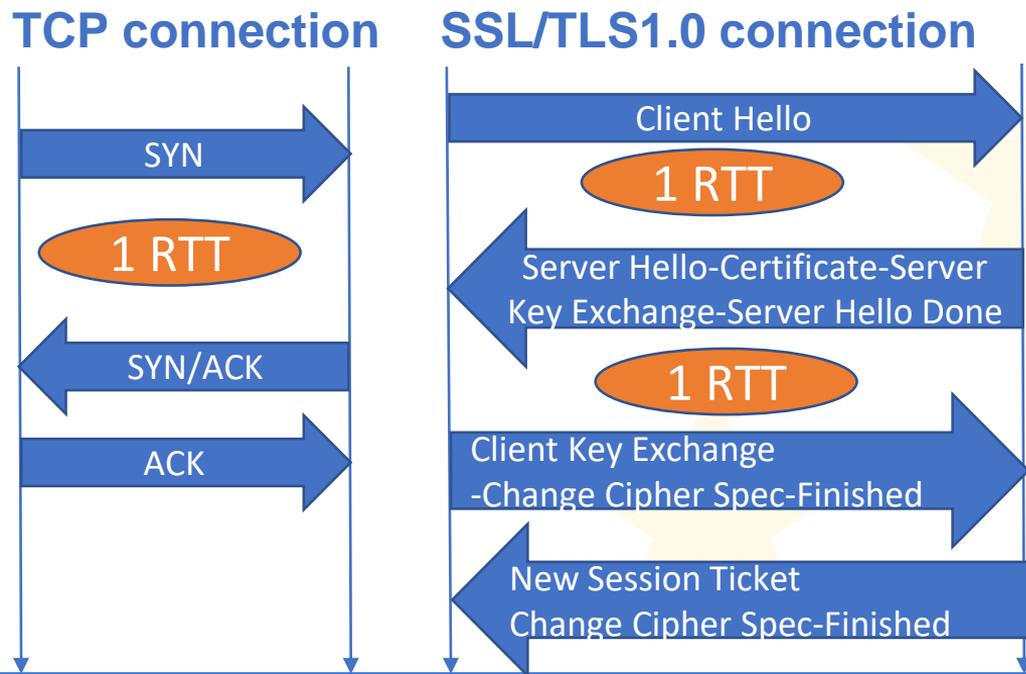
Client send  
Encrypted  
PreMaster  
Secret after  
negotiated  
with Server





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

## HTTP/2.0



- Old TLS needs 2 RTT at the first 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

```
保留された通信 | 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](https://wiki.wireshark.org/SampleCaptures) SIP call over TLS 1.3 transport with enabled RTCP. Used [openssl 1.1.1 prerelease version](https://wiki.wireshark.org/SampleCaptures) (<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

The screenshot shows the Wireshark interface with the filter 'ssl' applied. The packet list pane shows several packets, with packet 4 selected. The details pane shows the structure of the TLSv1.3 Record Layer: Handshake Protocol: Client Hello.

No.	Time	Source	Destination	Protocol	Length	Info
4	0.000874	217.12.247.98	217.12.244.34	TLSv1.3	316	Client Hello
6	0.029061	217.12.244.34	217.12.247.98	TLSv1.3	2081	Server Hello, Application Data, Application Data, Appl...
8	0.030460	217.12.247.98	217.12.244.34	TLSv1.3	142	Application Data
9	0.030627	217.12.247.98	217.12.244.34	TLSv1.3	928	Application Data

Frame 4: 316 bytes on wire (2528 bits), 316 bytes captured (2528 bits) on Linux cooked capture  
> Internet Protocol Version 4, Src: 217.12.247.98, Dst: 217.12.244.34  
> Transmission Control Protocol, Src Port: 59360, Dst Port: 5061, Seq: 1, Ack: 1, Len: 248  
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: 254dc7519191161daafd8ddbcef30dbe6f3c95a9fb4e140...  
      Session ID Length: 0

The screenshot shows the Wireshark Flow Graph for the selected packet. It displays a sequence of packets between 217.12.247.98 and 217.12.244.34. The flow starts with a Client Hello from 217.12.247.98 to 217.12.244.34, followed by a Server Hello and Application Data from 217.12.244.34 to 217.12.247.98, and then a series of Application Data packets from 217.12.247.98 to 217.12.244.34.

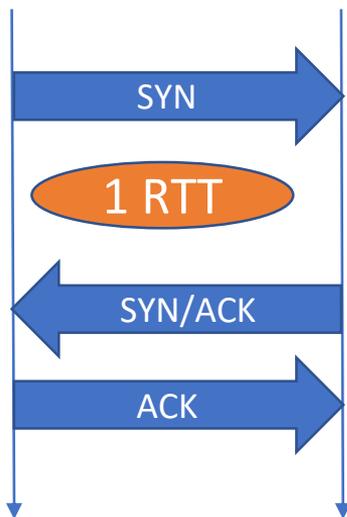
Time	217.12.247.98	217.12.244.34	Comment
0.000874	98380	5061	Client Hello
0.029061	5061	98380	Server Hello, Application Data...
0.030460	98380	5061	Application Data
0.030627	98380	5061	Application Data
0.030675	98380	5061	Application Data
0.030803	98380	5061	Application Data
0.033797	98380	5061	Application Data
0.034148	98380	5061	Application Data
0.034758	98380	5061	Application Data
0.034819	98380	5061	Application Data
0.034858	98380	5061	Application Data
0.058316	98380	5061	Application Data
0.073397	98380	5061	Application Data
0.073442	98380	5061	Application Data
0.073469	98380	5061	Application Data

Packet 15: TLSv1.3: Application Data  
 Limit to display filter    Flow type: All Flows    Addresses: Any  
Reset  
Save As...    Close    Help

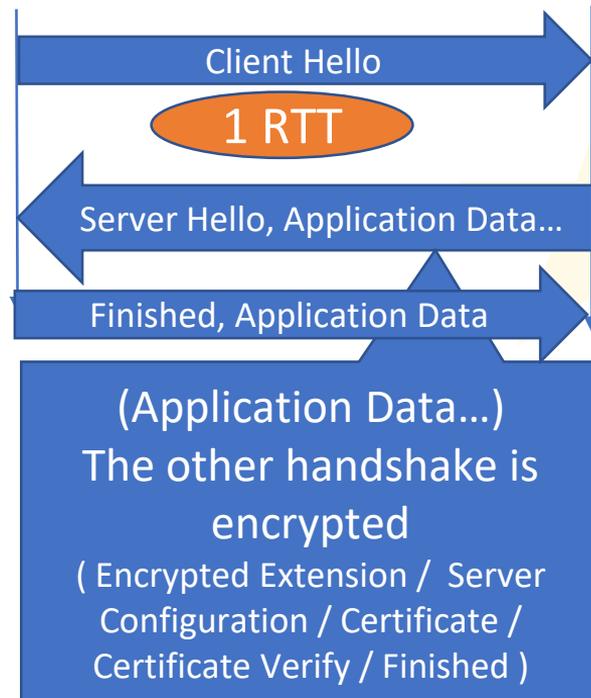


# TLS1.3 1-RTT handshake

## TCP connection



## TLS1.3 handshake



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

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

Client send Application data after receiving Server packet

It needs just 1 Round trip time from Client side





# Client Hello

(contains former 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

( contains 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

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

Send Server Nonce





# 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_negotiation (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=71)
    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
  TLSv1.3 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 IQIC
- 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...
```

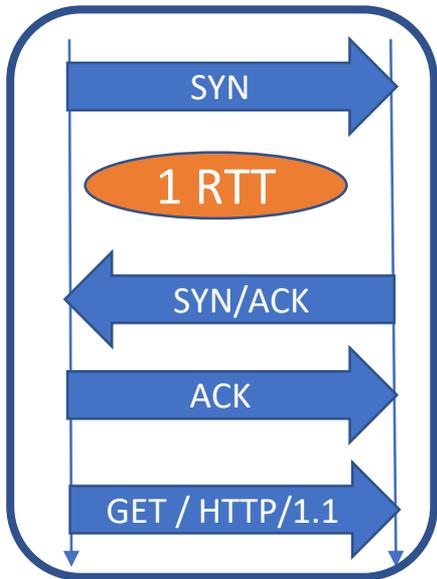
- Transactions are independent and based on IP/UDP
- Next time Client try to use 0-RTT way.





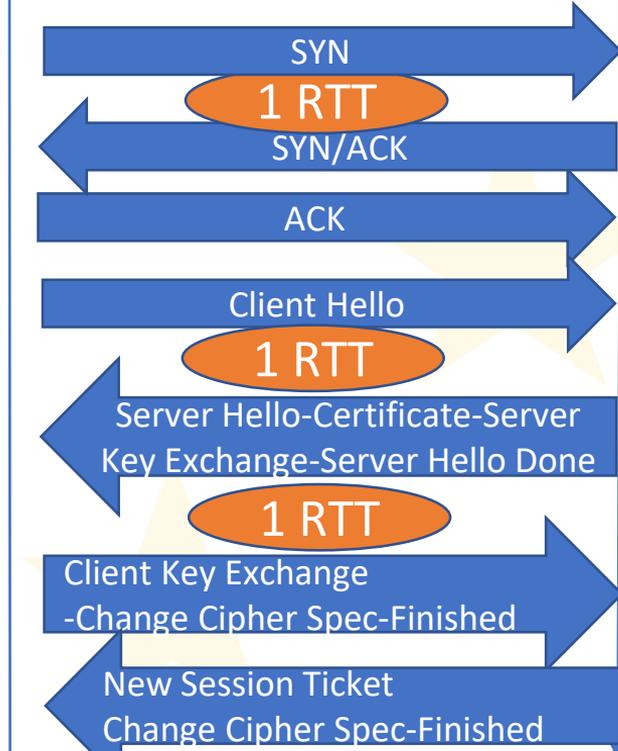
# Comparizon between HTTP/1.1 HTTP/2.0 and IETF QUIC

## HTTP/1.1

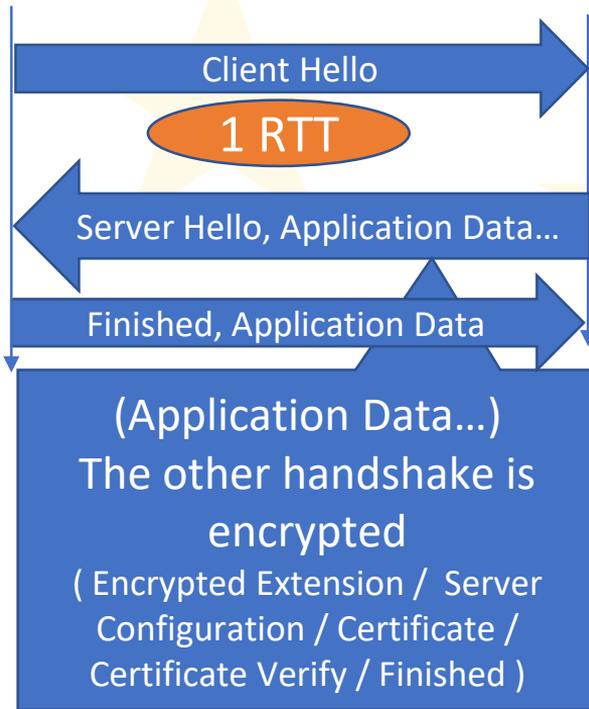


**X many times**

## HTTP/2.0 with SSL/TLS



## IETF QUIC handshake





# Use Wireshark

Thank you very much !!

どうもありがとうございました！

