University of Glasgow

Improving Secure Connection Establishment

Networked Systems (H) – Lecture 4

2019-2020

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow | This work is licensed under the Creative Commons Attribution-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nd/4.0/ or send a letter to Creative Commons, PO Box 1866,

Lecture Outline

- Limitations of TLS v1.3
  - Slow Connection Establishment
  - Metadata Leakage
  - Protocol Ossification
- QUIC Transport Protocol
  - Performance, security, and avoiding ossification
  - Unified protocol handshake
  - Reliable multi-streaming transport

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Limitations of TLS v1.3

- Slow Connection Establishment
- Metadata Leakage
- Protocol Ossification

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Limitations of TLS v1.3



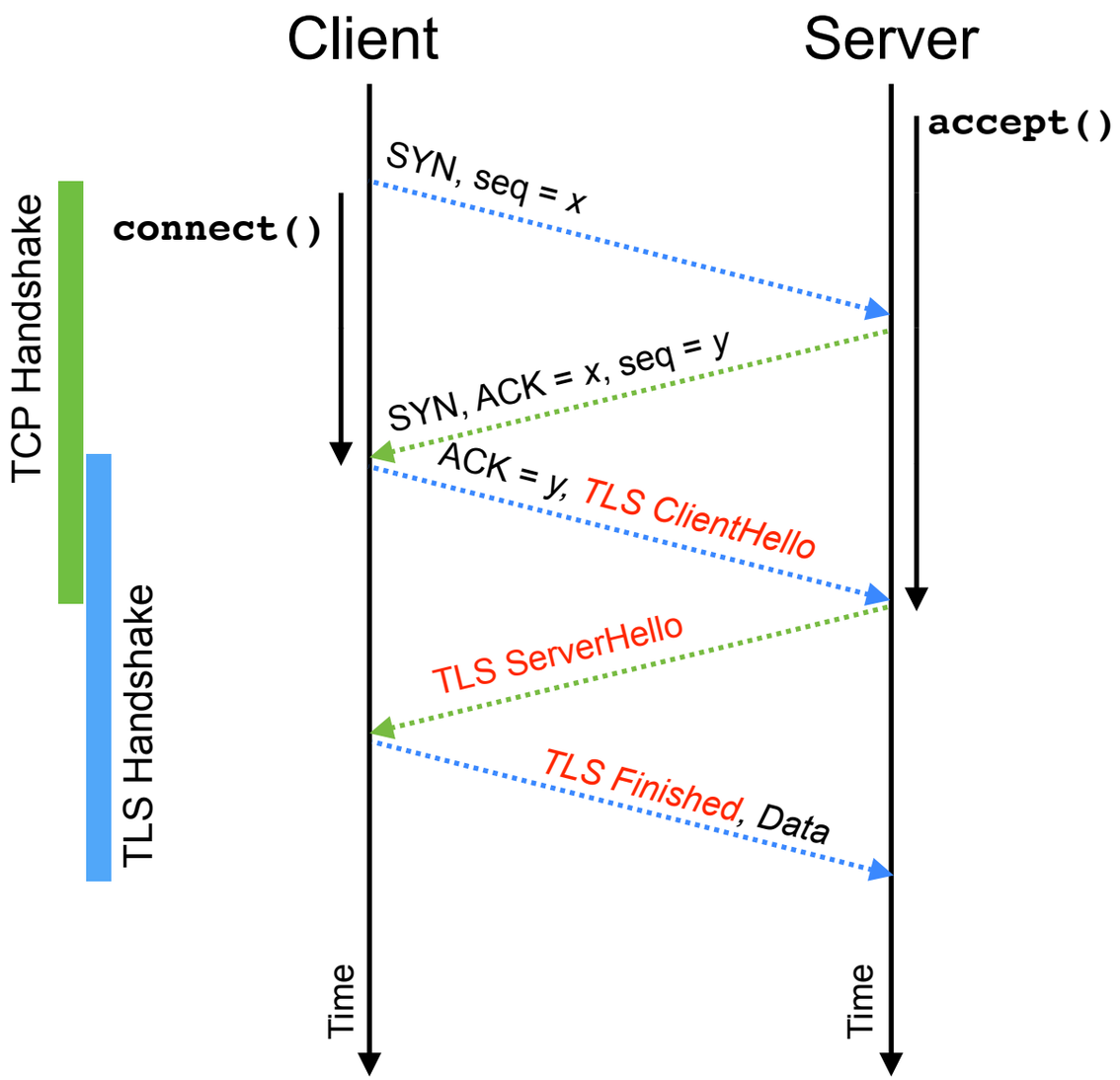
- TLS v1.3 is a tremendous success
  - Some **significant security improvements** when compared to TLS v1.2
    - Removed support for older and less secure encryption and key exchange algorithms
    - Removed support for secure algorithms that have proven difficult to implement correctly
  - Some **performance improvements** to the initial handshake and with 0-RTT mode
- Despite this, TLS v1.3 has some limitations that are hard to fix
  - Connection establishment is still relatively slow
  - Connection establishment leaks potentially sensitive metadata
  - The protocol is ossified due to middlebox interference

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

TLS v1.3 Connection Establishment Performance (1/2)

- TCP connection established as usual:
  - **`SYN`** → **`SYN+ACK`** → **`ACK`**
- TLS handshake protocol runs inside TCP connection:
  - TLS **`ClientHello`** sent with final **`ACK`**
  - TLS **`ServerHello`** sent in response
  - TLS **`Finished`** message concludes, and carries initial secure data record

- First data sent 2x RTT after connection establishment starts
- Earliest response received 3x RTT after connection establishment starts



Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

TLS v1.3 Connection Establishment Performance (2/2)

- Average web page comprises 1.7 MB of data, fetched as 69 HTTP requests, using 15 TCP connections
- 83% of HTTP requests run over TLS

  https://httparchive.org/reports/page-weight

- **Enormous amount of time wasted, waiting for TCP and TLS connection establishment handshakes**

Destination	Average RTT (ms)
London	72.5
New York	153.3
Los Angeles	221.2
Sydney	381.2

RTT measurements (ping times) from residential site in Glasgow

- Can we speed up TLS connection setup?
  - 0-RTT Connection Reestablishment – speed-up connections to known servers
  - Concurrent TCP and TLS handshake – speed-up connections to all servers

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

0-RTT Connection Reestablishment (1/4)

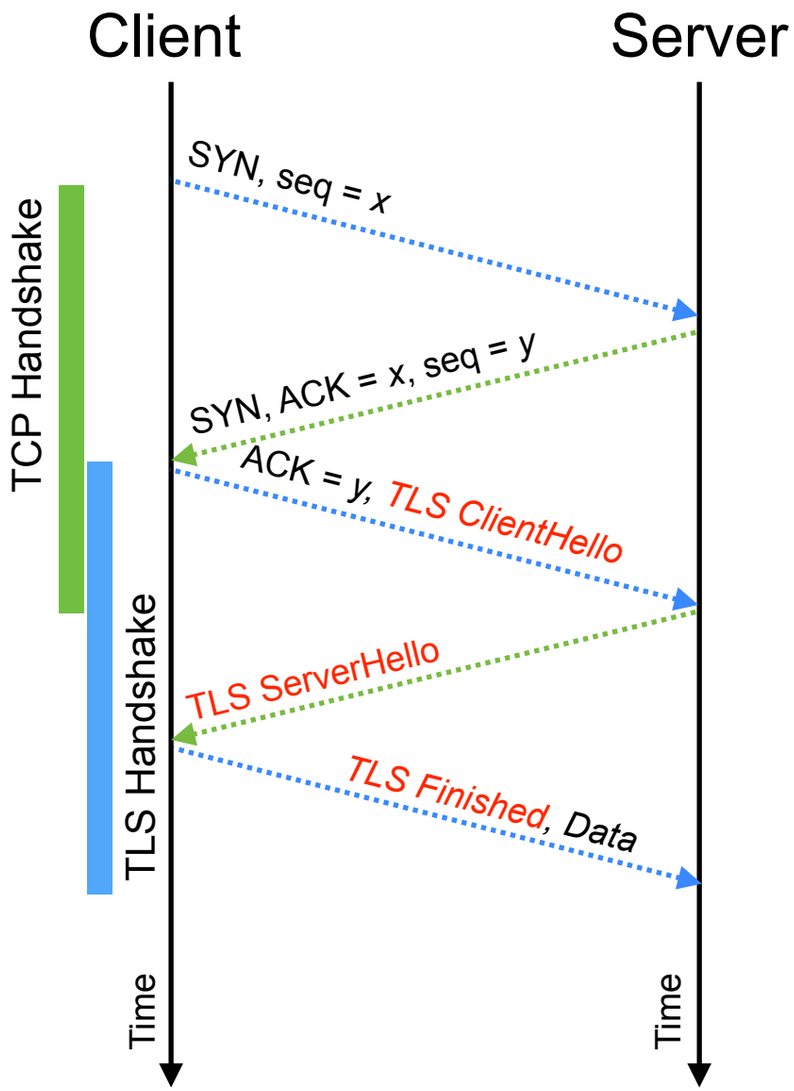
- Common to connect to a previously known TLS server – is it possible to shortcut the connection establishment in such cases?

- Need to understand:
  - What is the role of the TLS handshake?
  - How to encrypt initial data?
  - What are the potential risks?

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

0-RTT Connection Reestablishment (2/4)

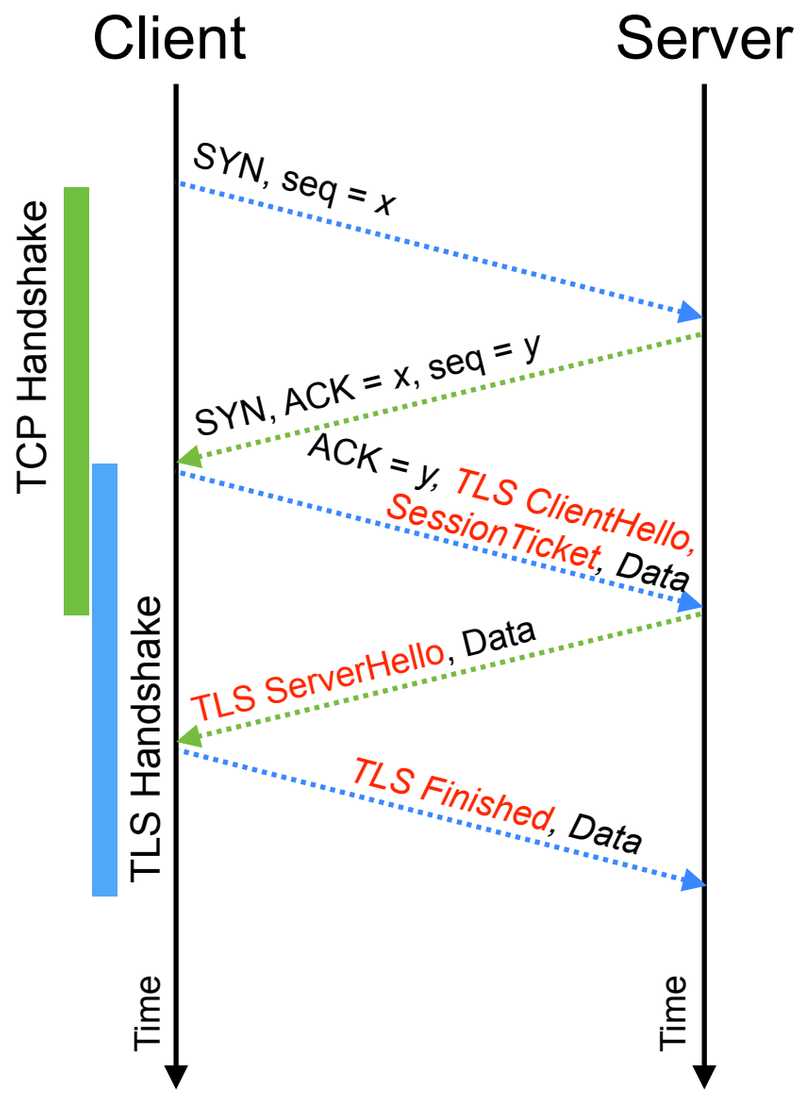
- **What is the role of the TLS handshake?**
  - Uses public key cryptographic techniques to establish an **ephemeral session key**, used to encrypt the data
    - The **`ClientHello`** and **`ServerHello`** are used to exchange material used to derive a session key – *using ECDHE key negotiation*
    - The session is ephemeral – different for each connection; derived from the public keys and a random value
    - The ephemeral session key provides **forward secrecy** – each connection has a unique key; if the encryption key for one session leaks, it doesn't help an attacker break other sessions
  - Retrieve the server's certificate, allowing the client to authenticate the server
    - The **`ServerHello`** contains the certificate



Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

0-RTT Connection Reestablishment (3/4)

- **How to encrypt initial data?**
  - Cannot negotiate ephemeral session key for initial data – relies on data exchanged in the handshake
  - Can reuse a **pre-shared key** agreed in a previous TLS session
- In a previous TLS connection:
  - Server sends the client a **`PreSharedKey`** along with **`SessionTicket`** to identify the key
- When reestablishing a connection:
  - In TCP segment with **`ClientHello`**, client sends the **`SessionTicket`** along with data encrypted using the corresponding **`PreSharedKey`**
  - The server uses the **`SessionTicket`** to look up the **`PreSharedKey`** and decrypt the data
  - The **`ClientHello`** and **`ServerHello`** complete a usual key exchange – data with **`ServerHello`** and later protected using ephemeral session key
  - 1-RTT due to TCP; zero RTTs added due to TLS



Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

0-RTT Connection Reestablishment (4/4)

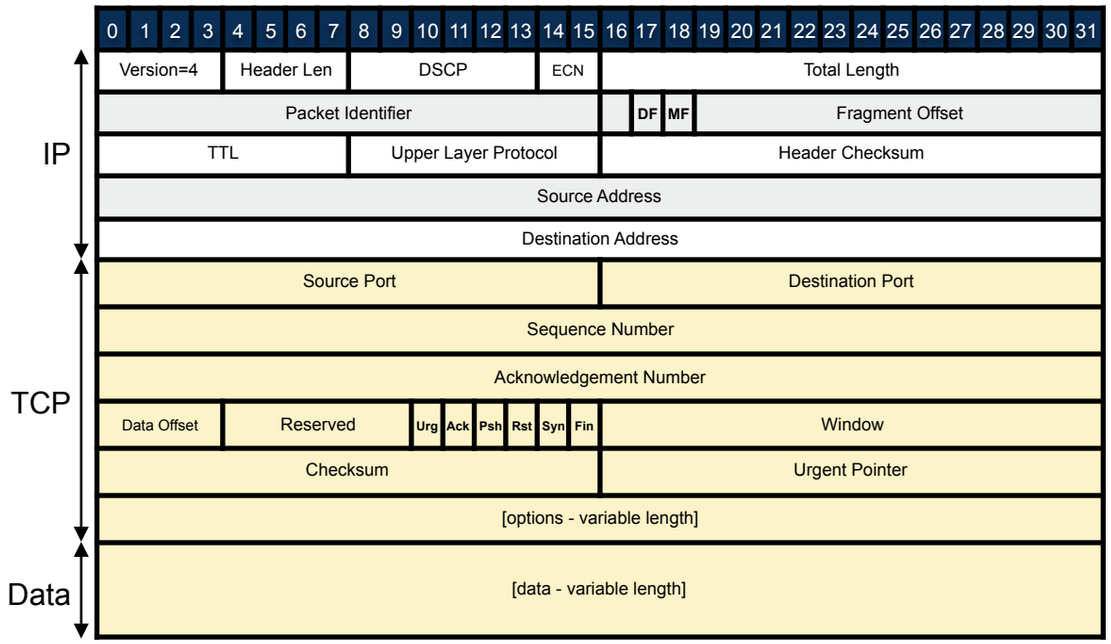
- **What are the potential risks?**
  - 0-RTT data sent with **`ClientHello`** using a **`PreSharedKey`** is not forward secret
    - Use of **`PreSharedKey`** links TLS connections – if session where **`PreSharedKey`** is distributed is compromised, 0-RTT data sent using that key in future connections will also be compromised
  - 0-RTT data sent with **`ClientHello`** using a **`PreSharedKey`** is subject to replay attack
    - The 0-RTT data is accepted during TLS connection establishment
    - If on-path attacker captures and replays the TCP segment with the **`ClientHello`**, **`SessionTicket`**, and data protected with the **`PreSharedKey`**, that data will be accepted by the server again
      - The server will respond to the replay, trying to complete the handshake – this might fail
      - But – by then, the data will have been accepted
  - Ensure 0-RTT data is **idempotent** to avoid this risk

- **Be very careful using 0-RTT data in TLS v1.3** – trades performance for safety

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

TLS v1.3 Metadata Leakage

- TLS operates within a TCP connection
  - TCP layer leaks port numbers, metadata about progress of connection
  - IP layer leaks IP addresses
  - **Are these a privacy risk?**
- When used with HTTP, **`ClientHello`** includes Server Name Indication (SNI) extension



- Identifies web site being requested – so the web server knows what public key to use in the **`ServerHello`**
  - Required to support shared hosting, with multiple websites on one server
  - Has to be unencrypted – sent before session keys are negotiated
  - Can't encrypted with **`PreSharedKey`**, since that's provided by server, and goal is to select the server
- Significant privacy concern with TLS v1.3
- See https://datatracker.ietf.org/doc/draft-ietf-tls-esni/ for work-in-progress attempt to resolve

CC BY ND

TLS v1.3 Protocol Ossification (1/3)

- TLS widely implemented, but many implementations are poor quality:
  - Some TLS servers drop connection on receipt of unknown TLS version number, rather than falling back to old version
  - Some firewalls block connections if the **`ClientHello`** is structured differently to that used by TLS 1.2 and earlier, even if TLS 1.3 is signalled



- **`ClientHello`** includes the latest TLS version supported by client
- **`ServerHello`** either accepts that, or replies with latest version it supports
- **`Finished`** either accepts that, or the connection fails

- Original design of TLS 1.3 changed **`ClientHello`**
  - Updated the version number
  - Removed some now unused header fields
- Measurements showed these changes caused ~8% of TLS 1.3 connections to fail → due to firewall bugs

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

TLS v1.3 Protocol Ossification (2/3)

- Final version of TLS 1.3 changed the design to work around these bugs
  - Version number in **`ClientHello`** says TLS **1.2**
  - Unused header fields are present, with dummy values
  - Add an extension header to **`ClientHello`**, to signal the real version number
  - (Similar changes in **`ServerHello`**)

- **Protocol ossification** is a significant concern
  - TLS is not the only protocol to include such workarounds
  - Widely deployed faulty implementations constrain design of most protocols



- **`ClientHello`** includes the latest TLS version supported by client
- **`ServerHello`** either accepts that, or replies with latest version it supports
- **`Finished`** either accepts that, or the connection fails

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

TLS v1.3 Protocol Ossification (3/3)

- How to avoid protocol ossification?
  - Ossification happens when extension mechanisms, or allowed flexibility, are not used
    - TLS 1.3 was released ten years after TLS 1.2
    - Allowed products to be built and deployed that didn't do version negotiation correctly, since no new versions to negotiate
    - Allowed products to be built that relied on the presence and order of fields in ClientHello, since all implementations included the same fields in the same order
  - **Generate Random Extensions And Sustain Extensibility** (GREASE)
    https://datatracker.ietf.org/doc/rfc8701/
    - If the protocol allows fields to be sent if different orders, make sure this happens
    - If the protocol allows extensions, make sure to send extensions sometimes
    - If the protocol allows for different versions, negotiate different versions sometimes
    - *Do this even if the extensions or versions will be ignored or fall back to the older version*
    - The "use it or lose it" approach to avoiding ossification

cc BY ND

Discussion

- TLS v1.3 is a significant improvement on prior versions:
  - Faster
  - More secure

- But – still runs within a TCP connection:
  - Has to wait for TCP connection establishment
  - Has to suffer TCP ossification and metadata leakage
- Still suffers from ossification at TLS level

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

QUIC Transport Protocol

- Performance, security, and avoiding ossification
- Unified protocol handshake
- Reliable multi-streaming transport

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

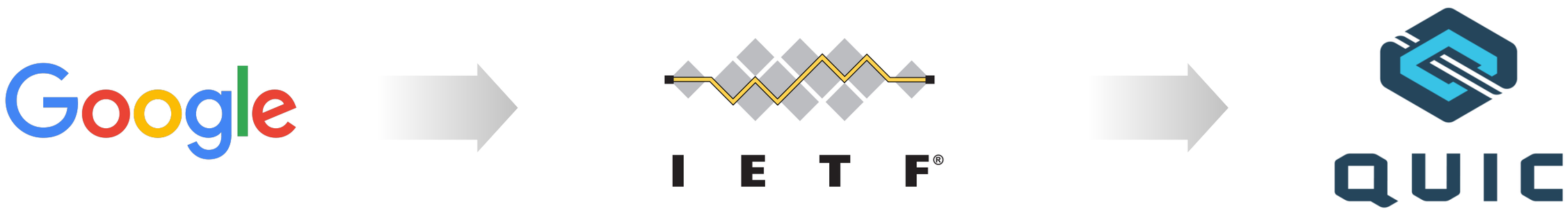
QUIC: Performance, Security, and Avoiding Ossification

- **What is wrong with TLS v1.3 over TCP?**
  - Slow to connect – due to sequential TCP and TLS handshakes
  - Protocol is ossified and hard to extend – due to widely deployed poor quality implementations; TCP middleboxes
  - Protocol leaks metadata – hard to fix, due to ossification

- Solution: replace TLS v1.3 over TCP with a unified secure transport protocol → **QUIC**
  - Avoid ossification via systematic application of GREASE and encryption
  - Avoid metadata leakage via pervasive encryption
  - Reduce latency by overlapping TLS and transport handshake

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

History of QUIC



- 2012-2015: Initial proposal and experimental deployment by Google
- 2016-2020: IETF QUIC working group
  - https://datatracker.ietf.org/wg/quic/about/
  - https://quicwg.org
  - https://github.com/quicwg
  - As of January 2020, specifications have gone through 26 iterations in IETF – standardisation led to **significant differences and improvements compared to initial Google proposal**
  - Expect Internet standards track RFCs documenting QUIC to be published in late 2020

CC BY ND

QUIC Overview (1/2)

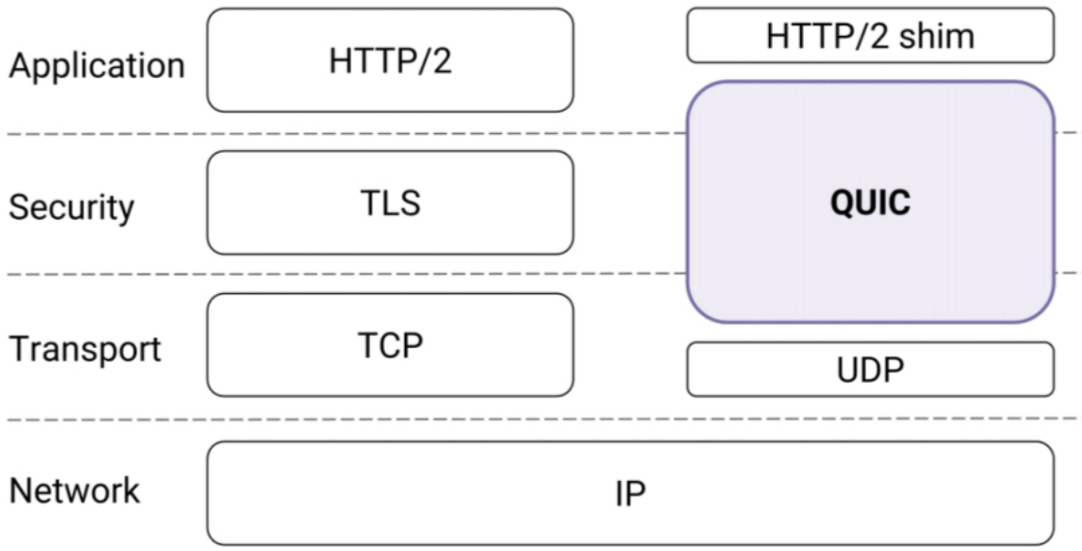


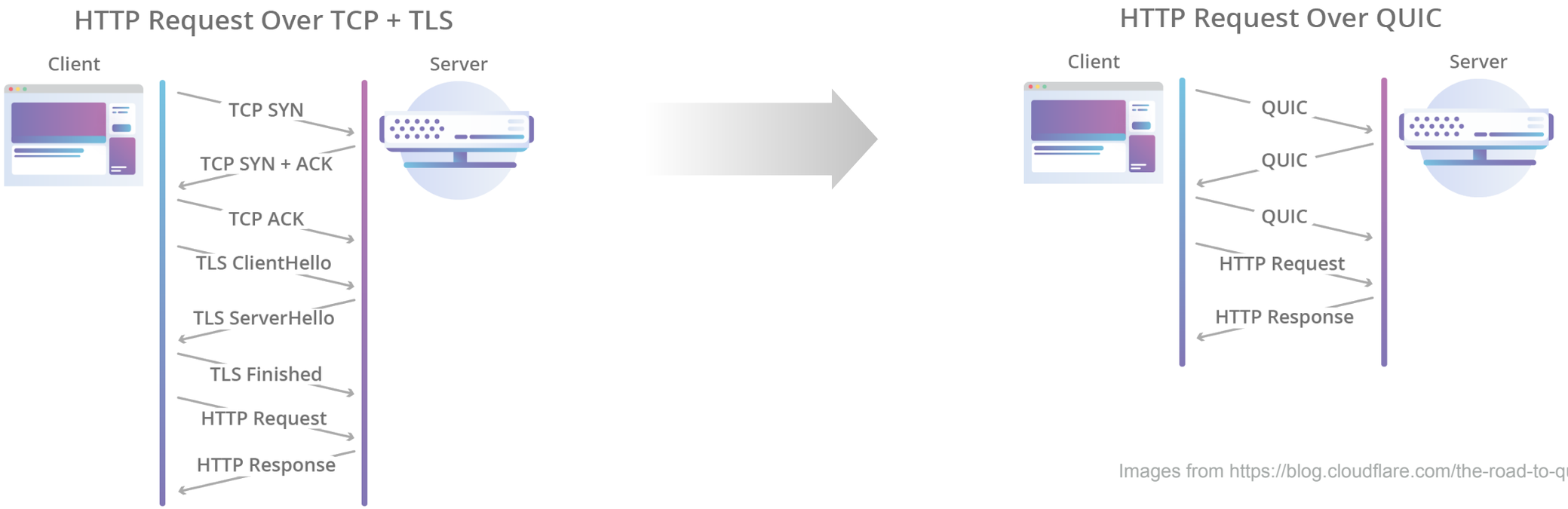
Figure 1: QUIC in the traditional HTTPS stack.

Langley, A., Riddoch, A., Wilk, A., Vicente, A., Krasic, C., Zhang, D., Yang, F., Koura- nov, F., Swett, I., Iyengar, J., Bailey, J., Dorfman, J., Roskind, J., Kulik, J., Westin, P., Tenneti, R., Shade, R., Hamilton, R., Vasiliev, V., Chang, W.-T., and Shi, Z. The QUIC transport protocol: Design and Internet-scale deployment. In Proceedings of the SIGCOMM Conference (Los Angeles, CA, USA, Aug. 2017), ACM. https://dl.acm.org/authorize.cfm?key=N33907

- QUIC replaces TCP, TLS, and some parts of HTTP
  - TCP → reliable, ordered, byte streams; congestion control
  - TLS → security
  - HTTP → sub-stream multiplexing over the connection
- Runs on UDP to support deployment
- Encrypts entire payload and most of the transport headers
- Typically implemented in user-space

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

QUIC Overview (2/2)



Images from https://blog.cloudflare.com/the-road-to-quic/

- A TLS v1.3 over TCP connection:
  - Performs the TCP three-way handshake to negotiate transport parameters; then
  - Performs the TLS v1.3 three-way handshake to negotiate security parameters
- A QUIC connection:
  - Performs a *single* three-way handshake to negotiate transport and security parameters

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

The QUIC Transport Protocol

- Key points to the design of QUIC:
  - Operation over UDP
  - Invariants and avoiding ossification
  - Headers and frame structure
  - Long-header packets and the initial handshake
  - Security
- Short-header packets and ongoing data transfer
  - Streams and framing
  - Head-of-line blocking
  - Congestion Control

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

QUIC over UDP

- **Why run QUIC over UDP rather than over IP?**
- To ease end-system deployment in user-space applications
  - User-space applications, running over UDP, are easy to build
    - BSD sockets; portable → same API works everywhere
    - Widely understood programming model
    - No need for privileged access
  - No portable, unprivileged, interface to build applications that run directly over IP
    - Implementations have to run within the operating system kernel
    - Deploying kernel updates is difficult
    - Deploying application updates is straightforward
- To avoid protocol ossification due to middlebox interference
  - Firewalls block anything other than TCP and UDP



This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/COMST.2016.2626780, I Communications Surveys & Tutorials

De-ossifying the Internet Transport Layer: A Survey and Future Perspectives

Giorgos Papastergiou, Gorry Fairhurst, David Ros, Anna Brunstrom, Karl-Johan Grinnemo, Per Hurtig, Naeem Khademi, Michael Tüxen, Michael Welzl, Dragana Damjanovic and Simone Mangiante

***Abstract*—It is widely recognized that the Internet transport layer has become ossified, where further evolution has become hard or even impossible. This is a direct consequence of the ubiquitous deployment of middleboxes that hamper the deployment of new transports, aggravated further by the limited flexibility of the Application Programming Interface (API) typically presented to applications. To tackle this problem, a wide range of solutions have been proposed in the literature, each aiming to address a particular aspect. Yet, no single proposal has emerged that is able to enable evolution of the transport layer. In this work, after an overview of the main issues and reasons for transport-layer ossification, we survey proposed solutions and discuss their potential and limitations. The survey is divided into five parts, each covering a set of point solutions for a different facet of the problem space: 1) designing middlebox-proof transports, 2) signaling for facilitating middlebox traversal, 3) enhancing the API between the applications and the transport layer, 4) discovering and exploiting end-to-end capabilities, and 5) enabling user-space protocol stacks. Based on this analysis, we then identify further development needs towards an overall solution. We argue that the development of a comprehensive transport layer framework, able to facilitate the integration and cooperation of specialized solutions in an application-independent and flexible way, is a necessary step toward making the Internet transport architecture truly evolvable. To this end, we identify the requirements for such a framework and provide insights for its development.**

***Index Terms*—Transport protocols, protocol-stack ossification, API, middleboxes, user-space networking stacks.**

I. Introduction and background

Networks can and do vary significantly in the set of functions they offer and their ability to move data between endpoints. The transport layer operates across the network and is responsible for efficient and robust end-to-end communication between network endpoints. The term end-to-end is often associated with a principle, called the end-to-end argument [1]. This suggests that "functions placed at low levels of a system may be redundant, or of little value, when compared with the cost of providing them at that low level." This argument followed Schroeder et al.'s earlier work on system design and security [2], and is now generally considered as a simple guide for which services should be realised at the transport layer.

The transport layer was designed to hide the details and variability of the network service from the applications that need to use it. The Internet's transport layer also contains other functions that are difficult or impossible to provide within a network, such as reliability, verification of delivery, flow control to prevent the application from overwhelming the remote endpoint, congestion control to prevent the application from overwhelming the network, etc. People using the Internet mostly run applications that are based on the Transmission Control Protocol, TCP [3], which provides these transport functions.

Some applications need a different set of services to those offered by TCP. For example, a web client may wish to be able to prioritize sub-flows carrying specific objects, a multimedia flow may prefer timeliness to reliable delivery, and IP telephony can be tolerant to packet loss or in some cases to bit errors. There are many cases where TCP simply does not meet the need of applications—yet it ends up being used because it "just works," but not necessarily very well [4]. Applications that do not want the transport semantics of TCP typically just use the User Datagram Protocol, UDP [5]. While UDP provides flexibility that allows any set of services to be defined, every function needed has to be implemented at the application layer.

Some initiatives have developed alternate protocols to TCP, suited for other application types, for instance: the Datagram Congestion Control Protocol (DCCP) [6] was proposed to support streaming multimedia; the Stream Control Transport Protocol (SCTP) [7], [8] originally targeted telephony signaling; UDP-Lite [9] supports error-tolerant audio and video services over wireless links. However, despite being standardized, with available implementations for common platforms, these transports are seldom seen in the general Internet, and TCP and UDP remain the only widely used transports.

A. Transport-layer ossification: overview of issues

Why do developers and users not adopt more modern protocols? It is not because new transports do not meet a real need. The following paragraphs examine the main reasons for this *ossification* of the transport layer.

G. Papastergiou and D. Ros are with Simula Research Laboratory, P.O. Box 134, 1325 Lysaker, Norway. E-mail: {gpapaste,dros}@simula.no.
G. Fairhurst is with University of Aberdeen, AB24 3UE Aberdeen, United Kingdom. E-mail: gorry@erg.abdn.ac.uk.
A. Brunstrom, K.-J. Grinnemo and P. Hurtig are with Karlstad University, Universitetsgatan 2, 651 88 Karlstad, Sweden. E-mail: {anna.brunstrom,karl-johan.grinnemo,per.hurtig}@kau.se.
N. Khademi and M. Welzl are with University of Oslo, P.O. Box 1072 Blindern, 0316 Oslo, Norway. E-mail: {naeemk,michawe}@ifi.uio.no.
M. Tüxen is with Münster University of Applied Sciences, Biomatckstraße 11, 48565 Steinfurt, Germany. E-mail: tuexen@fh-muenster.de.
D. Damjanovic is with Mozilla Corporation. E-mail: ddamjanovic@mozilla.com.
S. Mangiante is with EMC Corporation, Ovens, Co. Cork, Ireland. E-mail: simone.mangiante@emc.com.

1553-877X (c) 2016 IEEE. Translations and content mining are permitted for academic research only. Personal use is also permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

G. Papastergiou, G. Fairhurst, D. Ros, A. Brunstrom, K.-J. Grinnemo, P. Hurtig, N. Khademi, M. Tüxen, M. Welzl, D. Damjanovic, and S. Mangianti, "De-ossifying the Internet transport layer: A survey and future perspectives", IEEE Communications Surveys and Tutorials (Nov. 2016). https://dx.doi.org/10.1109/COMST.2016.2626780/

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Avoiding Ossification (1/3)

- Deployment experience → if a field in a network protocol is visible to the network, someone will implement a middlebox that relies on its presence
- Once a protocol has been widely deployed, very hard to change



Is it Still Possible to Extend TCP?

Michio Honda[*], Yoshifumi Nishida[*], Costin Raiciu[†], Adam Greenhalgh[‡], Mark Handley[‡], Hideyuki Tokuda[*]
Keio University[*], Universitatea Politehnica Bucuresti[†], University College London[‡]
{micchie,nishida}@sfc.wide.ad.jp, costin.raiciu@cs.pub.ro
{a.greenhalgh,m.handley}@cs.ucl.ac.uk, hxt@ht.sfc.keio.ac.jp

ABSTRACT

We've known for a while that the Internet has ossified as a result of the race to optimize existing applications or enhance security. NATs, performance-enhancing-proxies, firewalls and traffic normalizers are only a few of the middleboxes that are deployed in the network and look beyond the IP header to do their job. IP itself can't be extended because "IP options are not an option" [10]. Is the same true for TCP?

In this paper we develop a measurement methodology for evaluating middlebox behavior relating to TCP extensions and present the results of measurements conducted from multiple vantage points. The short answer is that we can still extend TCP, but extensions' design is very constrained as it needs to take into account prevalent middlebox behaviors. For instance, absolute sequence numbers cannot be embedded in options, as middleboxes can rewrite ISN and preserve undefined options. Sequence numbering also must be consistent for a TCP connection, because many middleboxes only allow through contiguous flows.

We used these findings to analyze three proposed extensions to TCP. We find that MPTCP is likely to work correctly in the Internet or fallback to regular TCP. TcpCrypt seems ready to be deployed, however it is fragile if resegmentation does happen—for instance with hardware offload. Finally, TCP extended options in its current form is not safe to deploy.

Categories and Subject Descriptors

C.2.2 [**Computer-communication Networks**]: Network Protocols; C.2.6 [**Computer-communication Networks**]: Internetworking

General Terms

Measurement, Design, Experimentation, Standardization

Keywords

Middleboxes, Measurements, TCP, Protocol design

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
IMC'11, November 2–4, 2011, Berlin, Germany.
Copyright 2011 ACM 978-1-4503-1013-0/11/11 ..$10.00.

1. INTRODUCTION

The Internet was designed to be extensible; routers only care about IP headers, not what the packets contain, and protocols such as IP and TCP were designed with options fields that could be used to add additional functionality. The great virtue of the Internet was always that it was *stupid*; it did no task especially well, but it was extremely flexible and general, allowing a proliferation of protocols and applications that the original designers could never have foreseen.

Unfortunately the Internet, as it is deployed, is no longer the Internet as it was designed. IP options have been unusable for twenty years[10] as they cause routers to process packets on their slow path. Above IP, the Internet has benefited (or suffered, depending on your viewpoint) from decades of optimizations and security enhancements. To improve performance [2, 7, 18, 3], reduce security exposure [15, 29], enhance control, and work around address space shortages [22], the Internet has experienced an invasion of middleboxes that *do* care about what the packets contain, and perform processing at layer 4 or higher *within* the network.

The problem now faced by designers of new protocols is that there is no longer a well defined or understood way to extend network functionality, short of implementing everything over HTTP[25]. Recently we have been working on adding both multipath support[11] and native encryption[5] to TCP. The obvious way to do this, in both cases, is to use TCP options. In the case of multipath, we would also like to stripe data across more than one path. At the end systems, the protocol design issues were mostly conventional. However, it became increasingly clear that no one, not the IETF, not the network operators, and not the OS vendors, knew what will and what will not pass through all the middleboxes as they are currently deployed and configured. Will TCP options pass unchanged? If the sequence space has holes, what happens? If a retransmission has different data than the original, which arrives? Are TCP segments coalesced or split? These and many more questions are crucial to answer if protocol designers are to extend TCP in a deployable way. Or have we already lost the ability to extend TCP, just like we did two decades ago for IP?

In this paper we present the results from a measurement study conducted from 142 networks in 24 countries, including cellular, WiFi and wired networks, public and private networks, residential, commercial and academic networks. We actively probe the network to elicit middlebox responses that violate the end-to-end transparency of the original Internet architecture. We focus on TCP, not only because it is by far the most widely used transport protocol, but also because while it is known that many middleboxes modify TCP behavior [6], it is not known how prevalent such middleboxes are, nor precisely what the *emergent behavior* is with TCP extensions that were unforeseen by the middlebox designers.

Table 3: Unknown Option in Syn

Observed Behavior	TCP Port 34343	80	443
Passed	129 (96%)	122 (86%)	133(94%)
Removed	6 (4%)	20 (14%)	9 (6%)
Changed	0 (0%)	0 (0%)	0 (0%)
Error	0 (0%)	0 (0%)	0 (0%)
Total	135 (100%)	142 (100%)	142 (100%)

Table 4: Known Option in Data

Observed Behavior	TCP Port 34343	80	443
Passed	129 (96%)	122 (86%)	133 (94%)
Removed	6 (4%)	9 (6%)	6 (4%)
Changed	0 (0%)	4 (3%)	3 (2%)
Error	0 (0%)	7 (5%)	0 (0%)
Total	135 (100%)	142 (100%)	142 (100%)

Table 5: Unknown Option in Data

Observed Behavior	TCP Port 34343	80	443
Passed	129 (96%)	122 (86%)	133(94%)
Removed	6 (4%)	13 (9%)	9 (6%)
Changed	0 (0%)	0 (0%)	0 (0%)
Error	0 (0%)	7 (5%)	0 (0%)
Total	135 (100%)	142 (100%)	142 (100%)

Table shows interference with TCP connections by middleboxes

In 2011, 15% of TCP connections to port 80 went via middleboxes that removed TCP options

M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is it still possible to extend TCP?", In Proc. ACM Internet Measurement Conference, Berlin, Nov. 2011. https://dx.doi.org/10.1145/2068816.2068834

CC BY ND

Avoiding Ossification (2/3)

- Protocol ossification affected the design of:
  - TLS 1.3
  - Multipath TCP
  - TCP Fast Open
  - TCP Selective Acknowledgements
  - …
- Increasingly viewed as a problem in the standards community
  - Difficult to evolve network protocols to address new requirements
  - A system that can no longer evolve and change will die

However, in other cases, ossification can have unwanted outcomes. Ossification can frustrate the evolution of a transport protocol. A mechanism implemented in a network device, such as a firewall, that requires a header field to have only a specific known set of values can prevent the device from forwarding packets using a different version of the protocol that introduces a feature that changes to a new value for the observed field.

An example of this type ossification was observed in the development of Transport Layer Security (TLS) 1.3 [RFC8446], where the design needed to function in the presence of deployed middleboxes that relied on the presence of certain header fields exposed in TLS 1.2. The design of MPTCP also had to be revised to account for middleboxes (known as "TCP Normalizers") that monitor the evolution of the window advertised in the TCP header and then reset connections when the window did not grow as expected. Similarly, issues have been reported using TCP. For example, TCP Fast Open can experience middleboxes that modify the transport header of packets by removing "unknown" TCP options, segments with unrecognised TCP options can be dropped, segments that contain data and set the SYN bit can be dropped, or middleboxes that disrupt connections that send data before completion of the three-way handshake.

Fairhurst & Perkins Expires August 3, 2020 [Page 6]

Internet-Draft Transport Header Encryption January 2020

Other examples of ossification have included middleboxes that modify transport headers by rewriting TCP sequence and acknowledgement numbers, but are unaware of the (newer) TCP selective acknowledgement (SACK) Option and therefore fail to correctly rewrite the selective acknowledgement header information to match the changes that were made to the fixed TCP header, preventing SACK from operating correctly.

In all these cases, middleboxes with a hard-coded, but incomplete, understanding of transport behaviour, interacted poorly with transport protocols after the transport behaviour was changed.

Gorry Fairhurst and Colin Perkins, Considerations around Transport Header Confidentiality, Network Operations, and the Evolution of Internet Transport Protocols (.txt|.pdf), Internet Engineering Task Force, January 2020, Work in progress, https://datatracker.ietf.org/doc/draft-ietf-tsvwg-transport-encrypt/

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Avoiding Ossification (3/3)

- Three tools to prevent ossification of QUIC:
  - Published protocol invariants
  - Pervasive encryption of transport headers
  - GREASE

- QUIC is a new design – want to **avoid ossification**, starting with the initial deployments
  - Engineer the protocol to make it difficult, ideally impossible, for middleboxes to interfere with QUIC connections

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

QUIC Invariants

- IETF listed properties of QUIC that are expected to remain unchanged as new versions of the protocol of developed
  - Explicit guidance to middlebox developers: you may assume these properties of QUIC will not change
  - Corollary: IETF **will** change other fields or properties between QUIC versions
- What is guaranteed?
  - Packets will start with a long- or short-header, determined by value of first bit
  - Long header packets include version number, destination- and source-connection identifiers
  - Short header packets include the destination connection identifier
  - Long- and short-header packets may be sent arbitrarily within a connection

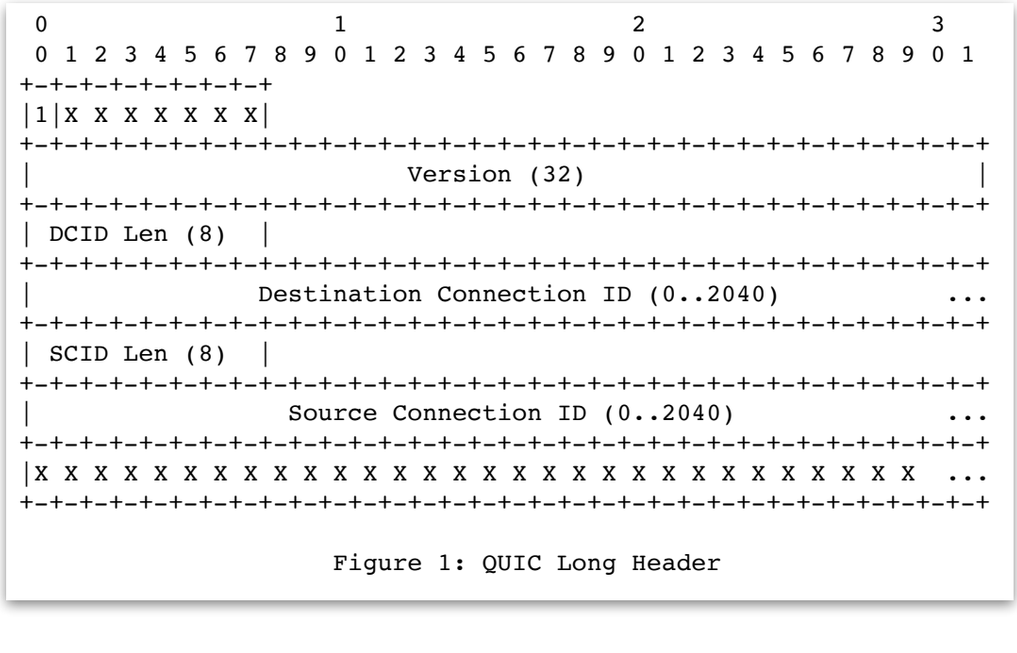


Figure 1: QUIC Long Header

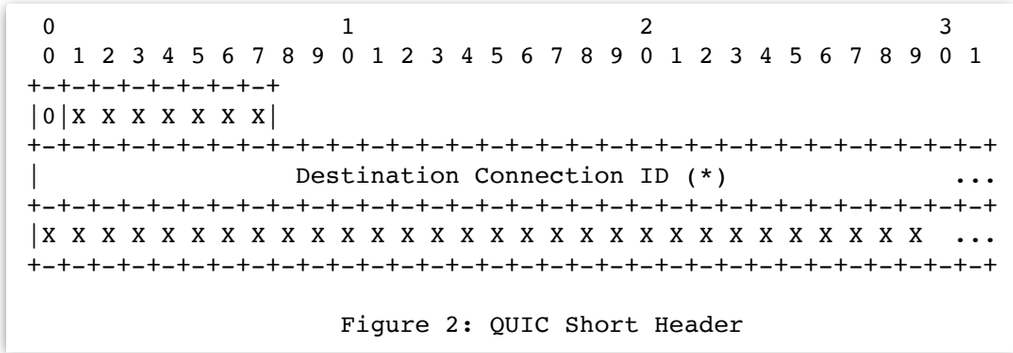


Figure 2: QUIC Short Header

https://datatracker.ietf.org/doc/draft-ietf-quic-invariants/

X = bit may take arbitrary value

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Avoiding Ossification via Pervasive Encryption

- QUIC encrypts as much data as possible
  - Entire packet *except* invariant fields and the last 7-bits of the first byte is encrypted
  - Entire packet is authenticated
- Encryption → most headers unreadable
  - Encryption keys for initial handshake derived from Connection ID
    - **These** messages unencrypted with TLS over TCP, and **don't need to be encrypted**
    - QUIC provides no more security, but makes it **expensive** for middleboxes to read handshake
  - Rest of QUIC connection protected by TLS 1.3
    - TLS **`ClientHello`**, **`ServerHello`**, **`Finished`** messages sent inside QUIC handshake; TLS 1.3 selects keys to encrypt following QUIC packets
- Authentication → if a middlebox changes the headers, the change can be detected

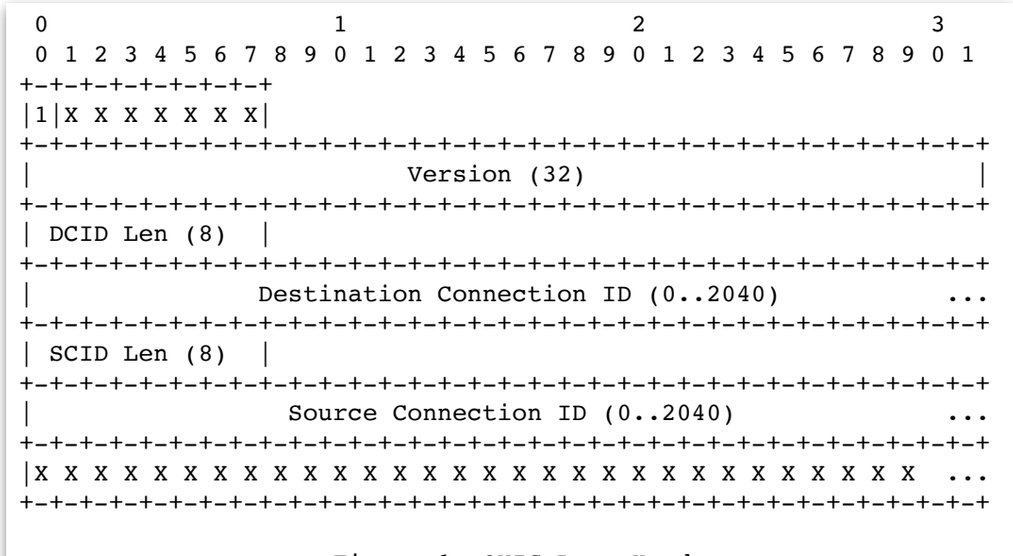


Figure 1: QUIC Long Header

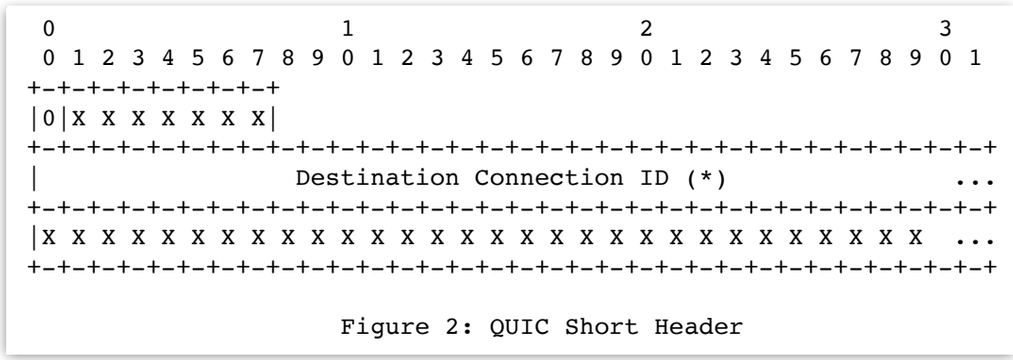


Figure 2: QUIC Short Header

https://datatracker.ietf.org/doc/draft-ietf-quic-invariants/

X = bit may take arbitrary value

CC BY ND

Avoiding Ossification via GREASE

- **Generate Random Extensions and Sustain Extensibility (GREASE)**
  - Every field encrypted – makes contents look random – or sometimes given random value
    - New connections use a Connection ID that is randomly chosen
    - Version negotiation randomly tested
      - Version numbers matching **`0x?a?a?a?a`** for testing version negotiation – will be rejected by servers
      - Clients will randomly try to negotiate such versions
    - Unused header fields have random values
  - Goal: poor quality middleboxes can't make an assumption about field values since every field changes apparently randomly
    - Hopefully avoid ossification, since nothing to ossify around
    - Smarter middleboxes will work with invariant fields and decrypt initial handshake

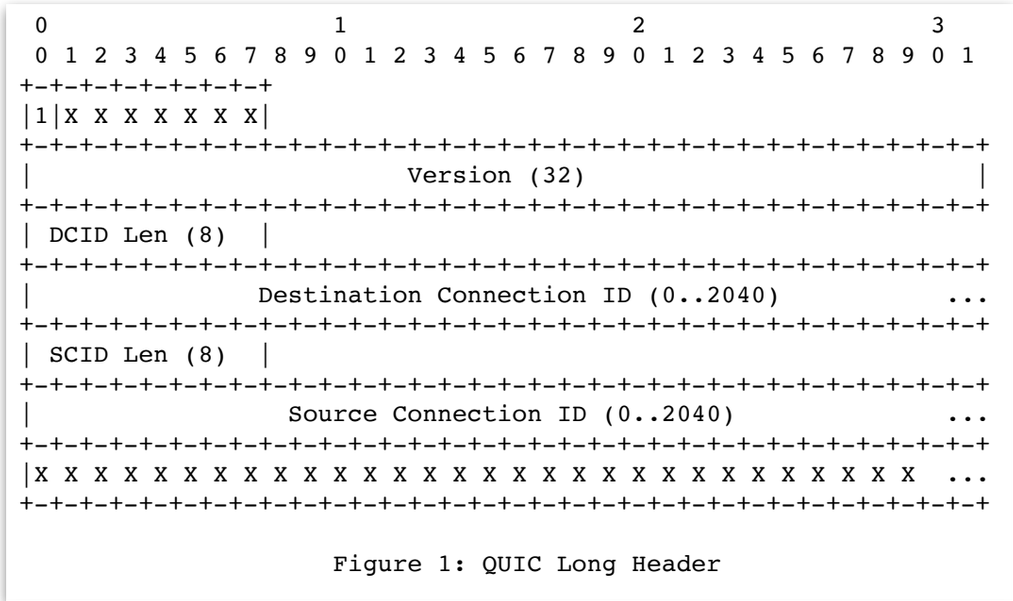


Figure 1: QUIC Long Header

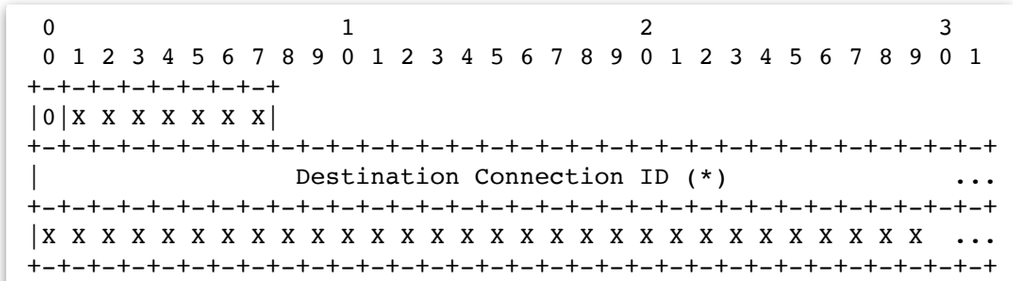


Figure 2: QUIC Short Header

https://datatracker.ietf.org/doc/draft-ietf-quic-invariants/

X = bit may take arbitrary value

CC BY ND

Long Headers and Short Headers

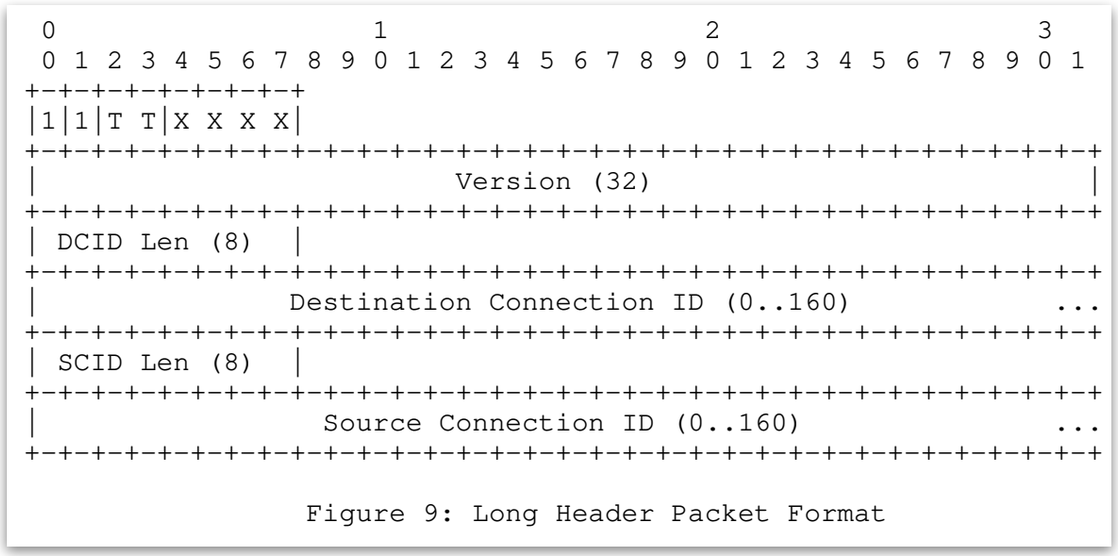


Figure 9: Long Header Packet Format

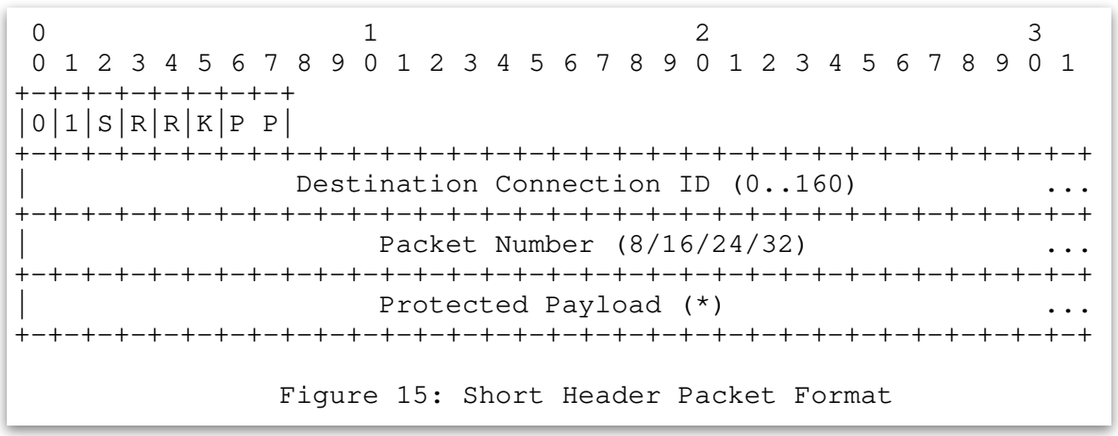


Figure 15: Short Header Packet Format

- **Long header packets**
  - Initial – initiates connection, carries TLS **`ClientHello`** and **`ServerHello`**
  - 0-RTT – initiates connection, carries TLS **`ClientHello`** when 0-RTT data is used
  - Handshake – concludes TLS handshake and finishes negotiating QUIC options
  - Version negotiation
- **Short header packets**
  - 1-RTT – Used for all packets sent after the TLS handshake is complete
- Multiple QUIC packets can be sent in a single UDP packet

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Frames

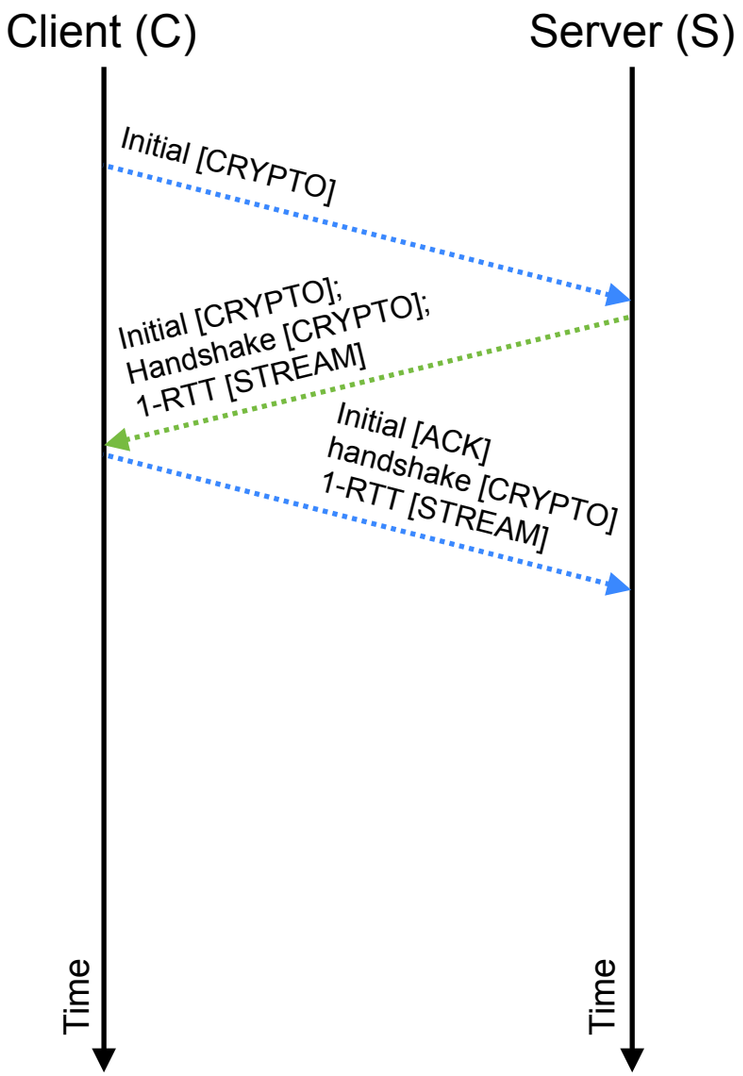
- The body of a QUIC packet contains an encrypted sequence of **frames**
  - CRYPTO frames are used to carry TLS messages such as the **`ClientHello`**, **`ServerHello`** etc.
  - STREAM and ACK frames send data and acknowledgements
  - Migration between two network paths is supported by PATH_CHALLENGE and PATH_RESPONSE frames
  - Other frames control progress of a QUIC connection
- Compared to TCP, QUIC headers are limited in scope
  - Functionality provided by frames instead
  - e.g., TCP header has sequence numbers and acknowledgements; QUIC sends this information in STREAM and ACK frames

Type Value	Frame Type Name	Definition	Packets
0x00	PADDING	Section 19.1	IH01
0x01	PING	Section 19.2	IH01
0x02 - 0x03	ACK	Section 19.3	IH_1
0x04	RESET_STREAM	Section 19.4	__01
0x05	STOP_SENDING	Section 19.5	__01
0x06	CRYPTO	Section 19.6	IH_1
0x07	NEW_TOKEN	Section 19.7	___1
0x08 - 0x0f	STREAM	Section 19.8	__01
0x10	MAX_DATA	Section 19.9	__01
0x11	MAX_STREAM_DATA	Section 19.10	__01
0x12 - 0x13	MAX_STREAMS	Section 19.11	__01
0x14	DATA_BLOCKED	Section 19.12	__01
0x15	STREAM_DATA_BLOCKED	Section 19.13	__01
0x16 - 0x17	STREAMS_BLOCKED	Section 19.14	__01
0x18	NEW_CONNECTION_ID	Section 19.15	__01
0x19	RETIRE_CONNECTION_ID	Section 19.16	__01
0x1a	PATH_CHALLENGE	Section 19.17	__01
0x1b	PATH_RESPONSE	Section 19.18	__01
0x1c - 0x1d	CONNECTION_CLOSE	Section 19.19	IH_1*
0x1e	HANDSHAKE_DONE	Section 19.20	___1

Table 3: Frame Types

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

QUIC Connection Establishment (1/5)



- Normal connection establishment and TLS security negotiation combined into one handshake
  - C → S: One UDP datagram containing one QUIC packet
    - QUIC **Initial** packet containing CRYPTO frame that contains TLS **`ClientHello`** message
  - S → C: One UDP datagram containing three QUIC packets
    - QUIC **Initial** packet containing CRYPTO frame that contains TLS **`ServerHello`** message
    - QUIC **Handshake** packet containing other TLS parameters
    - QUIC **1-RTT** (short header) packet containing a STREAM frame with any initial data sent from server to client
  - C → S: One UDP datagram containing three QUIC packets
    - QUIC **Initial** packet acknowledging data from server
    - QUIC **Handshake** packet containing CRYPTO frame to finish TLS handshake
    - QUIC **1-RTT** (short header) packet containing a STREAM frame with the initial data sent from client to server
- All UDP datagrams sent during the QUIC handshake start with a QUIC long header packet

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

QUIC Connection Establishment (2/5)

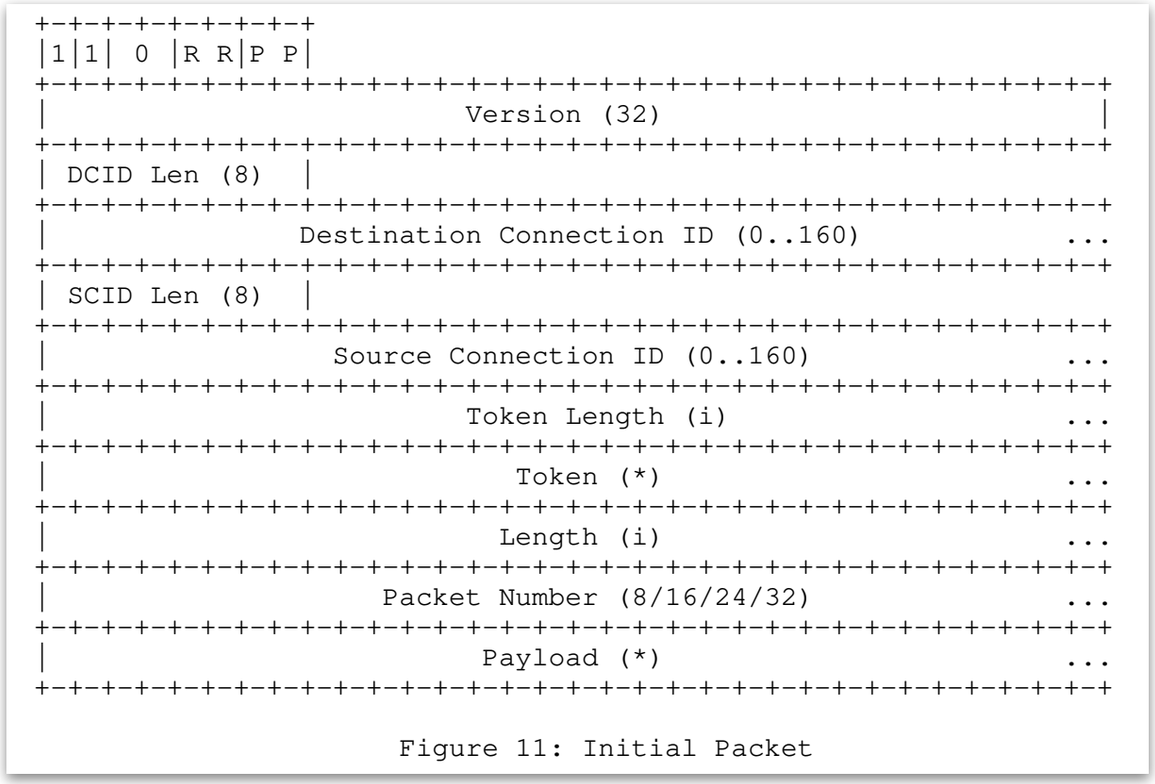


Figure 11: Initial Packet

- QUIC **Initial** packets play two roles:
  - Synchronise client and server state – like TCP's **`SYN`** - **`SYN+ACK`** - **`ACK`** handshake
  - Deliver CRYPTO frames within which the TLS 1.3 handshake operates
    - TLS **`ClientHello`** and **`ServerHello`**
  - Combined into one packet in QUIC, to save an RTT compared with TCP
    - Security built-in to QUIC from the start, but added to TCP later
- QUIC **Initial** packets can also carry an optional **`Token`**
  - Server sends an unpredictable token to client, securely encrypted using TLS
  - Client sends it back to server, encrypted
  - Attacker won't be able to guess token, so prevents spoofing connection

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

QUIC Connection Establishment (3/5)

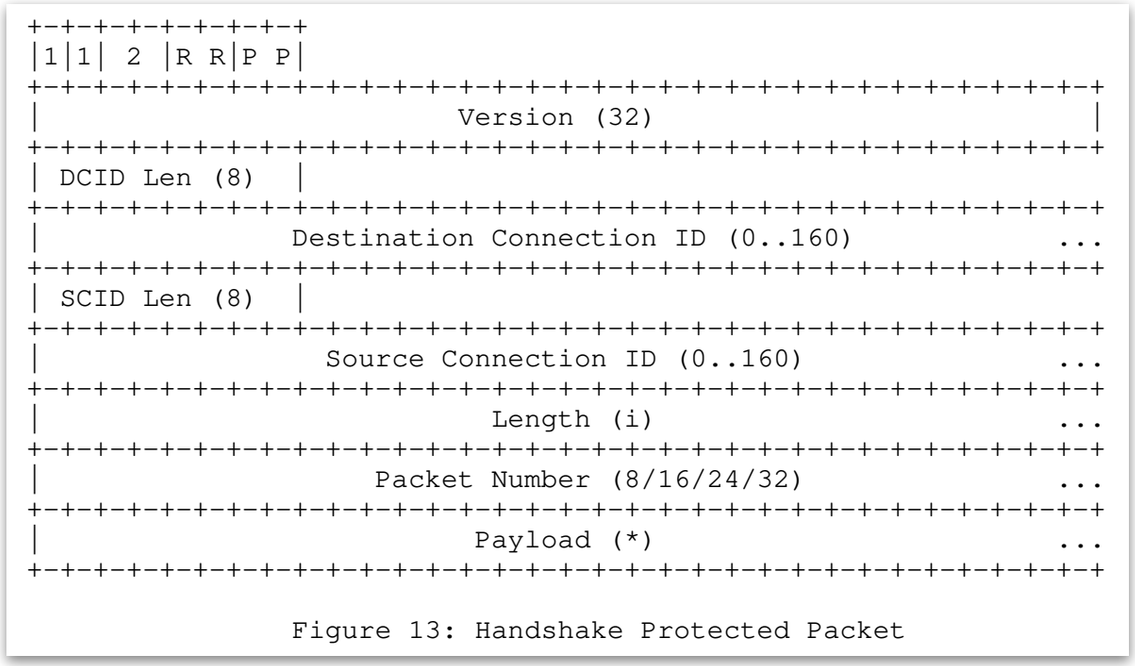
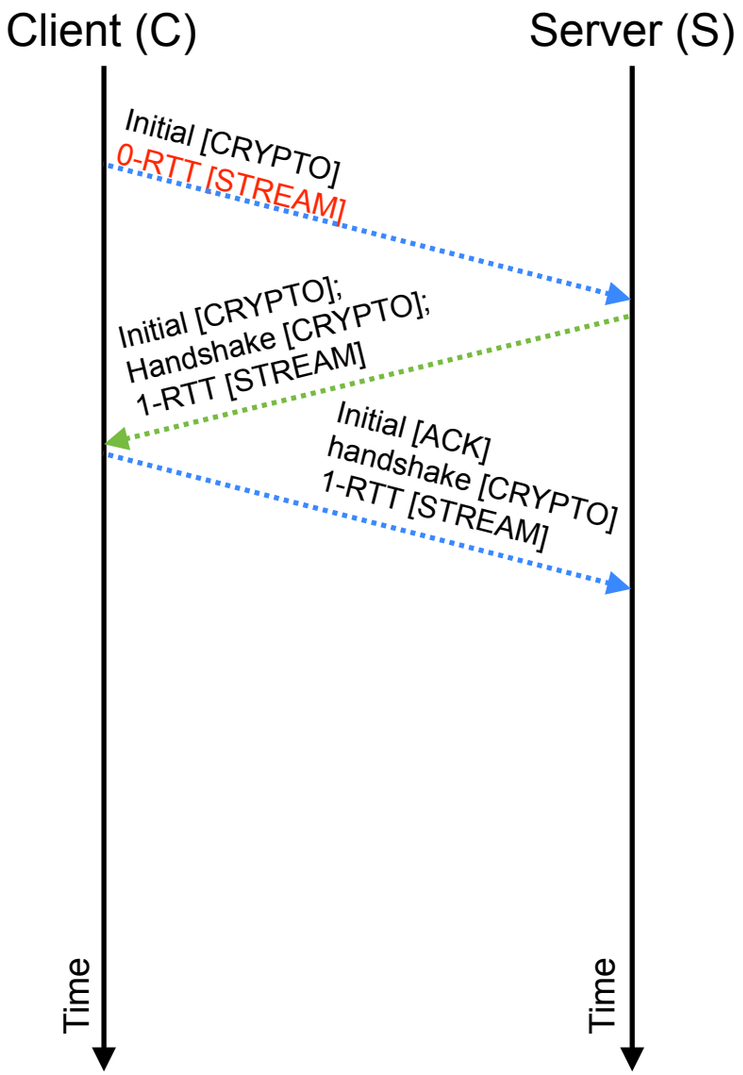


Figure 13: Handshake Protected Packet

- QUIC **Handshake** packets complete the TLS 1.3 exchange
  - e.g., the TLS **`Finished`** message
- Also used for TLS key renegotiation and parameter changes
  - TLS can renegotiate new keys part-way through a connection – this is done in **`Handshake`** packets

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

QUIC Connection Establishment (4/5)



- If client and server have previously communicated can establish **0-RTT** connection
  - QUIC **Initial** packet contains CRYPTO frame with a TLS **`ClientHello`** that has a previously **`PresharedKey`**
  - QUIC **0-RTT** packet included in the same UDP datagram contains the idempotent 0-RTT data:

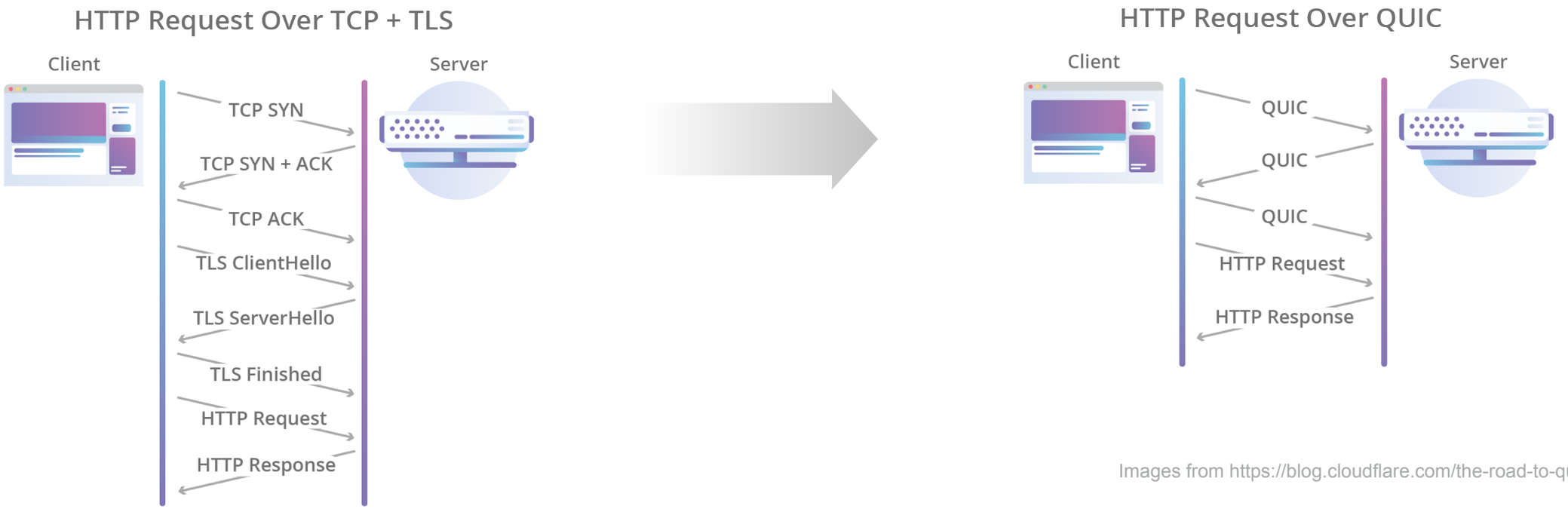
```
+-+-+-+-+-+-+-+-+
|1|1| 1 |R R|P P|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Version (32)                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| DCID Len (8)  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               Destination Connection ID (0..160)            ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| SCID Len (8)  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Source Connection ID (0..160)               ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Length (i)                        ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Packet Number (8/16/24/32)               ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Payload (*)                       ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                    Figure 12: 0-RTT Packet
```

- Behaves the same as 0-RTT data in TLS-over-TCP, but sent with initial connection establishment, rather than as first data within TCP connection

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

QUIC Connection Establishment (5/5)



Images from https://blog.cloudflare.com/the-road-to-quic/

- QUIC combines connection establishment and encryption key negotiation into a single handshake; TLS-over-TCP runs them sequentially
- Reduces secure connection establishment latency

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Data Transfer, Streams, and Reliability (1/3)

- QUIC data transfer mostly occurs in short header packets
  - STREAM frames
  - ACK frames

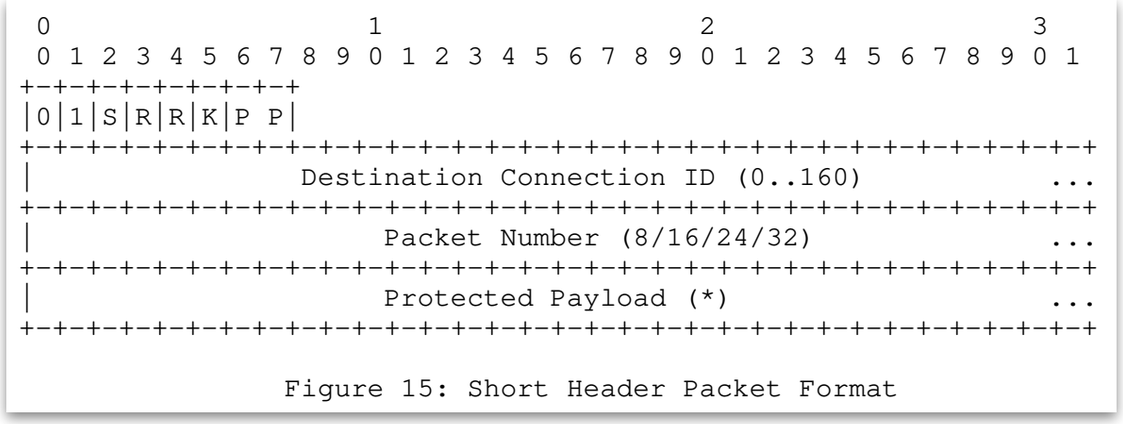


Figure 15: Short Header Packet Format

- Streams and framing, avoiding head-of-line blocking → Lecture 5
- Congestion Control as in TCP → Lecture 6

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Data Transfer, Streams, and Reliability (2/3)

- Data is sent within STREAM frames
  - Sent within a long header or short header packet
  - The QUIC **packets** contain a sequence number – frames are not acknowledged

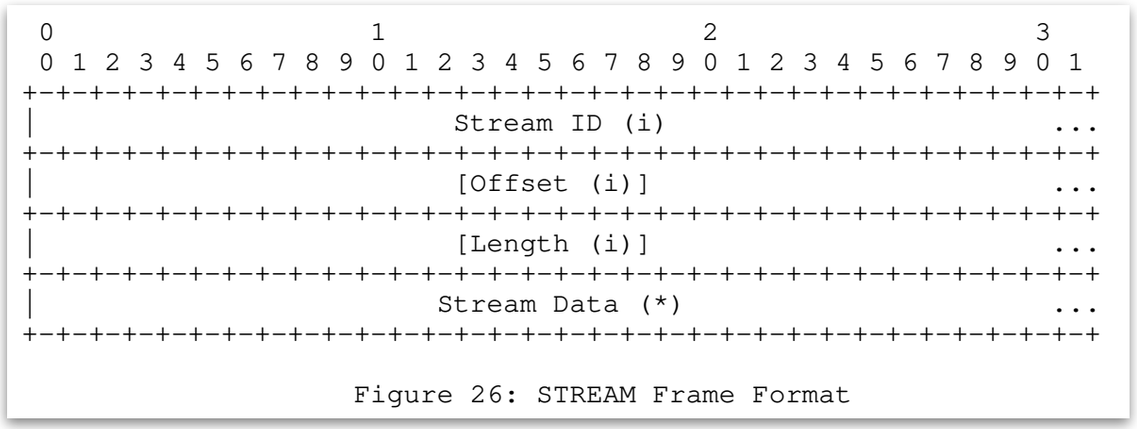


Figure 26: STREAM Frame Format

- TCP provides a single reliable byte stream
- QUIC provides multiple reliable byte streams within a single connection
  - Each stream is delivered reliably and in-order
  - Order is not preserved between streams within a QUIC connection

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

Data Transfer, Streams, and Reliability (3/3)

- QUIC sends ACK frames to indicate receipt of data
  - Sent within a long header or short header packet – unlike TCP, not part of headers
  - Indicate sequence numbers of QUIC **packets** that were received – not frames

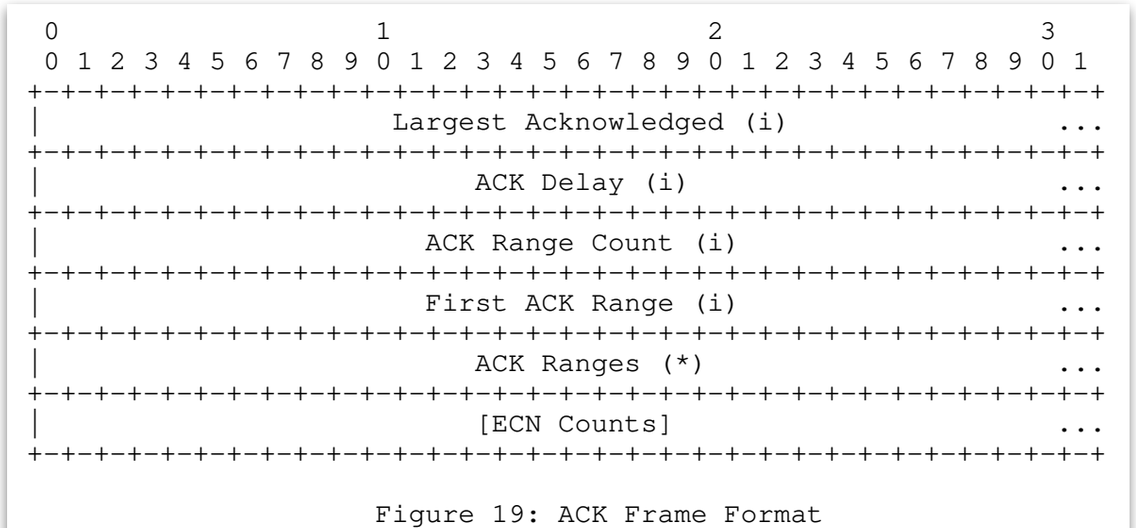


Figure 19: ACK Frame Format

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow

QUIC Benefits and Costs

- **Why is QUIC desirable?**
  - Reduces secure connection establishment latency
  - Reduces risk of ossification; easy to deploy
  - Supports multiple streams within a single connection



- **Why is QUIC problematic?**
  - Unfinished – protocol is still changing; likely corner cases that are not considered yet
  - Libraries and support new, poorly documented, and frequently buggy
  - CPU usage is high compared to TCL-over-TCP
    - TCP stack currently *much* better optimised
    - TCP and TLS often has hardware offline; QUIC doesn't yet
  - These issues will be resolved – **but it will take some years before QUIC is as stable and performant as TLS-over-TCP**
    - TCP has lasted 40 years – QUIC is a similarly long-term project, that is not yet a version 1.0

CC BY ND

Improving Secure Connection Establishment

- Limitations of TLS 1.3
- QUIC transport protocol
- TCP has dominated for 40 years – is QUIC the future?

Colin Perkins | https://csperkins.org/ | Copyright © 2020 University of Glasgow