QUICの話
Technical Overview of QUIC

大津 繁樹
Shigeki Ohtsu

Internet Initiative Japan

2014 Nov. 3

HTTP/2 Conference Japan 2014

Current Issues in TCP

- TCP Head of Line Blocking.
- Handshake cost by 3way handshake. — TCP Fast Open
- Small initcwnd and slow start. — initCWND10
- Large backoff caused by packet loss. — TCP cubic
- Increase socket buffer. — TCP_NOT_SENT_LOWAT
- NAT timeout and IP roaming
- TCP Buffer Bloat — Random packet drop in router

Need to upgrade OS on both end and middle box. It's a long way.

Current Issues in TLS

- Handshake cost in TLS negotiation
- Blocking in Change Cipher Spec
- 256bytes limit in ClientHello by Load Balancer LB bug.
- Server Certificate Chain gets large to send.
- Renegotiation and Resumption is not optimized.

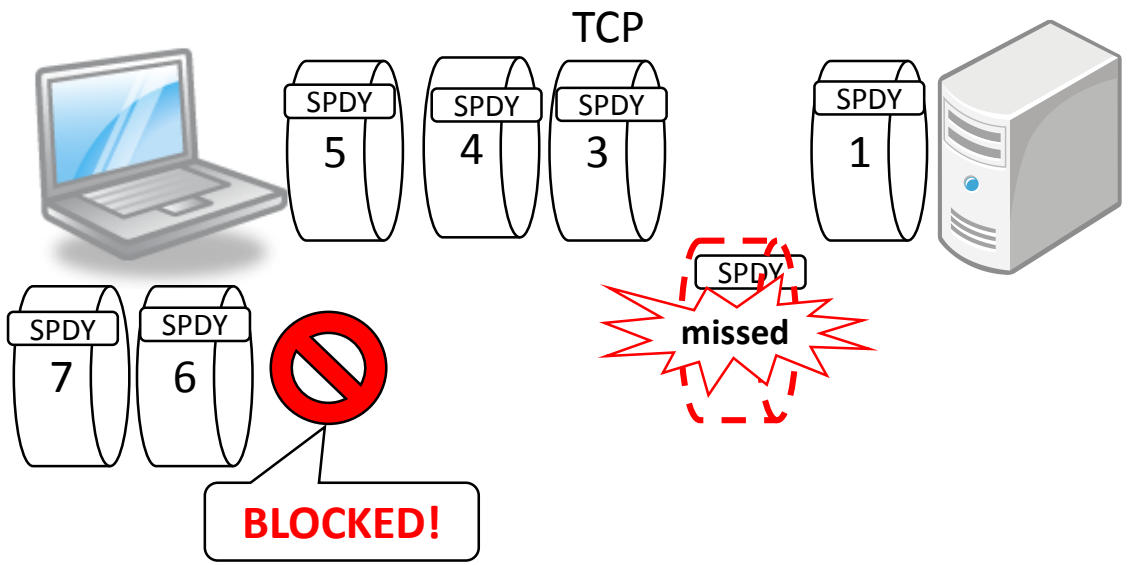
TLS False Start

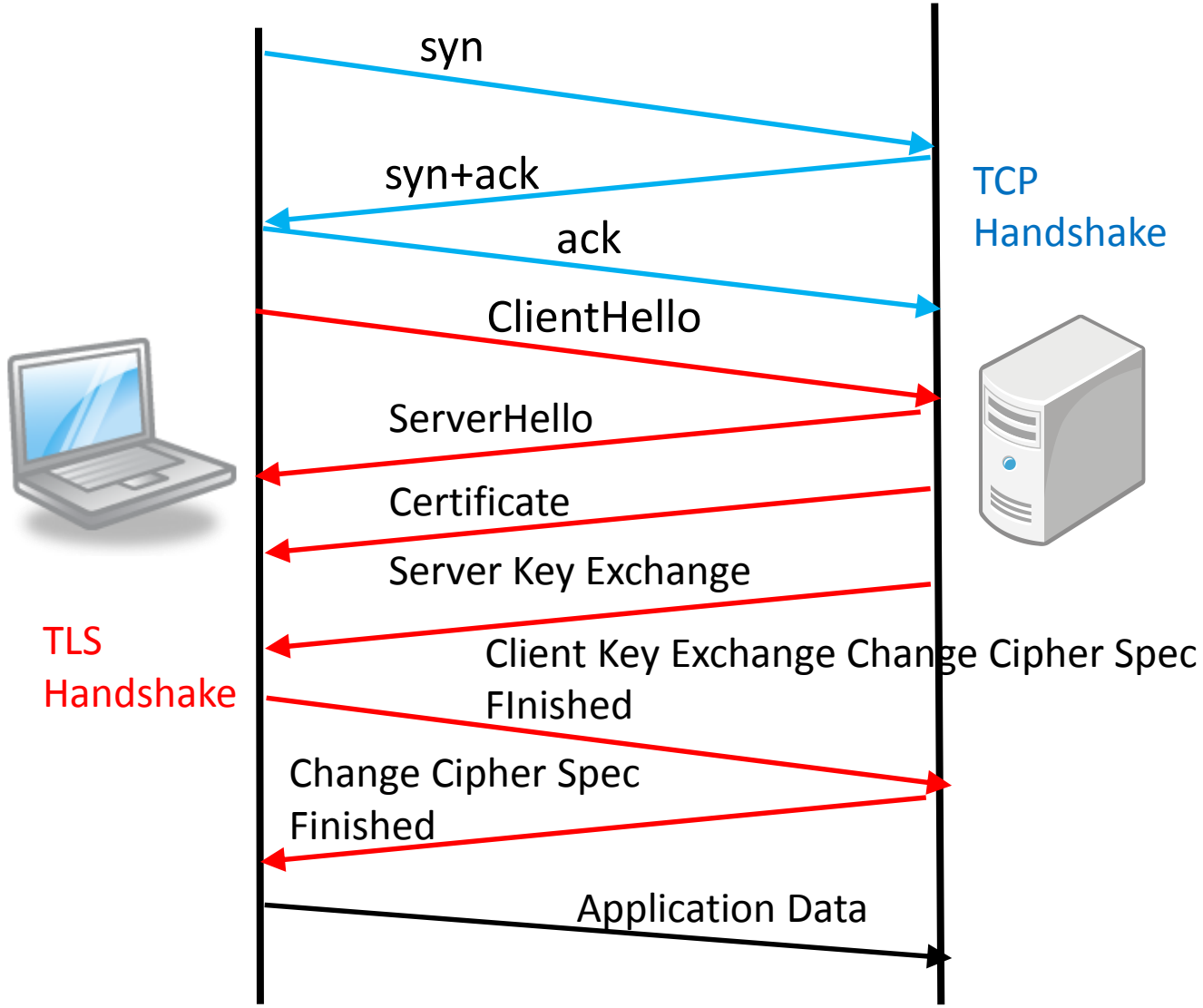
Ballooning Extension

TLS Ticket, Channel ID

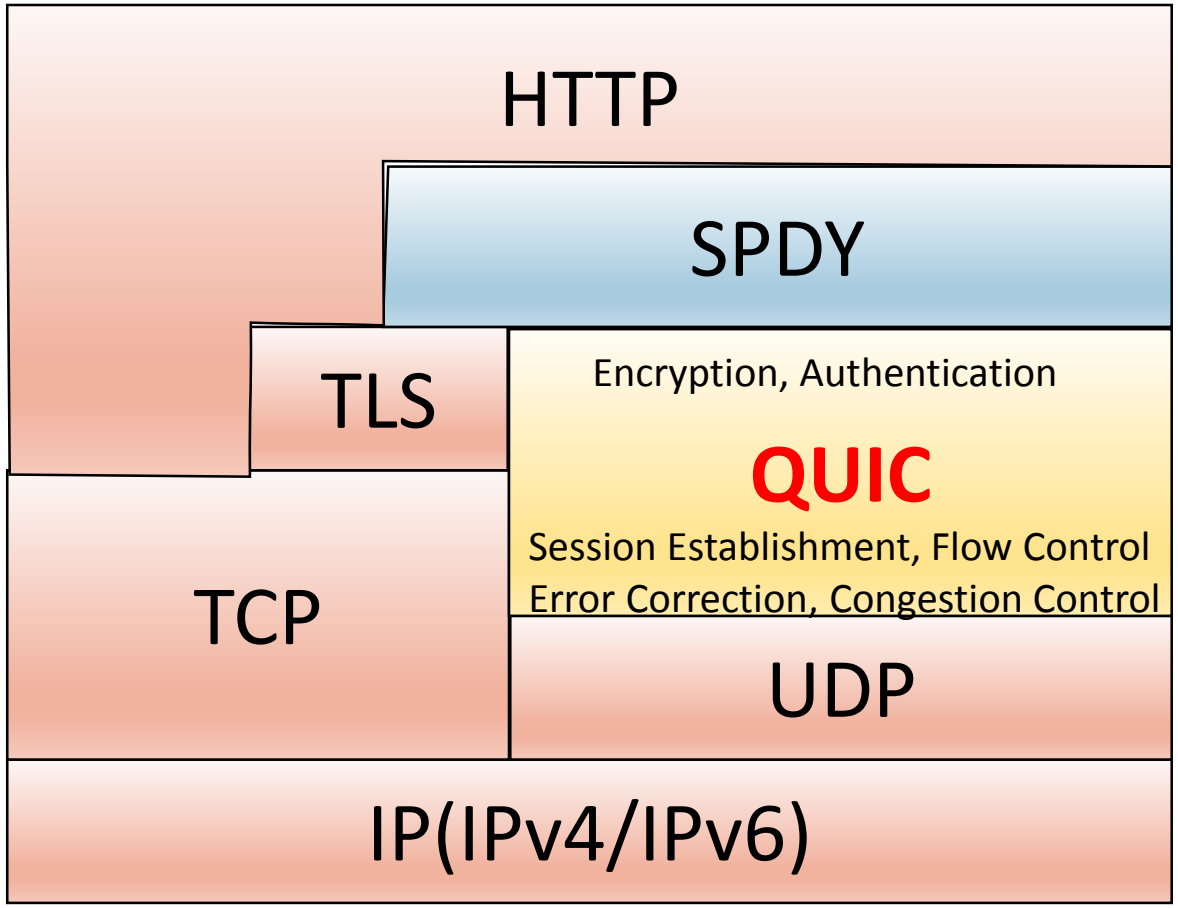
Need to upgrade TLS library on both end and middle box. It's a long way, too.

TCP HoL and TCP+TLS Handshake





QUIC(Quic UDP Internet Connections), MULTIPLEXED STREAM TRANSPORT OVER UDP = sophisticated TLS + TCP on UDP



QUIC Goals

1. Deploy in today's internet
2. **Low latency** (connect, and responses)
   a. It is *ALL* about the latency
3. Reliable-stream support (like SPDY)
   a. Reduce Head Of Line (HOL) blocking due to packet loss
4. Better congestion avoidance than TCP
   a. Iterate and experiment
5. Privacy and Security comparable to TLS
6. Mobile interface migration
7. Improve on quality of sliced bread

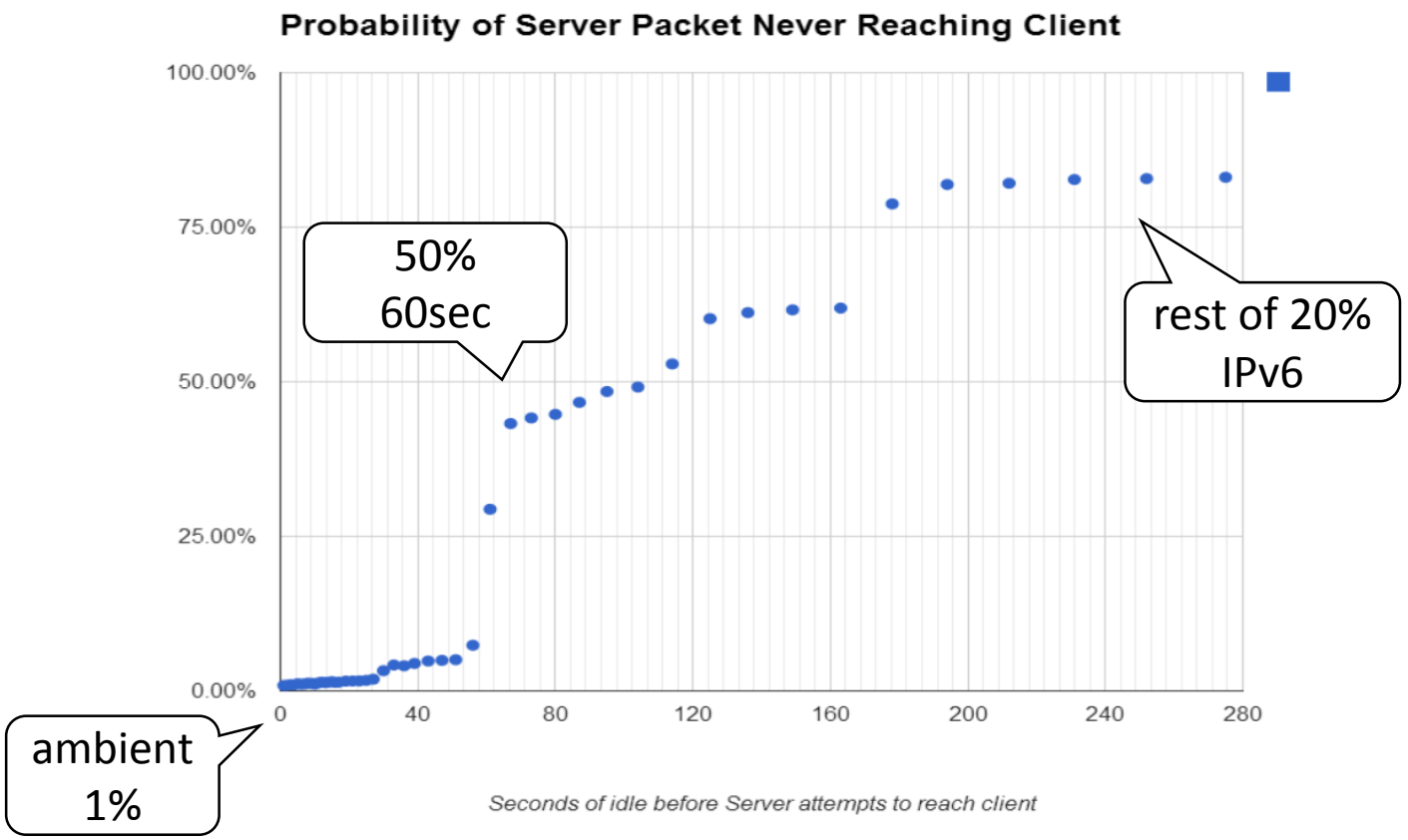
IETF-88 TSV Area Presentation By Jim Roskind http://www.ietf.org/proceedings/88/slides/slides-88-tsvarea-10.pdf

HTTP on UPD. Really?

- 91-94% of users can make outbound UDP to Google.

NAT Unbinding:

How much idle until unbinding?



IETF-88 TSV Area Presentation By Jim Roskind http://www.ietf.org/proceedings/88/slides/slides-88-tsvarea-10.pdf

QUIC History

- 2012 Oct: First Appeared in commit log
- 2013 Feb 25: First News in Tech Crunch. Known to the world.
- 2013 Jun. 27: Officially Announced in Chrome Blog(**Experimenting with QUIC**) Start serving QUIC on www.google.com etc.
- 2014 Aug. 29: Field Trial (Stable Channel: 0.2% desktop, 2% Android, Beta: 25% desktop, 50% Android) from net-dev announcement.

QUIC Version History

Almost Monthly
Version UP

Date	Version		Date	Version	Topics
2013 1/18	QUIC1		2014 2/9	QUIC15	
2013 7/26	QUIC7		2014 2/27	QUIC16	add STOP_WAITING
2013 8/9	QUIC8		2014 3/31	QUIC17	stream flow control (DATA)
2013 8/22	QUIC9		2014 4/28	QUIC18	add PING
2013 9/5	QUIC10		2014 5/17	QUIC19	connection flow control
2013 10/11	QUIC11		2014 6/20	QUIC20	
2013 10/31	QUIC12		2014 7/2	QUIC21	Flow control (HEADERS, CRYPTO)
2013 12/20	QUIC13		2014 8/8	QUIC22	
2014 2/6	QUIC14		2014 8/25	QUIC23	timestamp in Ack, CONGESTION_FEEDBACK deprected

Q023 seems to be more stable than before

QUIC on Chrome

SPDY capturing events (13081)

Alternate Protocol Mappings

Host	Alternate Protocol
quic-demo.iijplus.jp:80	80:quic p=1.000000
clients4.google.com:443	443:quic p=0.010000
www.google.co.jp:443	443:quic p=0.010000
www.gstatic.com:443	443:quic p=0.010000
www.google.com:443	443:quic p=0.010000
ssl.gstatic.com:443	443:quic p=0.010000
mts0.google.com:443	443:quic p=0.010000
mt0.google.com:443	443:quic p=0.010000
maps.gstatic.com:443	443:quic p=0.010000
fonts.gstatic.com:443	443:quic p=0.010000
apis.google.com:443	443:quic p=0.010000
www.google-analytics.com:80	80:quic p=0.010000
ssl.google-analytics.com:443	443:quic p=0.010000

QUIC capturing events (81241)

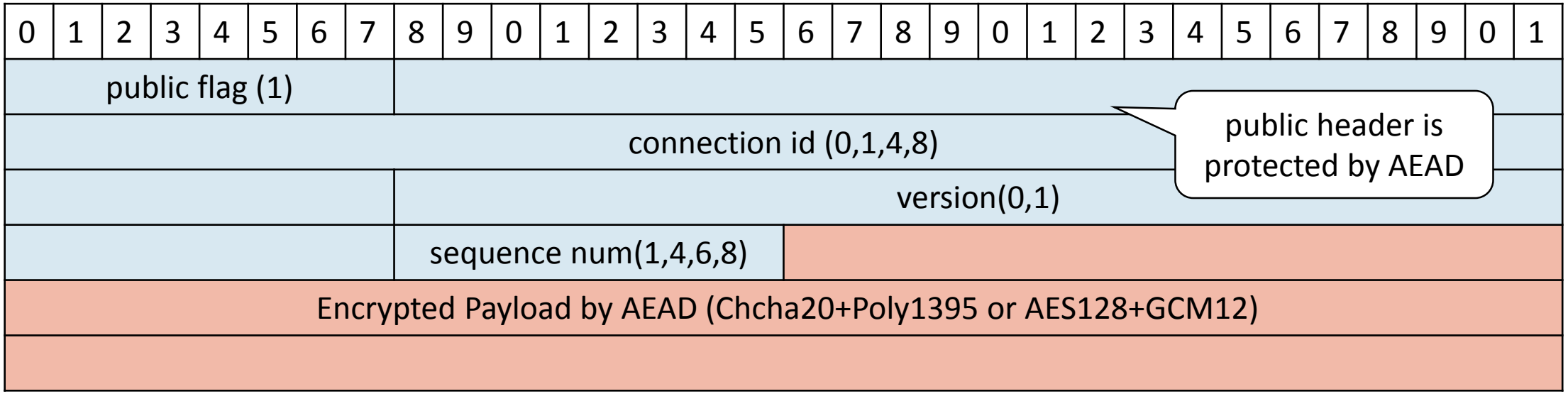
- QUIC Enabled: true
- Alternate Protocol Probability Threshold: 0
- Origin To Force QUIC On: :0
- QUIC connection options:
- Consistent Port Selection Enabled: false

QUIC sessions

View live QUIC sessions

Host	Secure	Version	Peer address	Connection UID	Active stream count	Active streams	Total stream count	Packets Sent	Packets Lost	Packets Received	Connected
quic-demo.iijplus.jp:80	false	QUIC_VERSION_23	192.168.56.200:80	14456011426433218002	1	893	445	1579	0	2210	true
accounts.google.com:443	true	QUIC_VERSION_23	74.125.204.84:443	3512115373630885281	0	None	2	14	0	15	true
apis.google.com:443 map.google.com:443 maps.google.com:443 mts0.google.com:443	true	QUIC_VERSION_23	74.125.235.169:443	4155993567989230875	0	None	10	101	0	182	true
clients4.google.com:443 gg.google.com:443	true	QUIC_VERSION_23	74.125.235.100:443	17071327921568089203	0	None	4	28	0	33	true
fonts.googleapis.com:443	true	QUIC_VERSION_23	74.125.204.95:443	6273421037963735059	0	None	0	5	0	5	true
fonts.gstatic.com:443 ssl.gstatic.com:443	true	QUIC_VERSION_23	74.125.235.191:443	862479597957106054	0	None	3	44	0	79	true
mail.google.com:443	true	QUIC_VERSION_23	74.125.235.181:443	8523583242707538299	0	None	1	8	0	10	true
mt0.google.com:443	true	QUIC_VERSION_23	173.194.117.238:443	14925893666429730333	0	None	8	58	0	96	true
ssl.google-analytics.com:443	true	QUIC_VERSION_23	74.125.235.158:443	6616480318124246971	0	None	0	5	0	9	true
www.gmail.com:443	true	QUIC_VERSION_23	74.125.235.181:443	2281575570898097 89	0	None	11	393	0	759	true
www.google.co.jp:443	true	QUIC_VERSION_23	74.125.235.191:443	7750091045683123386	0	None	22	117	0	179	true
www.google.com:443	true	QUIC_VERSION_23	74.125.235.178:443	730107540064837349	0	None	9	50	0	83	true

QUIC Wire Frame Format (Public Header)



Connection ID (a.k.a GUID) is a 64bit random value to identify client connection.
Even after client's ip or port was changed, QUIC can handle connection.

Crypto Stream(stream=1) is not Encrypted but protected FNV1a-128 hash(using lower 96bit)

QUIC Public Flag(8bit)

NONE	-	-	-	-	-	-	-	0
VERSION	-	-	-	-	-	-	-	1
RST	-	-	-	-	-	-	1	-
0BYTE_CONNECTION_ID	-	-	-	-	0	0	-	-
1BYTE_CONNECTION_ID	-	-	-	-	0	1	-	-
4BYTE_CONNECTION_ID	-	-	-	-	1	0	-	-
8BYTE_CONNECTION_ID	-	-	-	-	1	1	-	-
1BYTE_SEQUENCE	-	-	0	0	-	-	-	-
2BYTE_SEQUENCE	-	-	0	1	-	-	-	-
4BYTE_SEQUENCE	-	-	1	0	-	-	-	-
6BYTE_SEQUENCE	-	-	1	1	-	-	-	-

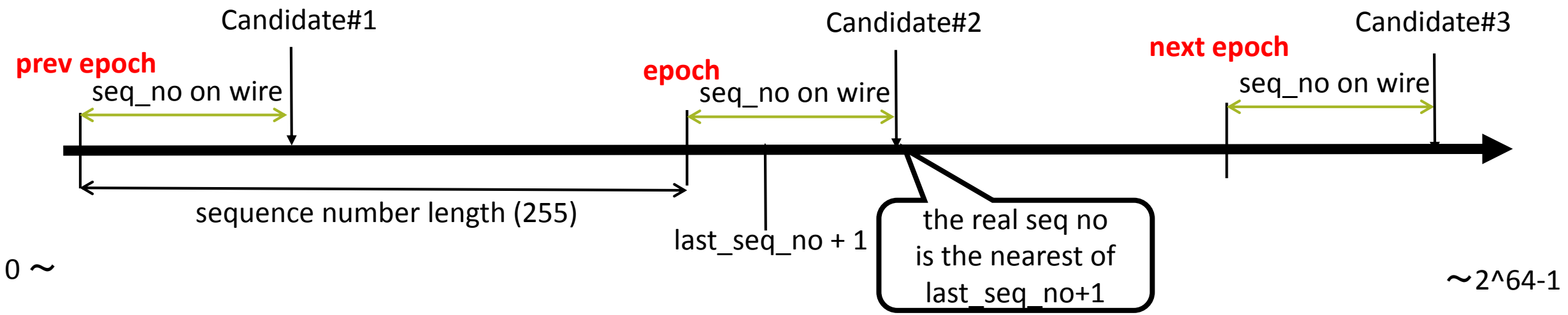
Version field is **no longer needed** after negotiation.

Use **8byte** Connection ID by default unless negotiated.

Sequence Number is fixed 64bit. but it can be sent **only 1byte** on the wire.

Why 64bit seq num can send via 8bit length on the wire?

- Less than 256 frames on the fly. But their arrival is not ordered.
- Endpoint should remember the last_seq_no that was received lately.
- Choise the nearest number to last_seq_no+1 from the epoch number + received 8bit number.
- If we received very old frame, it can't be decrypted because seq is used in IV.



QUIC Wire Frame Format (Private)

0 1 2 3 4 5 6 7	8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
public flag (1)	
connection id (0,1,4,8)	
	version(0,1)
	sequence num(1,4,6,8) / private flag(1) / FEC offset(0,1)
frame_type(1)	Frame Data

Private Flag

NONE	-	-	-	-	-	-	-	0
ENTROPY	-	-	-	-	-	-	-	1
FEC_GROUP	-	-	-	-	-	-	1	-
FEC	-	-	-	-	-	1	-	-

Random bit

this frame is in FEC group

FEC parity data is included

Entropy hash to defense Optimistic Ack Attack

QUIC Private Flag

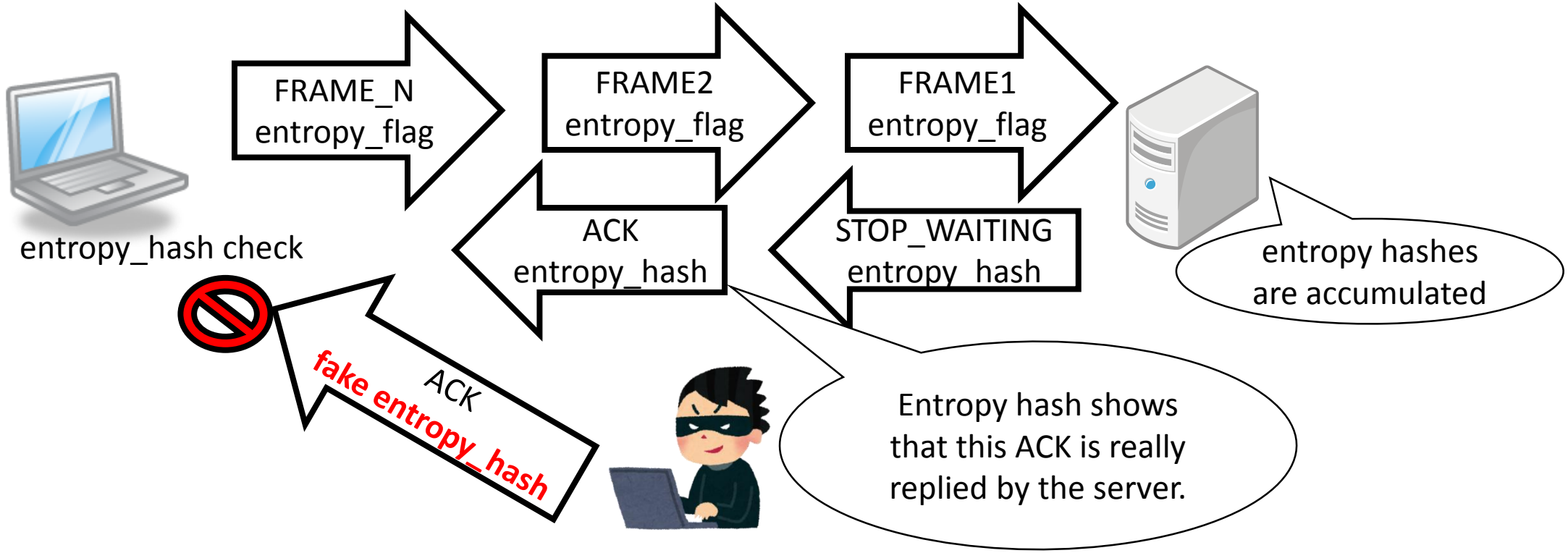
ENTROPY	-	-	-	-	-	-	-	

1bit entropy flag
0 or 1 Random

1byte entropy: entropy flag << seq number%8

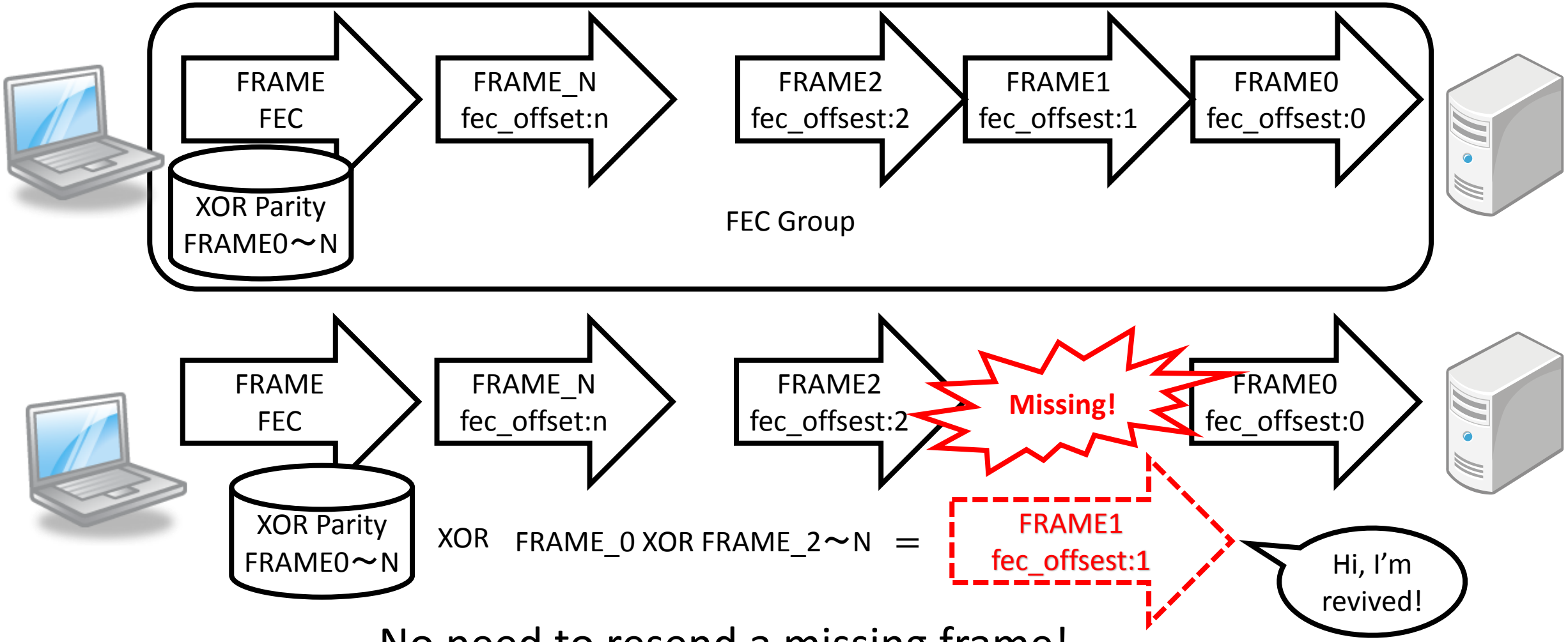
1byte entropy hash = frame1_entropy **XOR** frame2_entropy **XOR** …. **XOR** frameN_entropy

entropy hash is **cleared** when ACK is received



FEC (Forward Error Collection)

like a RAID in network packets (disabled on Chrome now)



No need to resend a missing frame!

QUIC Frame Type

CongestionFeedbackMask	0	0	1	0	0	0	0	0	(deprecated in Q022)
AckMask	0	1	0	0	0	0	0	0	
StreamMask	1	0	0	0	0	0	0	0	
0: PADDING	0	0	0	0	0	0	0	0	
1: RST_STREAM	0	0	0	0	0	0	0	1	
2: CONNECTION_CLOSE	0	0	0	0	0	0	1	0	
3: GOAWAY	0	0	0	0	0	0	1	1	
4: WINDOW_UPDATE	0	0	0	0	0	1	0	0	
5: BLOCKED	0	0	0	0	0	1	0	1	
6: STOP_WAITING	0	0	0	0	0	1	1	0	
7: PING	0	0	0	0	0	1	1	1	

Frame Type (Stream)

StreamMask	1	-	-	-	-	-	-	-
stream_id_length	-	-	-	-	-	-	-	-
offset_length	-	-	-	-	-	-	-	-
has_data_length	-	-	-	-	-	-	-	-
FIN	-	-	-	-	-	-	-	-

0x00-0x03 (length: 1～4)

0x00-0x07 (length: 0, 2～8)

whether data_length is appended or not

Quic Stream

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td></tr>
<tr><td colspan="8">public flag (1)</td><td colspan="24"></td></tr>
<tr><td colspan="32">connection id (0,1,4,8)</td></tr>
<tr><td colspan="8"></td><td colspan="24">version(0,1)</td></tr>
<tr><td colspan="8"></td><td colspan="8">sequence num(1,4,6,8)</td><td colspan="8">private flag(1)</td><td colspan="8">FEC offset(0,1)</td></tr>
<tr><td colspan="8">frame_type(0x80～)</td><td colspan="24">stream_id(1,2,3,4)</td></tr>
<tr><td colspan="16">offset(0, 2,3,4,5,6,7,8)</td><td colspan="16">data length(0, 2)</td></tr>
<tr><td colspan="32">Stream DATA</td></tr>
<tr><td colspan="32"></td></tr>
</table>

Crypt Functionality in QUIC

- Encryption both http:// and https://
- Server Cert Proof is made on https://

Quic Crypto Stream (stream_id=1)

<table>
<tr><td>public flag (1)</td><td colspan="3"></td></tr>
<tr><td colspan="4">connection id (0,1,4,8)</td></tr>
<tr><td></td><td colspan="3">version(0,1)</td></tr>
<tr><td></td><td>sequence num(1,4,6,8)</td><td>private flag(1)</td><td>FEC offset(0,1)</td></tr>
<tr><td>frame_type(0x80～)</td><td colspan="3">stream_id = 0x01</td></tr>
<tr><td colspan="2">offset(0, 2～8)</td><td colspan="2">data length(0, 2)</td></tr>
<tr><td colspan="4">FNV1a-128 hash truncated higher 32bit (12)　snipped</td></tr>
<tr><td colspan="4">Message Tag(4)</td></tr>
<tr><td colspan="2">Number of Entries(2)</td><td colspan="2">Padding(2)</td></tr>
<tr><td colspan="4">Tag#1(4)</td></tr>
<tr><td colspan="4">end_offset#1(4)</td></tr>
<tr><td colspan="4">Tag#2(4)</td></tr>
<tr><td colspan="4">end_offset#2(4)</td></tr>
<tr><td colspan="4">Value#1</td></tr>
<tr><td colspan="4">Value#2</td></tr>
</table>

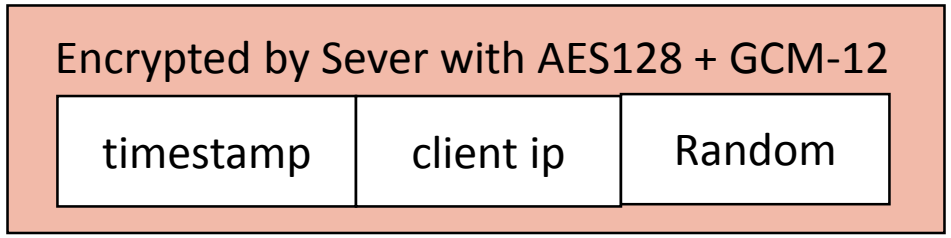
Quic Tag List (some of 60 total)

Crypto Tag	Meaning
CHLO	**Client Hello**
SHLO	**Server Hello**
SCFG	Server Config
REJ	**Reject**
CETV	Client encrypted tag-value pairs
PRST	Public Reset
SCUP	**Server Config Update**
P256	ECDH Curve P-256
C255	ECDH Curve25519 **Default**
NULL	null algorithm
AESG	AES128 + GCM-12
CC12	ChaCha20 + Poly1305 **Default**
STK	Source Address Token

Key Exchange

AEAD

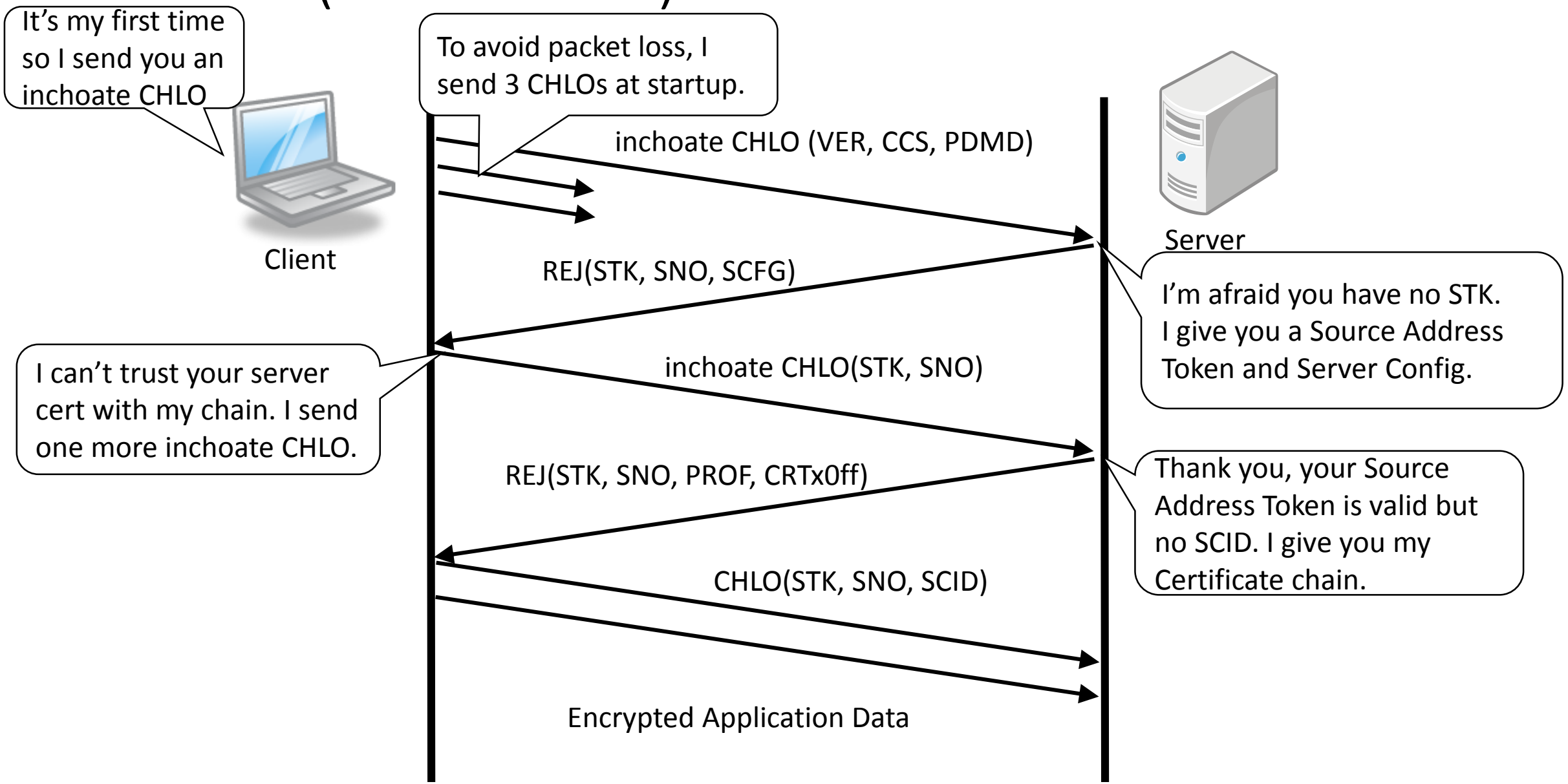
STK(56byte)



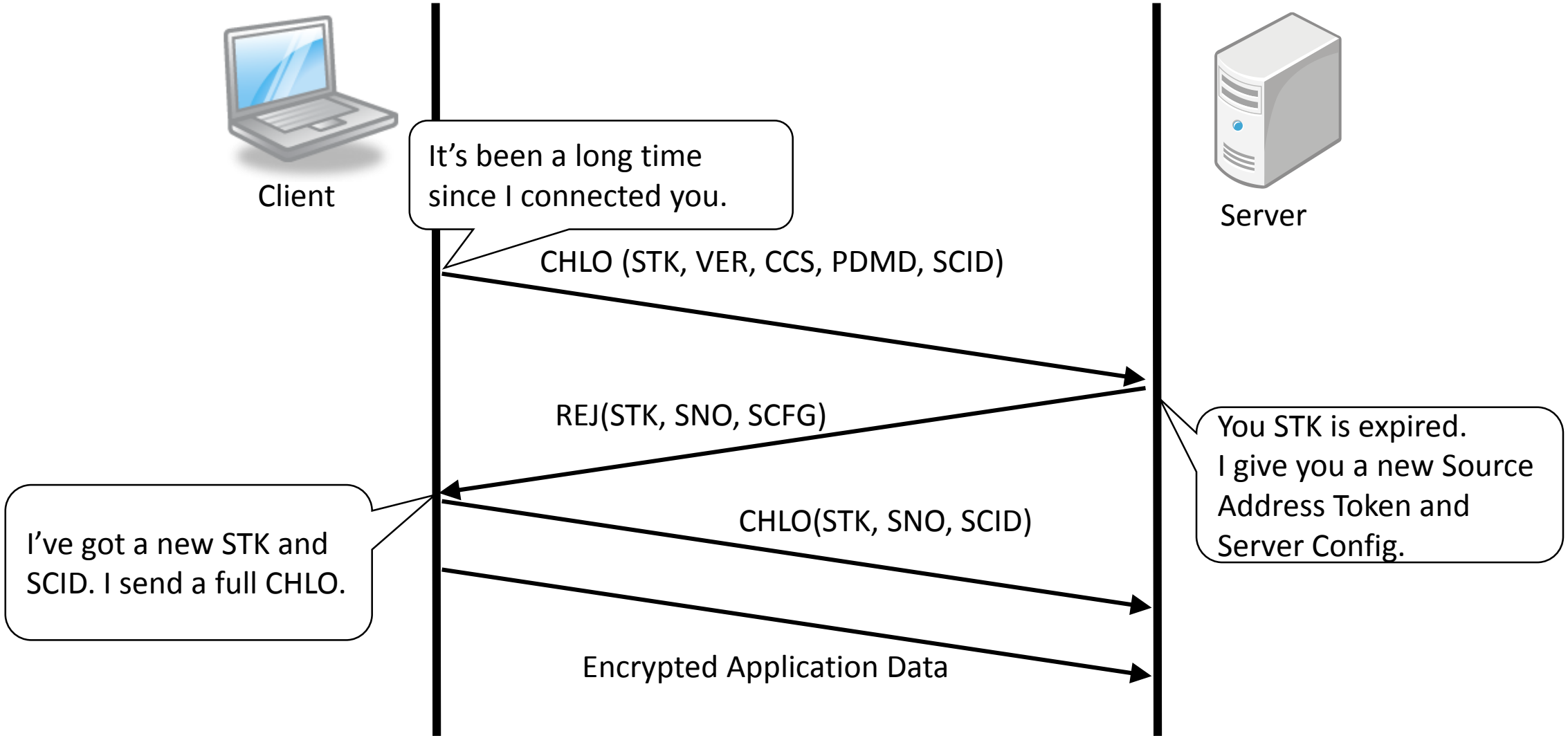
Example CHLO

CHLO
- PAD : (607 bytes of padding)
- SNI : 0x717569632D64656D6F2E69696A706C75732E6A70 Server Name Indicator
- STK : 0x95ACC8F704031697A396019034133971C0C65868BBBB32306B2929930C12FA0BF5889191CD1CAC1B7AE2B7FF17AF1C2F4880BC11 Source Address Token
- VER : 'Q023' Version
- CCS : 0x399FF95340F7FEC97B26E9E7E45C71FF Common Certificate Set
- NONC: 0x5452D757288EE7AC8D6709DD289D0C6BA0723C74960E2F4F30814F64C48F436A Nonce
- MSPC: 100 Max streams per connection
- AEAD: 'AESG' AEAD algorithm
- UAID: canary Chrome/40.0.2205.0 User Agent ID
- SCID: 0x83F4EA5D3EFD8BC21B829BA2BFDF304E Server Config ID
- SWND: 20 Server's Initial Congestion Window
- ICSL: 30 Idle connection state lifetime
- KATO: 0 Keep alive Timeout
- PUBS: 0x65F638326982B96A6B7251AEF0A61B568A6EE5D34FDD7B2E3C46D9453B809F36 Public Key
- KEXS: 'C255' Key Exchange methods
- COPT: Connection Option
- CGST: 'QBIC' Congestion Control Feedback Type
- IRTT: 957 Estimated initial RTT in use
- CFCW: 10485760 Initial session/connection flow control receive window
- IFCW: 10485760 Initial flow control receive window
- SFCW: 10485760 Initial stream flow control receive window

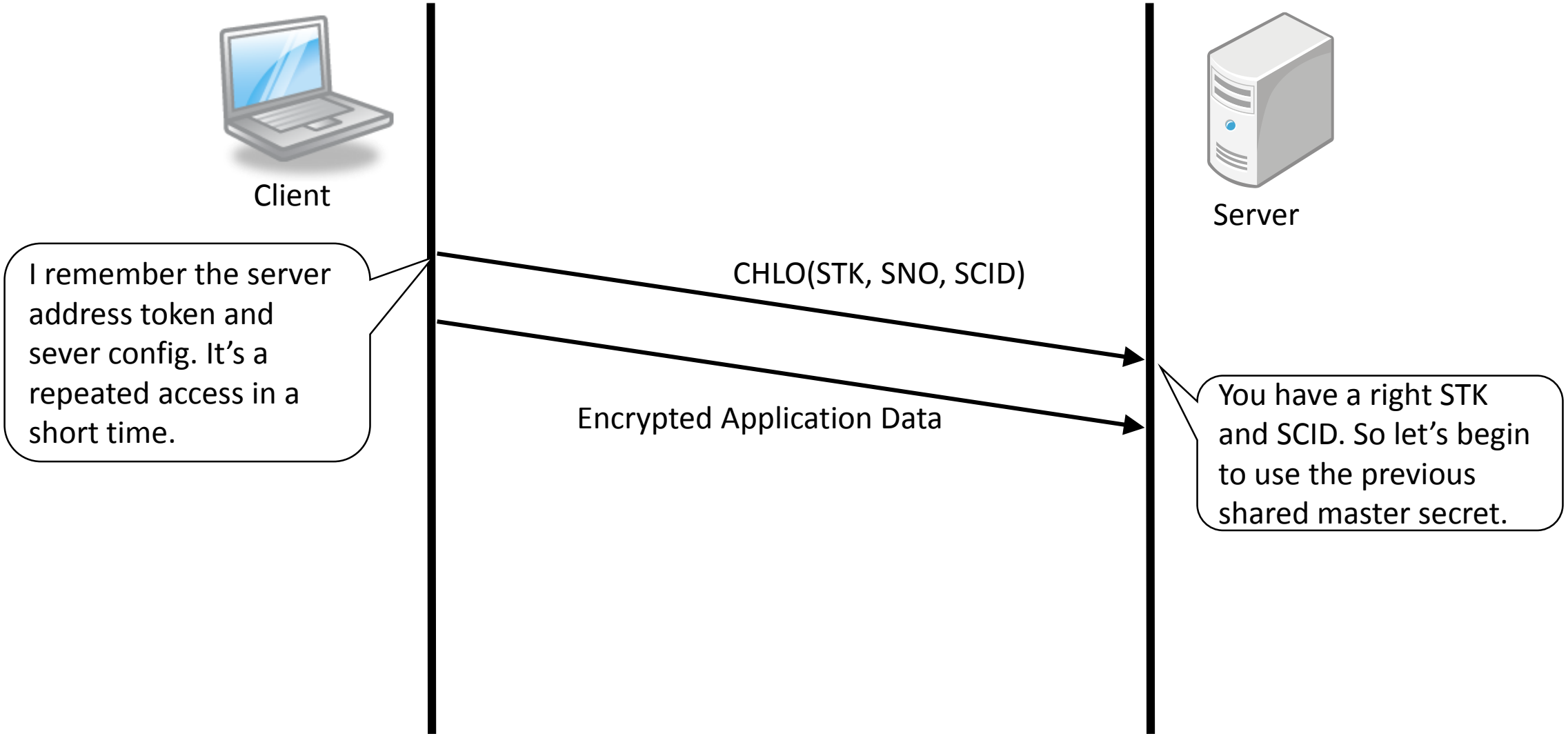
2-RTT (worst case)



1-RTT (normal)


Client
Server
It's been a long time since I connected you.
CHLO (STK, VER, CCS, PDMD, SCID)
REJ(STK, SNO, SCFG)
You STK is expired. I give you a new Source Address Token and Server Config.
I've got a new STK and SCID. I send a full CHLO.
CHLO(STK, SNO, SCID)
Encrypted Application Data

0-RTT (repeated resumption)



non Blocking during Cipher Secret Update

Client

Pleas Key Update!

SCUP(STK, SCFG{AEAD, SCID, PUBS, KEXS, OBIT, EXPY})

OLD KEY

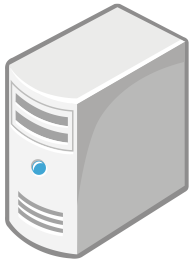
OLD KEY

OLD KEY

NEW KEY

NEW KEY

NEW KEY



Server

decrypter

alternative decrypter

I don’t known when client update key.
So I try decrypt frame with two (old and new) keys.

Hashed Cert Chain and Compressed Server Cert

Client send FNV1a hash of local cert chain as CCS

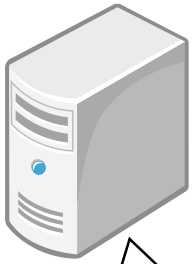
Cert List
+ Hash

Cached Cert List
+ Hash

CHLO(STK, …,CCS)
CCS : 0x399FF95340F7FEC9

REJ(CRT, PROF)

gzip compressed



Server find missed Certs from CCS and send gzip compressed cert chain as CRT and signature as PROF

Cert List#1
+ Hash

Cert List#2
+ Hash

Transport Functionality

ACK Frame

0 1 2 3 4 5 6 7	8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1		
public flag (1)			
connection id (0,1,4,8)			
	version(0,1)		
	sequence num(1,4,6,8)	private flag(1)	frame_type(0x40～0x7f)
entropy hash(1)	Largest Observed	Largest Observed Delta Time(2)	
num of received packet(1)	delta largest observed#1	time delta#1	
		delta largest observed#2	time delta#2
	delta largest observed#3	time delta#3	
num of missing packet(1)	missing packet seq#1	range length#1	missing packet seq#2
range length#2	missing packet seq#3	range length#3	num of revived packets(1)
revived packet seq#1	revived packet seq#2	revived packet seq#3	revived packet seq#4

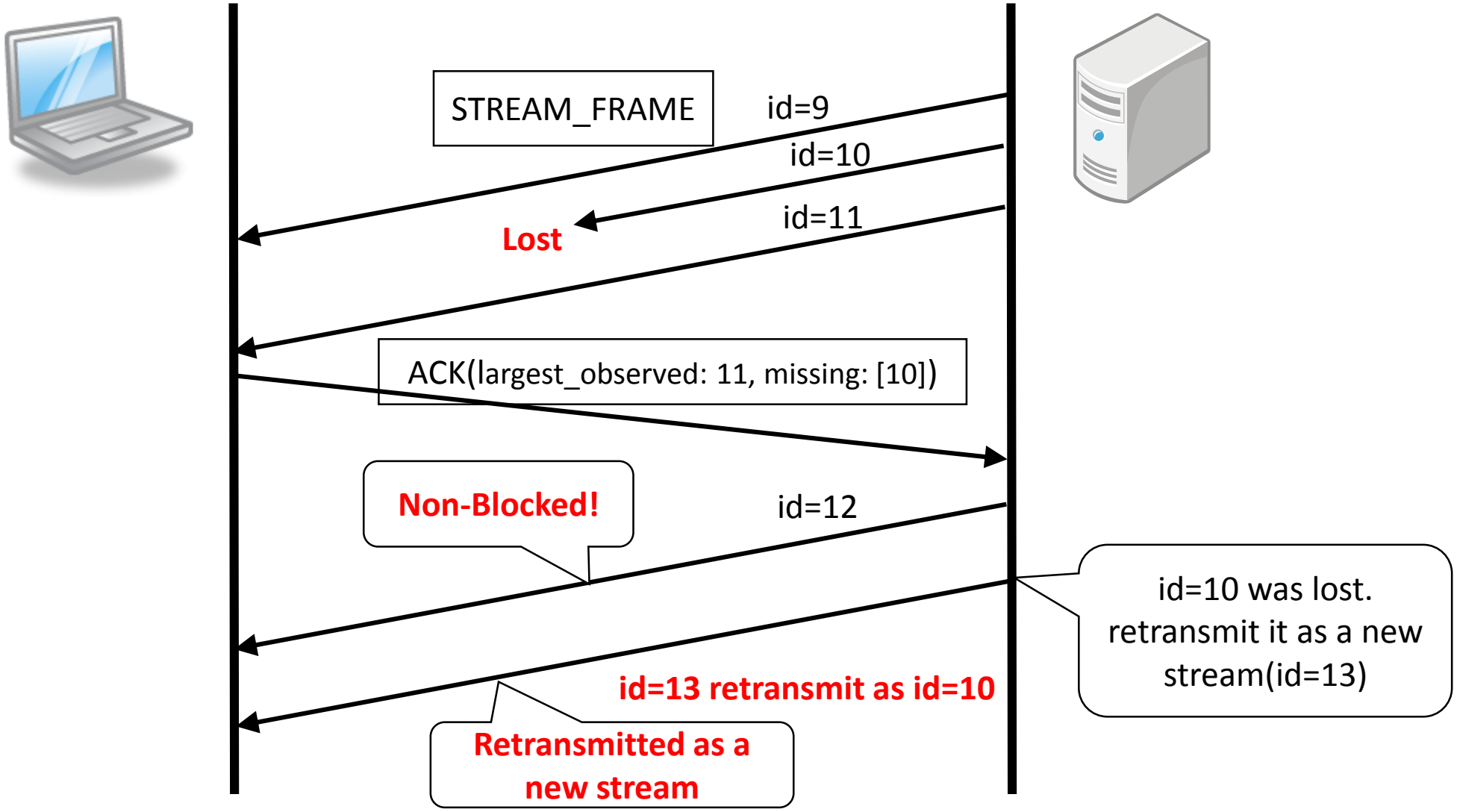
ACK conveys Largest Observed Seq, Packet Timestamp, Missing Packets and Revived Packets

ACK flag

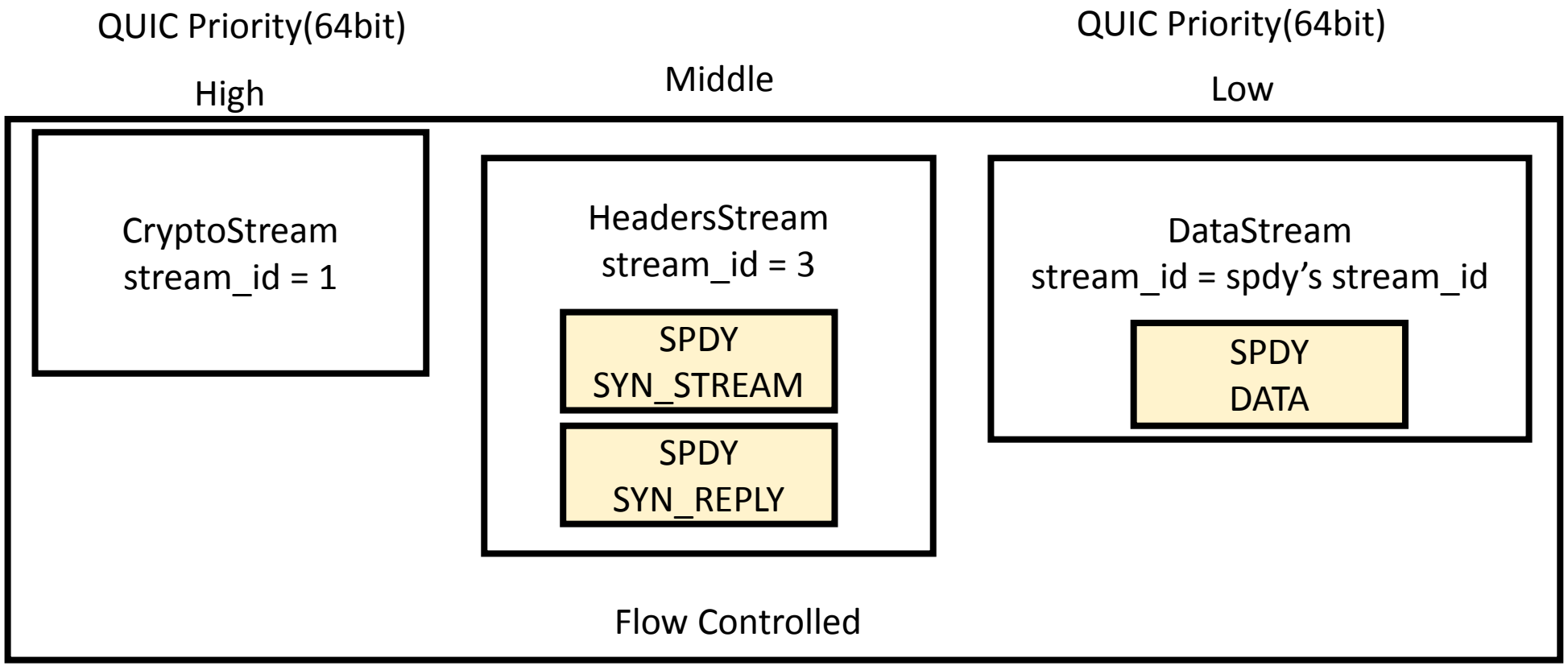
AckMask	0	1	-	-	-	-	-	-
Missing SequenceNumberLength	0	1	-	-	-	-	-	-
Largest Observed SequenceNumberLength	0	1	-	-	-	-	-	-
IsTruncated	0	1	-	-	-	-	-	-
HasNack	0	1	-	-	-	-	-	-

00: 1byte length
01: 2byte length
10: 4byte length
11: 6byte length

Retransmit a lost frame as a new stream



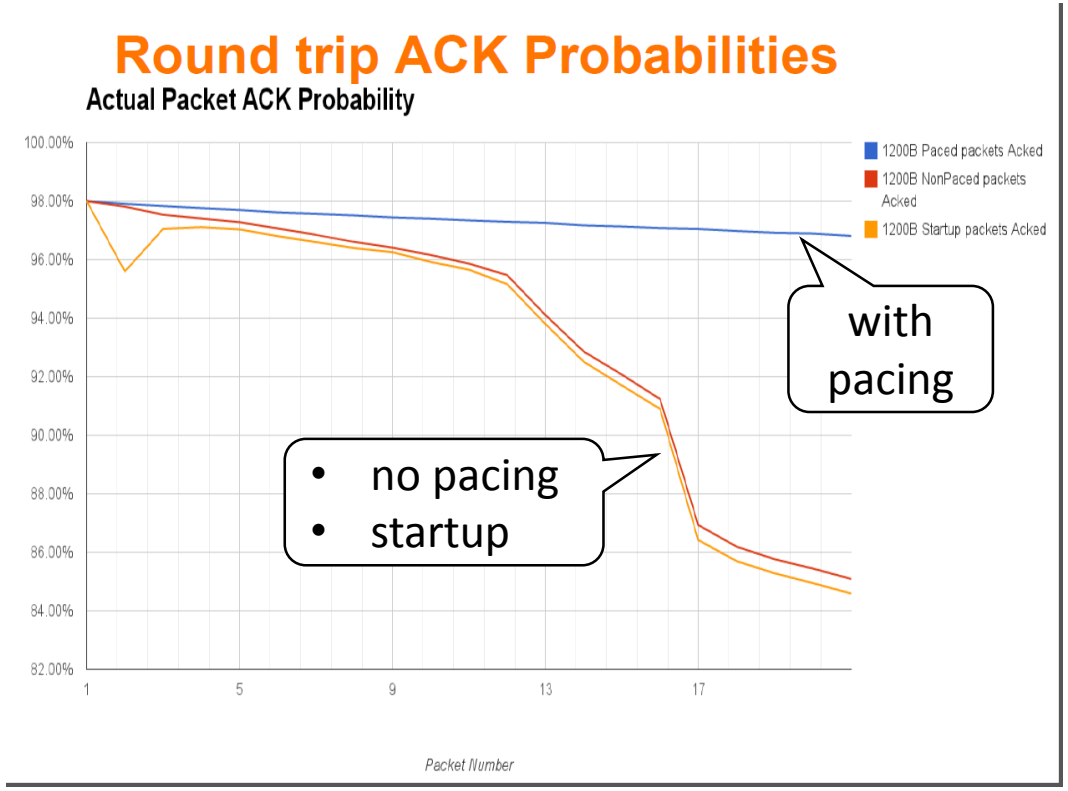
Quic Stream and SPDY Binding



Congestion Control

- Default TCP Cubic + pacing with bandwidth estimation
- Baseline: prevents Internet Congestion Collapse
- Google shows the graph that pacing causes good performance

- Maybe No time to present now.
- Need more investigation for me.

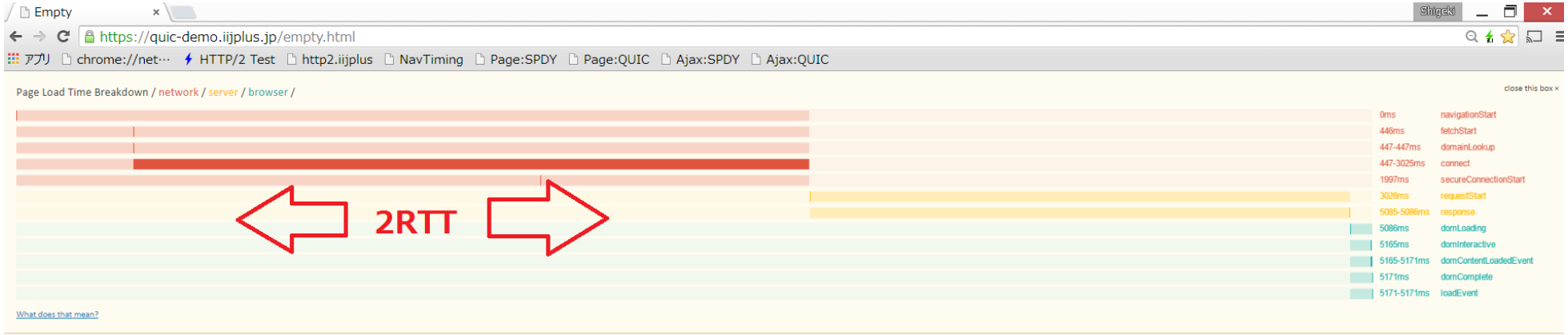


IETF-88 TSV Area Presentation By Jim Roskind
http://www.ietf.org/proceedings/88/slides/slides-88-tsvarea-10.pdf

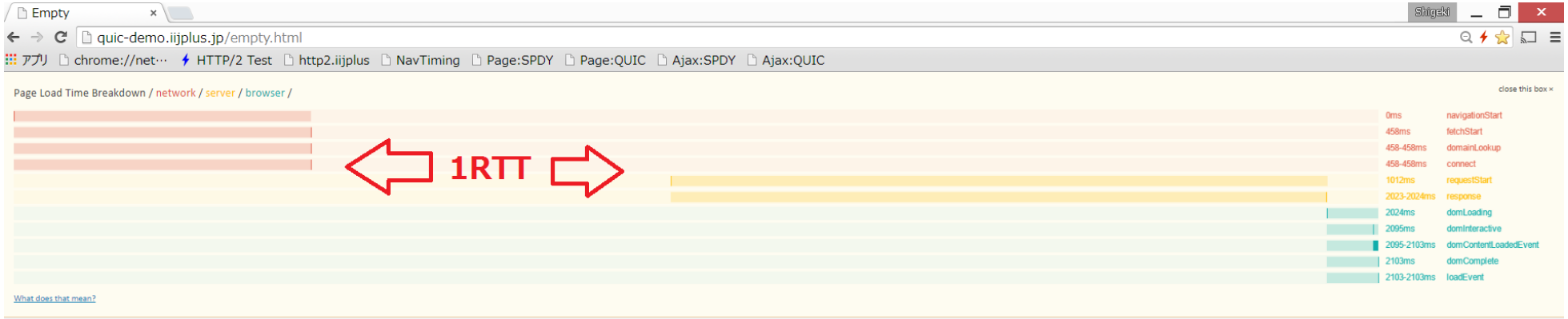
QUIC vs SPDY demo

- https://www.youtube.com/watch?v=bP-8vfDX2ts

QUIC vs SPDY demo #1 (1000ms RTT delay)

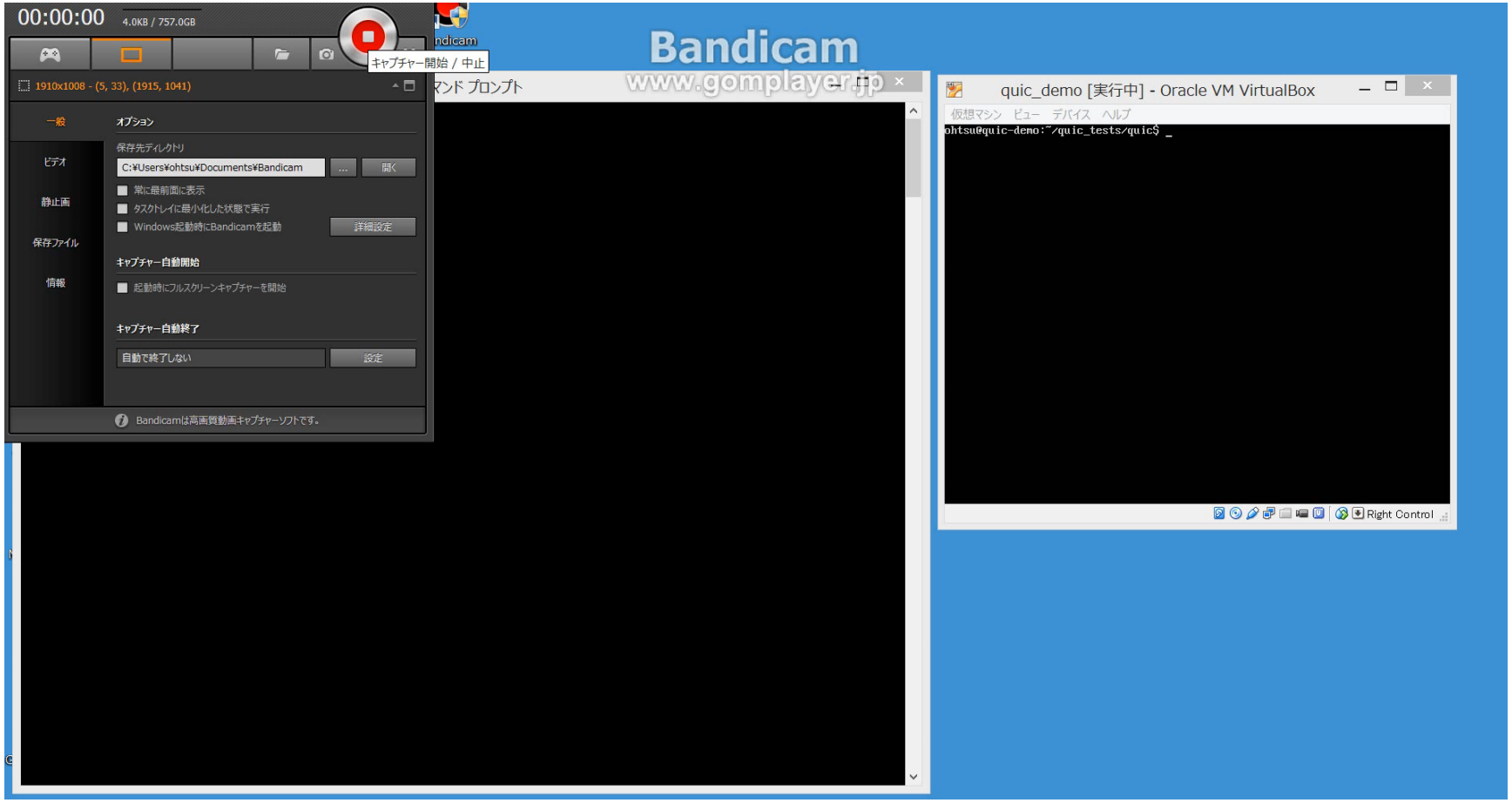


Initial RTT test with Navigation Timing API. Protocol: spdy/3.1



Initial RTT test with Navigation Timing API. Protocol: quic/1+spdy/3

QUIC vs SPDY demo #2 (30% packet loss) start at 1 min 30 sec



Thanks