

Pluginizing QUIC

Moshe Kol, DANSS 2021

Quentin De Coninck, François Michel, Maxime Piraux, Florentin Rochet, Thomas Given-Wilson, Axel Legay, Olivier Pereira, and Olivier Bonaventure. 2019. **Pluginizing QUIC**. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '19)*. Association for Computing Machinery, New York, NY, USA, 59–74. DOI:<https://doi.org/10.1145/3341302.3342078>

Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, and Zhongyi Shi. 2017. **The QUIC Transport Protocol: Design and Internet-Scale Deployment**. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 183–196. DOI:<https://doi.org/10.1145/3098822.3098842>

Agenda

- QUIC Protocol Overview
 - Motivation
 - Features overview

- Pluginized QUIC
 - Motivation
 - Plugins architecture

What is QUIC? (1/2)

- New transport protocol, secure and reliable
- UDP-based
 - Because UDP is well-supported on the Internet
- Initially designed by Google
 - Deployed in 2014
 - Seek to replace the HTTPS stack



<https://quicwg.org/>

What is QUIC? (2/2)

- Google reports better quality of experience
 - Improved YouTube rebuffer rate by 15-18%
 - Improved Google search latency by 3.6-8%

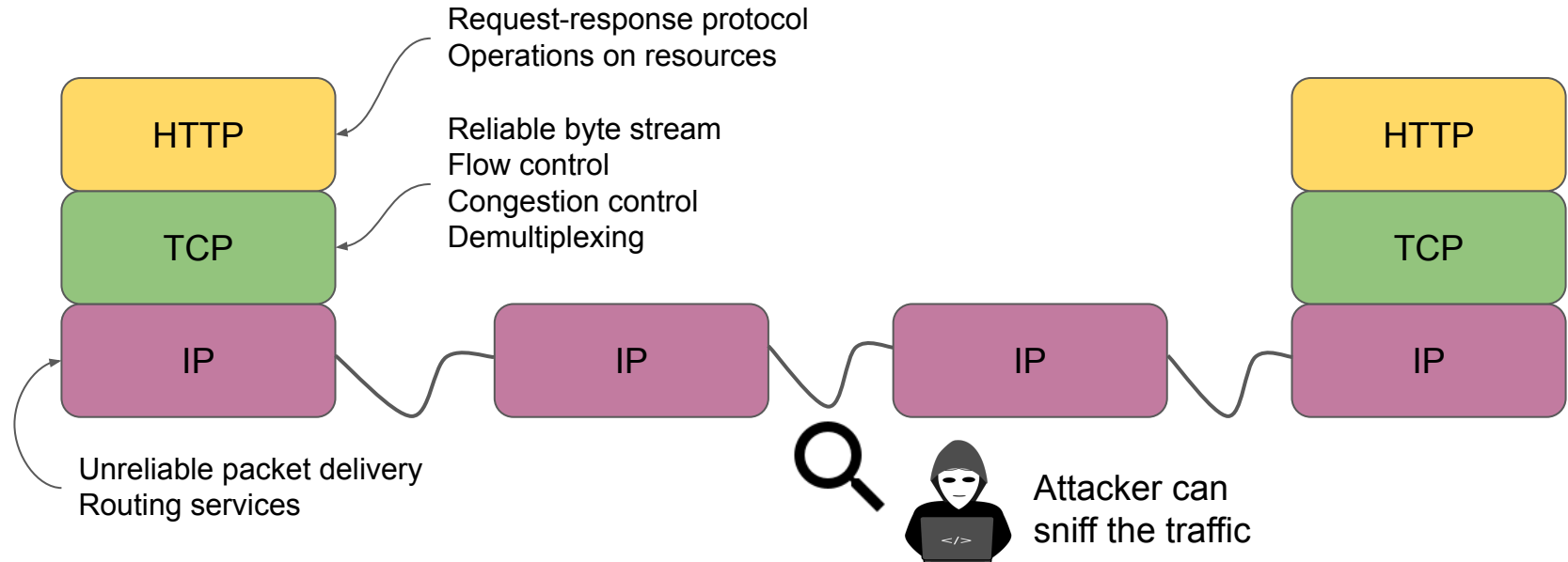
- IETF working group was formed in 2016
 - Current draft: 34
 - RFC expected at the end of 2021



<https://quicwg.org/>

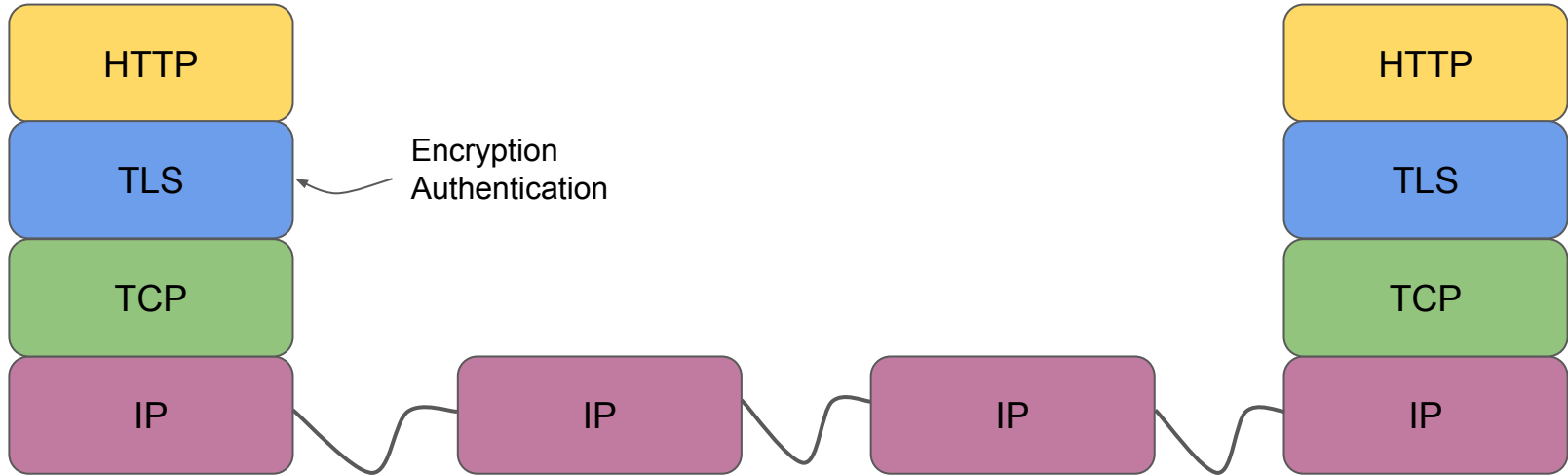
The story of the web (1/2)

- Layered design
 - End-to-end principle



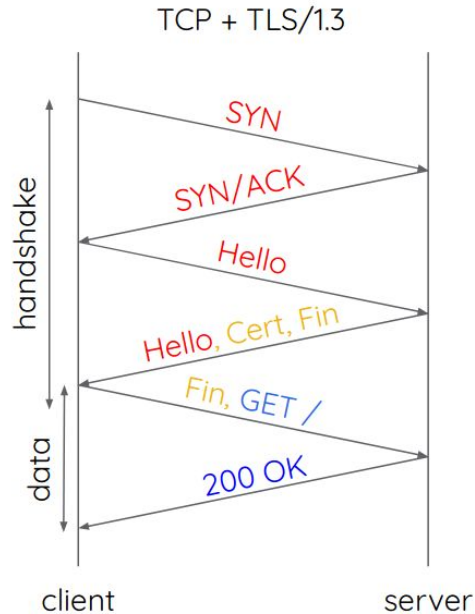
The story of the web (2/2)

- Security is needed



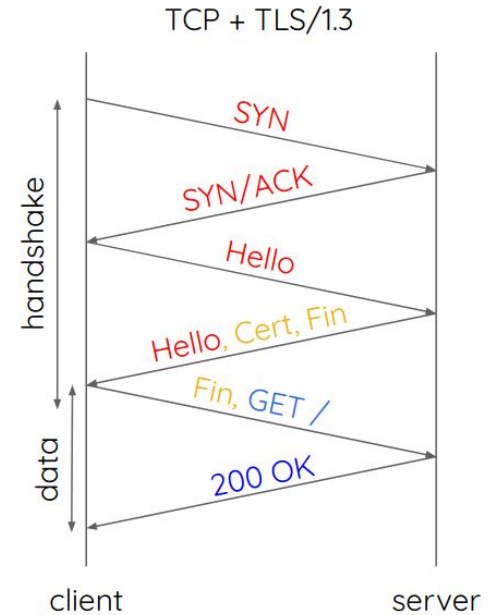
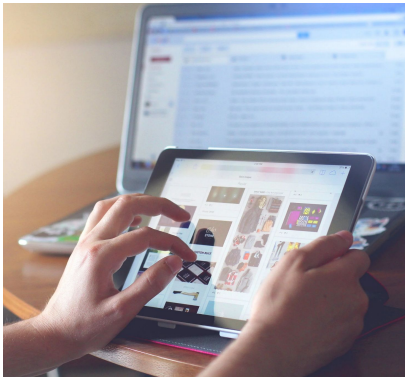
Drawbacks: Handshake latency (1/2)

- At least 2-RTTs before useful data can be sent



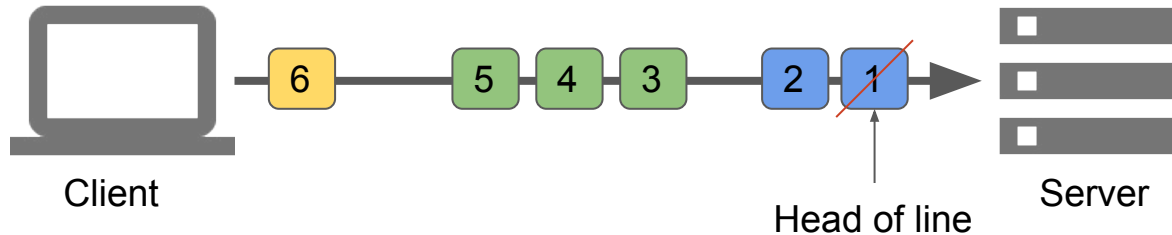
Drawbacks: Handshake latency (2/2)

- At least 2-RTTs before useful data can be sent
- Demands on reducing web latency
 - Web as a platform for applications
 - Many objects are needed to be fetched



Drawbacks: TCP head-of-line blocking

- HTTP/1.0: Each HTTP request requires a TCP connection.
- HTTP/1.1: Introduced persistent connection and pipelining.
- HTTP/2: Multiple HTTP requests on the same TCP connection.



Lost segment of one request delays others!

Drawbacks: Protocol ossification

- Middleboxes: “intermediary devices performing functions other than the normal” (RFC 3234)
 - NAT routers: rewrite transport and IP headers
 - Firewalls: block unknown traffic for security reasons
 - Load balancers, Proxies, ...

D.4. Middlebox Compatibility Mode

Field measurements [Ben17a] [Ben17b] [Res17a] [Res17b] have found that a significant number of middleboxes misbehave when a TLS client/server pair negotiates TLS 1.3. Implementations can increase the chance of making connections through those middleboxes by making the TLS 1.3 handshake look more like a TLS 1.2 handshake:

RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3

Drawbacks: Slow deployment

- TCP is implemented in the OS kernel
 - It takes time to make changes and upgrade systems

- Upgrade cycles
 - OS: order of months-years
 - Browsers: order of weeks (Chrome: 6 weeks)

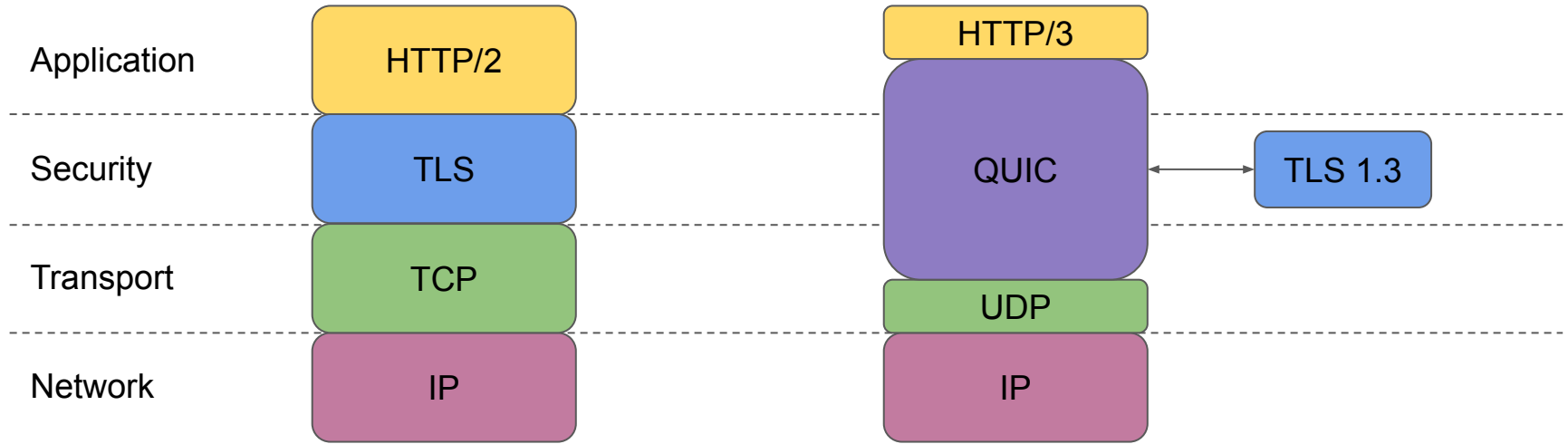
Drawbacks: Slow deployment

- TCP is implemented in the OS kernel
 - It takes time to make changes and upgrade systems
- Upgrade cycles
 - OS: order of months-years
 - Browsers: order of weeks (Chrome: 6 weeks)
- A UDP-based protocol is implemented in userspace
 - More flexible



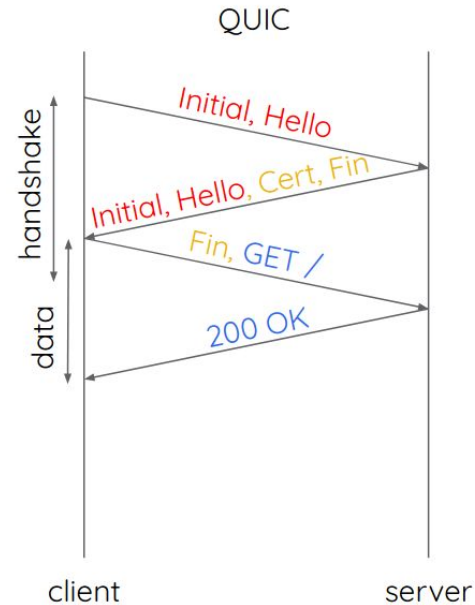
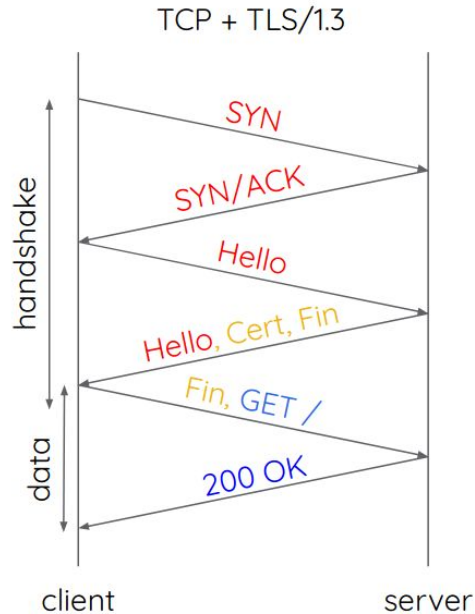
QUIC Overview

Introducing QUIC



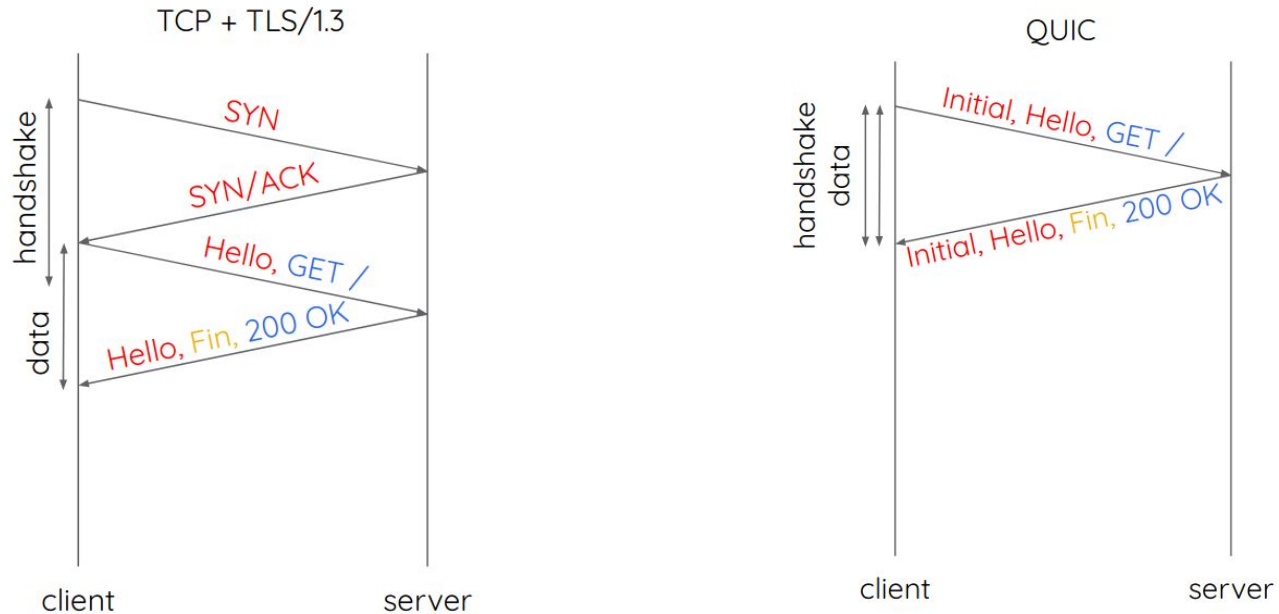
Overview: Connection establishment (1/3)

Initial connection



Overview: Connection establishment (2/3)

Subsequent connection



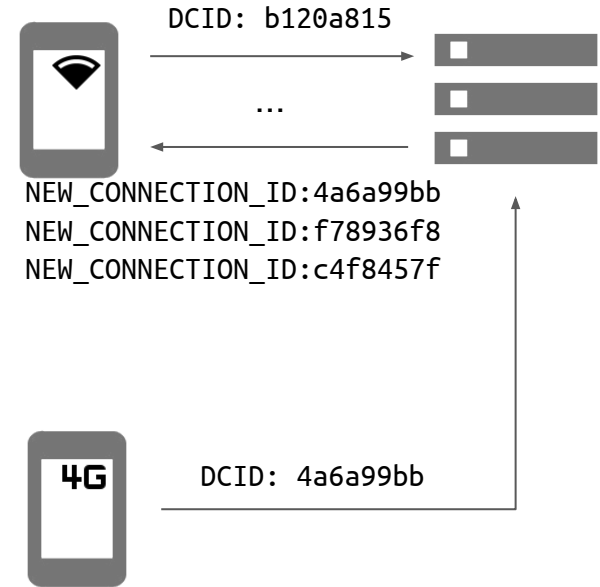
Overview: Connection establishment (3/3)

- Connection IDs are exchanged during handshake.
 - Each endpoint chooses a connection ID (up to 20 bytes).
 - Packets specify the other endpoint's CID.
- QUIC uses connection ID to identify a connection.
 - Unlike TCP which uses a 5-tuple (`src_ip`, `src_port`, `dst_ip`, `dst_port`, `transport`).
- Transport options exchanged in `quic_transport_parameters` TLS extension.
 - E.g. flow control limits, max number of streams, max idle time.

```
CID:  
d09d13e12c698b7fb21d  
c45e75a22a3b7af8f8e3
```

Overview: Connection migration

- Connection IDs gives more flexibility.
- Client can change IP address and keep the connection open.
- Resilient to NAT timeout and rebinding.

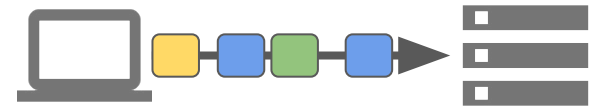


Overview: Stream multiplexing (1/2)

- Multiple streams within a single QUIC connection.
- A stream provides ordered delivery.
 - Similar to a single TCP connection.
- Identified by stream ID.
 - Odd IDs - client initiated; Even IDs - server initiated.
 - Can be unidirectional or bidirectional.



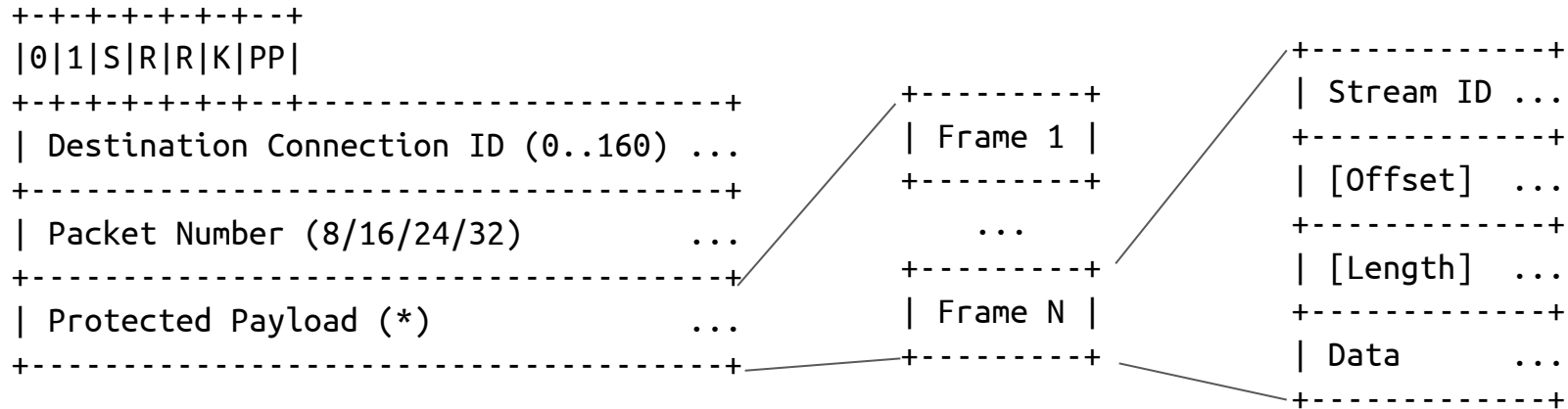
Multiple TCP connections



Single QUIC connection
(UDP-based)

Overview: Stream multiplexing (2/2)

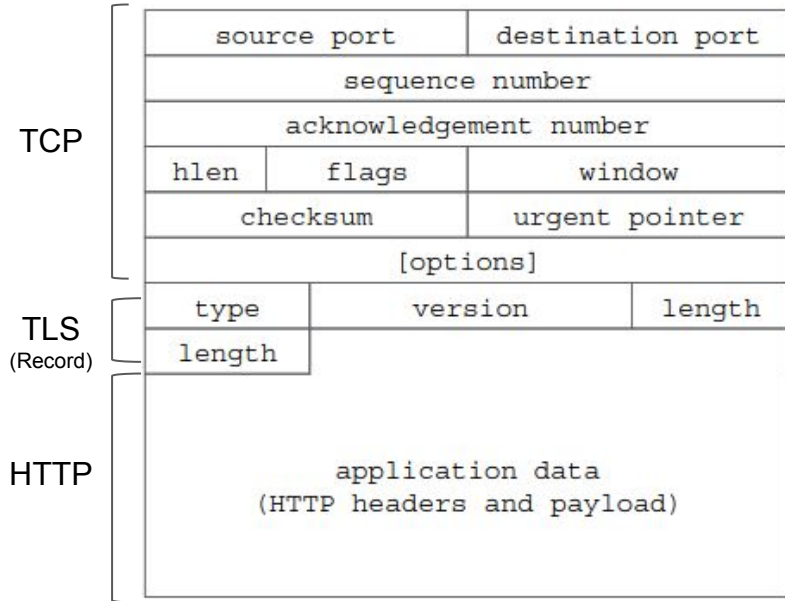
- QUIC packet contains multiple frames.



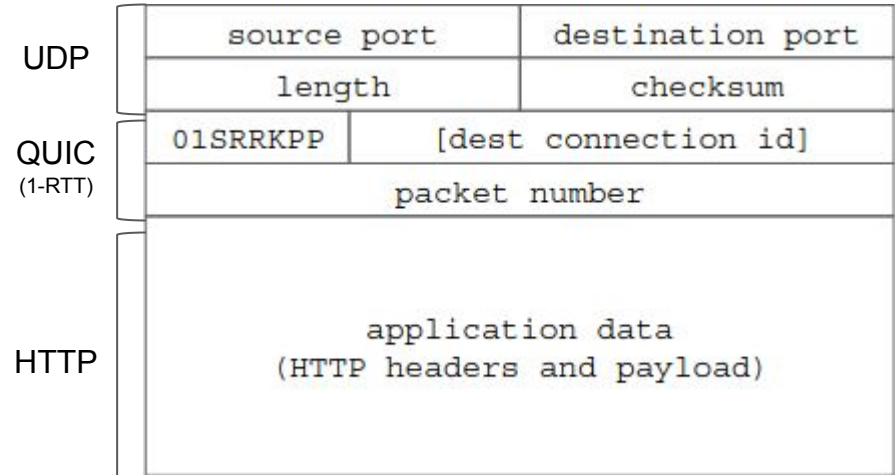
- Each frame has a type.
 - ACK, CRYPTO, STREAM, MAX_DATA, MAX_STREAM_DATA, ...

Overview: Authentication and encryption

TCP/TLS/HTTP

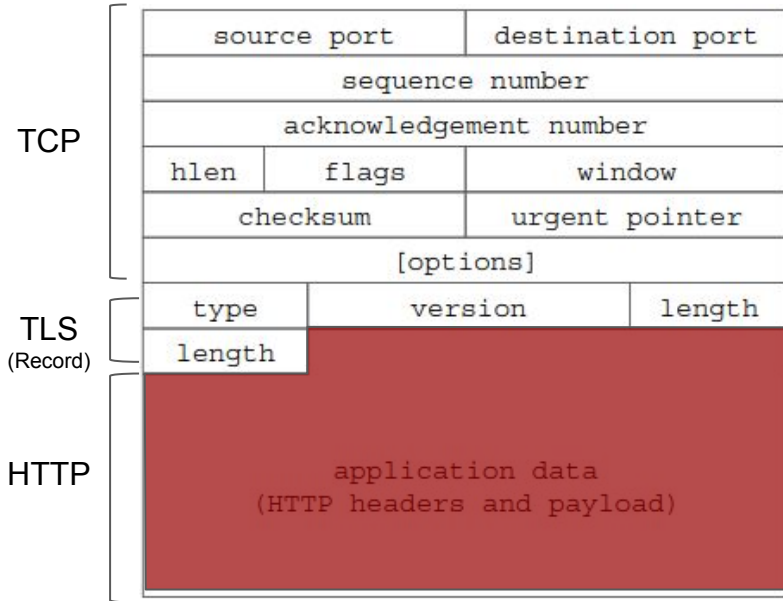


QUIC/HTTP

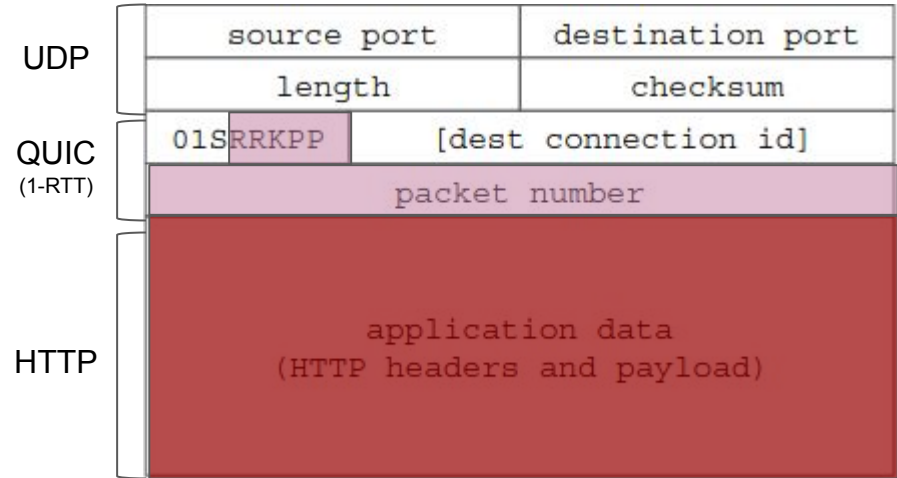


Overview: Authentication and encryption

TCP/TLS/HTTP

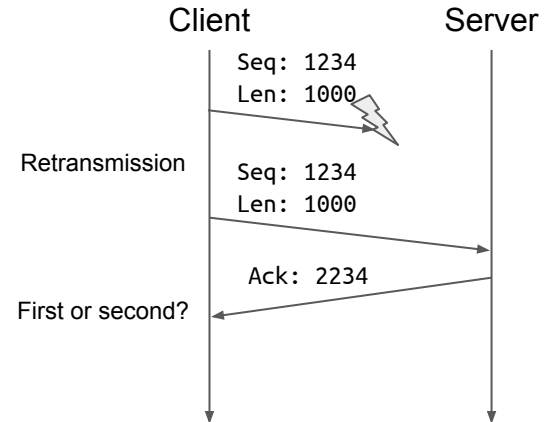


QUIC/HTTP



Overview: Loss recovery (1/2)

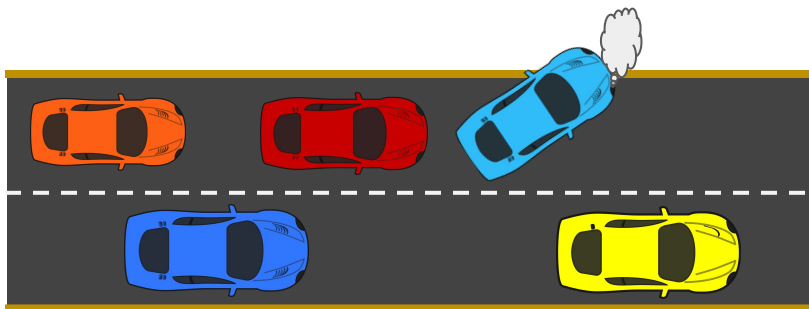
- QUIC uses monotonically increasing packet numbers
 - Unlike TCP which uses sequence numbers
- Packet numbers specify *transmission order*
 - Not delivery order
- Remedies TCP's *retransmission ambiguity* [3]



Overview: Loss recovery (2/2)

- No retransmission of the same QUIC packet
 - Lost frames are sent in a new packet
 - Gives flexibility to the packet scheduler

- Streams are independent w.r.t. ordering/retransmission
 - Addresses head-of-line blocking



Overview: Flow control

- At the stream level (`MAX_STREAM_DATA`)
 - Limits the number of bytes sent on a stream
- At the connection level (`MAX_DATA`)
 - Limits the number of bytes sent across all streams
- Controlling concurrency (`MAX_STREAMS`)
 - Limits the amount of streams



Overview: Congestion control

- Interface for incorporating different algorithms.
- In the spec: algorithm similar to TCP Reno.
- Accurate RTT estimate using ACK Delay.
 - Delay between receipt of a packet and the transmission of its ACK.
 - Useful for BBR.



Pluginized QUIC

What is Pluginized QUIC?

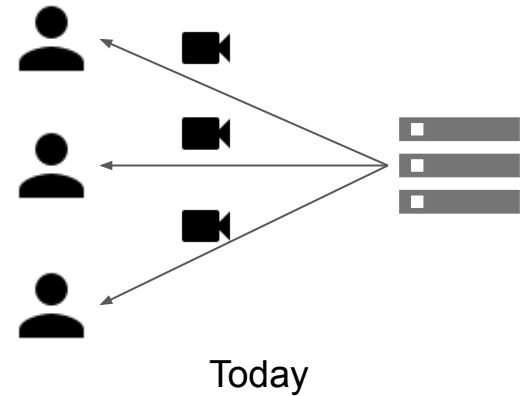
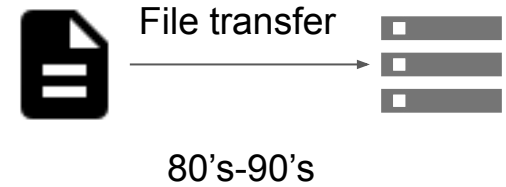
- A way to extend QUIC on a per-connection basis
- Endpoints dynamically exchange plugins and execute them

The logo for Pluginized QUIC (PQUIC) features the word "PQUIC" in a bold, sans-serif font. The letter "P" is colored green, while the letters "Q", "U", "I", and "C" are colored blue. The "Q" has a distinctive shape with a small loop at the bottom left.

<https://pquic.org/>

Protocols evolve

- Today's problems are not tomorrow's



Protocols evolve

- Today's problems are not tomorrow's
- Most protocols have an “options” field
 - To extend functionality
- Slow deployment
 - TCP SACK took nearly a decade to be adopted

An Analysis of Longitudinal TCP Passive Measurements (Short Paper)

Kensuke Fukuda

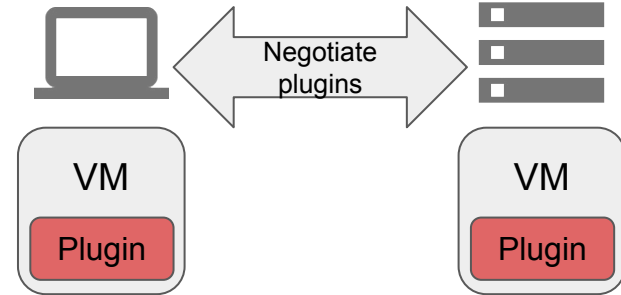
National Institute of Informatics,
Japan / PRESTO JST
kensuke@nii.ac.jp

Abstract. This paper reports on the longitudinal dynamics of TCP flows at an international backbone link over the period from 2001 to 2010. The dataset was a collection of traffic traces called MAWI data consisting of daily 15min pcap traffic trace measured at a trans-pacific link between Japan and the US. The environment of the measurement link has changed in several aspects (i.e., congestion, link upgrade, application). The main findings of the paper are as follows. (1) A comparison of the AS-level delays between 2001 and 2010 shows that the mean delay decreased in 55% of ASes, but the median value increased. Moreover, largely inefficient paths disappeared. (2) The deployment of TCP SACK increased from 10% to 90% over the course of 10 years. On the other hand, the window scale and timestamp options remained under-deployed (less than 50%).

2011

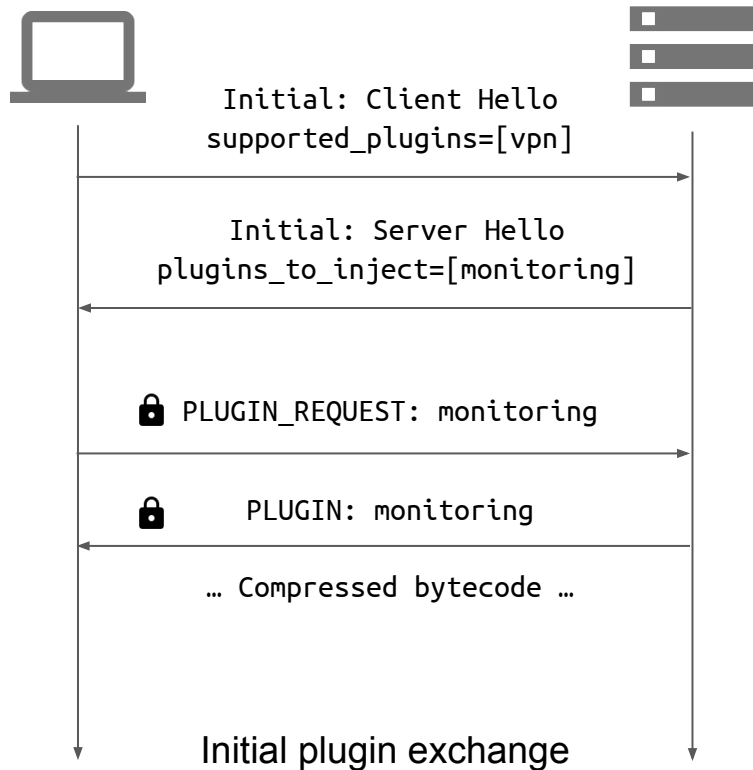
Idea: Extend via plugins

- Endpoints negotiate plugins on a per-connection basis
- Plugin's bytecode is injected to the peer
 - Modify behavior by attaching to well-defined locations
- Brings innovation to the transport layer
 - New congestion controller
 - New frames/transport parameters
 - Improved RTT calculation
 - ...



Plugins negotiation

- New QUIC transport parameters and frame types
- supported_plugins
 - Plugins a PQUIC peer can inject locally
- plugins_to_inject
 - Plugins that a PQUIC peer would like to communicate to the other PQUIC peer



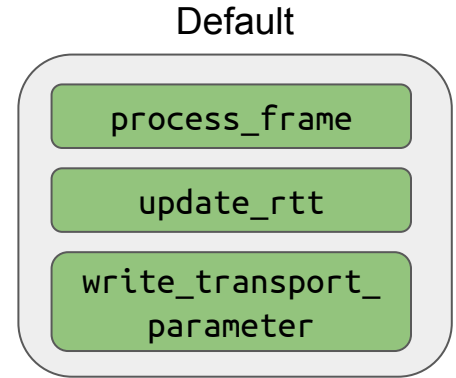
PQUIC Requirements

Given QUIC implementation, we need:

- Identifying protocol operations
- Running plugins in a safe environment
- Providing an API to the plugins
 - To modify connection state
 - Allocate memory
 - ...

Protocol operations

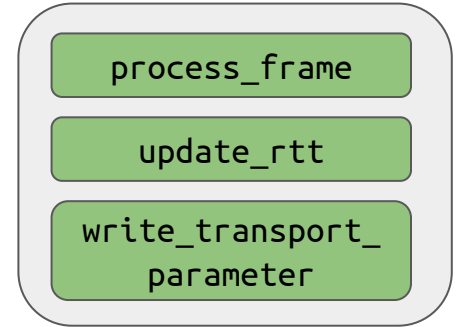
- Transport protocol: provides set of basic functions
 - Parsing and processing frames
 - PTO computation
 - Updating RTT
 - Remove acknowledged frames from the sending buffer
 - Add to send buffer
 -



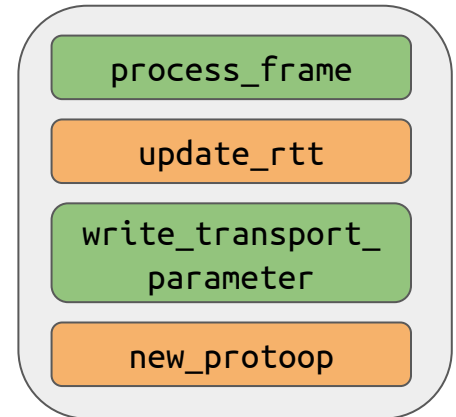
Protocol operations

- Transport protocol: provides set of basic functions
 - Parsing and processing frames
 - PTO computation
 - Updating RTT
 - Remove acknowledged frames from the sending buffer
 - Add to send buffer
 -
- Plugin: provides new set of protocol functions (modified or added)

Default




With plugin



Attaching to protocol operation (1/3)

```
protoop_arg_t
process_ack_frame_maybe_ecn(picoquic_cnx_t* cnx) {
    // ...
    picoquic_update_rtt(cnx, args...);
    // ...
}
```



```
picoquic_packet_t* picoquic_update_rtt(picoquic_cnx_t*
cnx, args...) {
    // PRE

    // REPLACE

    // POST
}
```

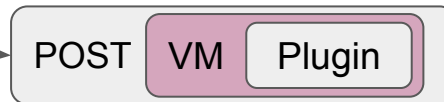
Attaching to protocol operation (2/3)

```
protoop_arg_t  
process_ack_frame_maybe_ecn(picoquic_cnx_t* cnx) {  
    // ...  
    picoquic_update_rtt(cnx, args...);  
    // ...  
}
```

```
picoquic_packet_t* picoquic_update_rtt(picoquic_cnx_t*  
cnx, args...) {
```



// REPLACE



Can read
connection state,
not write.

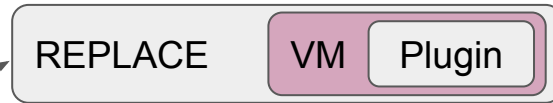
```
}
```

Attaching to protocol operation (3/3)

```
protoop_arg_t  
process_ack_frame_maybe_ecn(picoquic_cnx_t* cnx) {  
    // ...  
    picoquic_update_rtt(cnx, args...);  
    // ...  
}
```

```
picoquic_packet_t* picoquic_update_rtt(picoquic_cnx_t*  
cnx, args...) {  
    // PRE
```

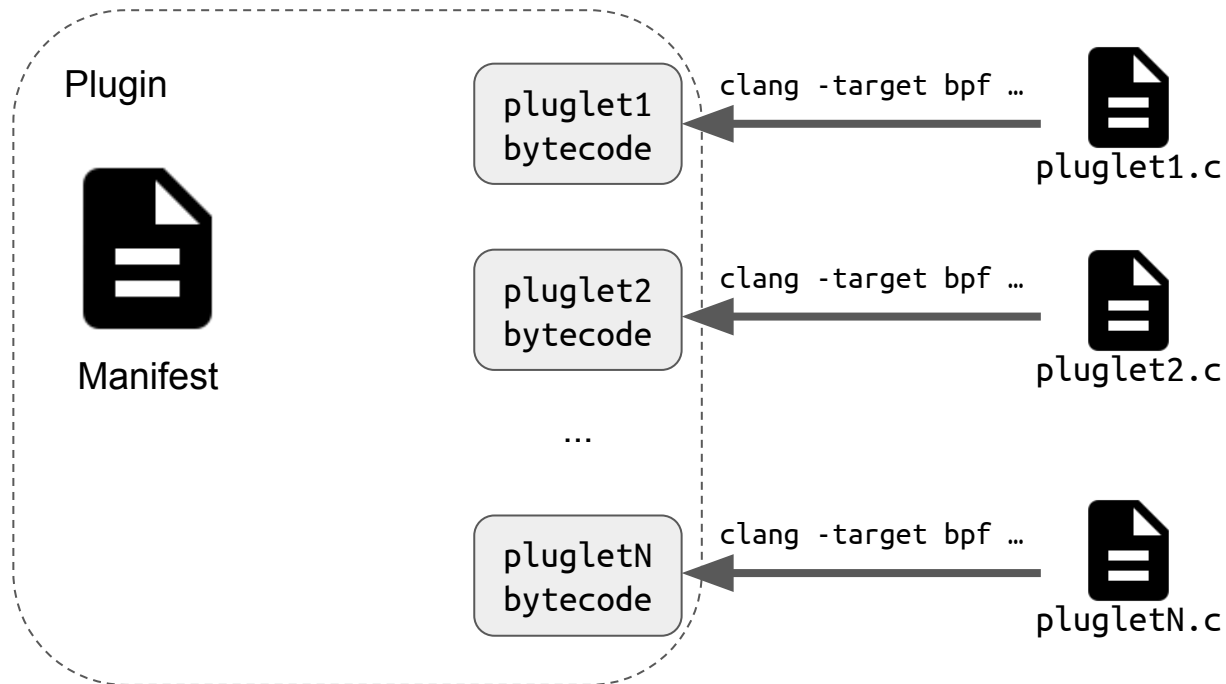
Can change
connection state.
Exclusive.



```
// POST  
}
```

Running plugins

- Plugin = manifest (where to attach) + collection of pluglets



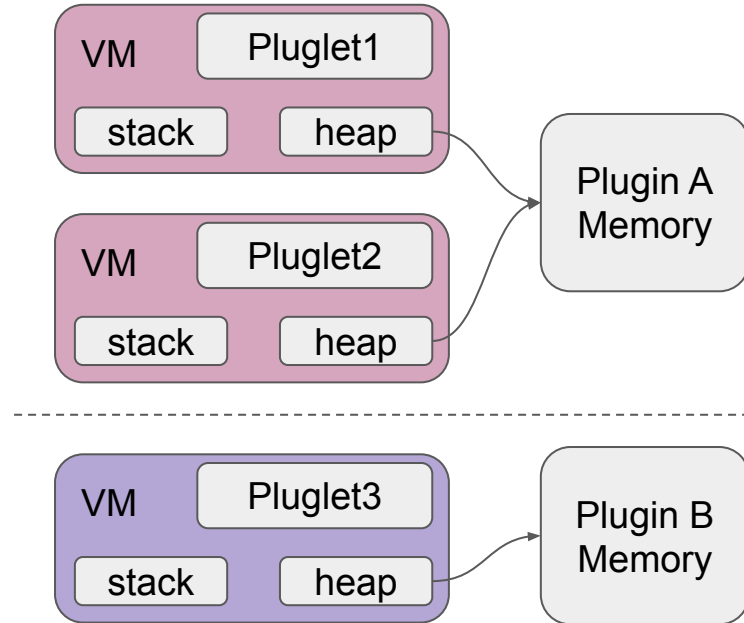
eBPF Virtual Machine

- Lightweight register-based virtual machine
 - Integrated in Linux kernel since 2014
 - RISC instruction set
- Just-in-time compilation
- Each pluglet runs in a Pluglet Runtime Environment (PRE)
 - Userspace eBPF virtual machine

```
$ sudo tcpdump -d "tcp port 0x1337"
(000) ldh    [12]
(001) jeq    #0x86dd    jt 2    jf 8
(002) ldb    [20]
(003) jeq    #0x6      jt 4    jf 19
(004) ldh    [54]
(005) jeq    #0x1337   jt 18   jf 6
(006) ldh    [56]
(007) jeq    #0x1337   jt 18   jf 19
(008) jeq    #0x800    jt 9    jf 19
(009) ldb    [23]
(010) jeq    #0x6      jt 11   jf 19
(011) ldh    [20]
(012) jset   #0x1fff    jt 19   jf 13
(013) ldx    4*([14]&0xf)
(014) ldh    [x + 14]
(015) jeq    #0x1337   jt 18   jf 16
(016) ldh    [x + 16]
(017) jeq    #0x1337   jt 18   jf 19
(018) ret    #262144
(019) ret    #0
```


Ensuring safe environment

- **Static verifier**
 - Looking for invalid opcodes/values
 - Requires exit instruction
- **Memory accesses are verified dynamically**
 - Access allowed only to plugin's stack and heap
- **Plugins are isolated**
 - Communication is allowed using a well-defined interface
 - `get()/set()` interface to read/write connection state



What about performance?

- JITed eBPF ~ 2x slower than native code
- `get()/set()` interface ~ 5x slower than direct access

Flexibility has its price!

Summary

- QUIC is a new transport protocol
 - Secure and reliable

- PQUIC dynamically extends QUIC via Plugins

References

- [1] Quentin De Coninck, François Michel, Maxime Piraux, Florentin Rochet, Thomas Given-Wilson, Axel Legay, Olivier Pereira, and Olivier Bonaventure. 2019. Pluginizing QUIC. In Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '19). Association for Computing Machinery, New York, NY, USA, 59–74. DOI:<https://doi.org/10.1145/3341302.3342078>
- [2] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasnic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, and Zhongyi Shi. 2017. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17). Association for Computing Machinery, New York, NY, USA, 183–196. DOI:<https://doi.org/10.1145/3098822.3098842>
- [3] P. Karn and C. Partridge. 1987. Improving round-trip time estimates in reliable transport protocols. SIGCOMM Comput. Commun. Rev. 17, 5 (Oct./Nov. 1987), 2–7. DOI:<https://doi.org/10.1145/55483.55484>
- [4] QUIC 101, David Schinazi, Chrome University. <https://youtu.be/dQ5AND4DPyU>

References

- [5] Does the QUIC handshake require compression to be fast? Patrick McManus, May 18, 2020
<https://www.fastly.com/blog/quic-handshake-tls-compression-certificates-extension-study>
- [6] Y. Cui, T. Li, C. Liu, X. Wang and M. Kühlewind, "Innovating Transport with QUIC: Design Approaches and Research Challenges," in IEEE Internet Computing, vol. 21, no. 2, pp. 72-76, Mar.-Apr. 2017, doi: 10.1109/MIC.2017.44.
- [7] Gagliardi E., Levillain O. (2020) Analysis of QUIC Session Establishment and Its Implementations. In: Laurent M., Giannetsos T. (eds) Information Security Theory and Practice. WISTP 2019. Lecture Notes in Computer Science, vol 12024. Springer, Cham.
https://doi.org/10.1007/978-3-030-41702-4_11
- [8] Understanding QUIC, Hunter Dellaverson, Tianxiang Li, Jana Iyengar, Lixia Zhang
<http://web.cs.ucla.edu/~lixia/papers/UnderstandQUIC.pdf>
- [9] ACM SIGCOMM 2020 Tutorial on the QUIC Protocol
<https://conferences.sigcomm.org/sigcomm/2020/tutorial-quic.html>

References

- [10] QUIC: A UDP-Based Multiplexed and Secure Transport
<https://www.ietf.org/archive/id/draft-ietf-quic-transport-34.txt>
- [11] Using TLS to Secure QUIC <https://www.ietf.org/archive/id/draft-ietf-quic-tls-34.txt>
- [12] QUIC Loss Detection and Congestion Control
<https://www.ietf.org/archive/id/draft-ietf-quic-recovery-34.txt>

Thanks for listening!