Computer Networks Architectures and Protocols

Arquitectura e Protocolos de Redes de Computadores

APRC

Lecture Ten and 1/2

João Leitão
Paulo Lopes

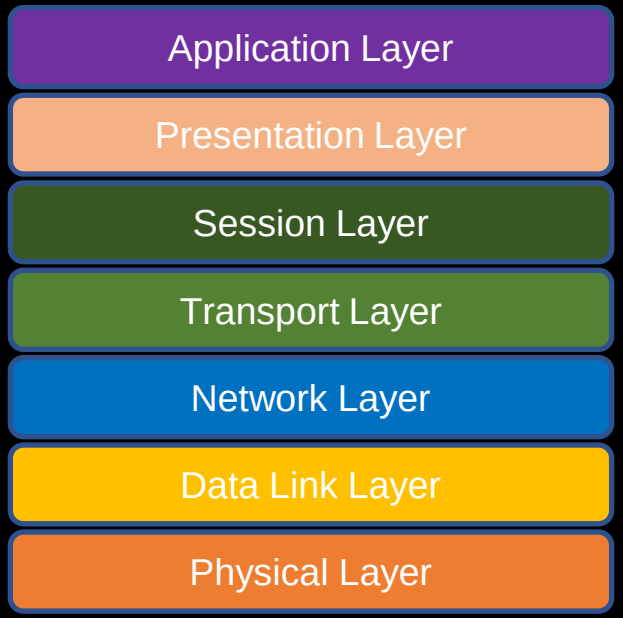
MEI/MIEI

Lecture 10 1/2

- Transport Layer (and Transport Protocols)
- End-to-End Argument

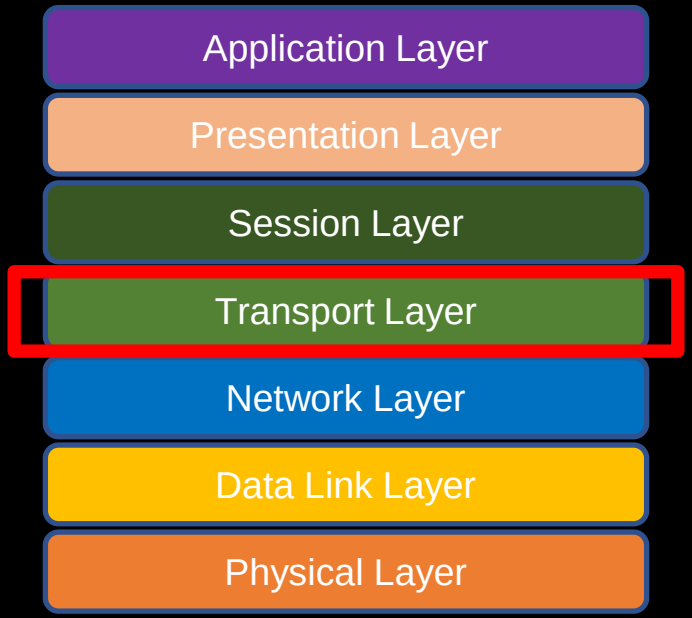
Today

OSI Model



Today

OSI Model



We are going to discuss the Transport Layer.

Transport Layer

The transport layer's tasks include error correction as well as segmenting and desegmenting data before and after it's transported across the network. This layer is also responsible for flow control and making sure that segmented data is delivered over the network in the correct sequence.

-- Technopedia

Transport Layer Responsibilities

The responsibilities of the Transport Layer might include:

- Data Integrity
- Multiplexing
- Same Order Delivery
- Connection-Oriented Communication
- Byte Orientation
- Flow Control
- Traffic Control

Transport Layer Responsibilities

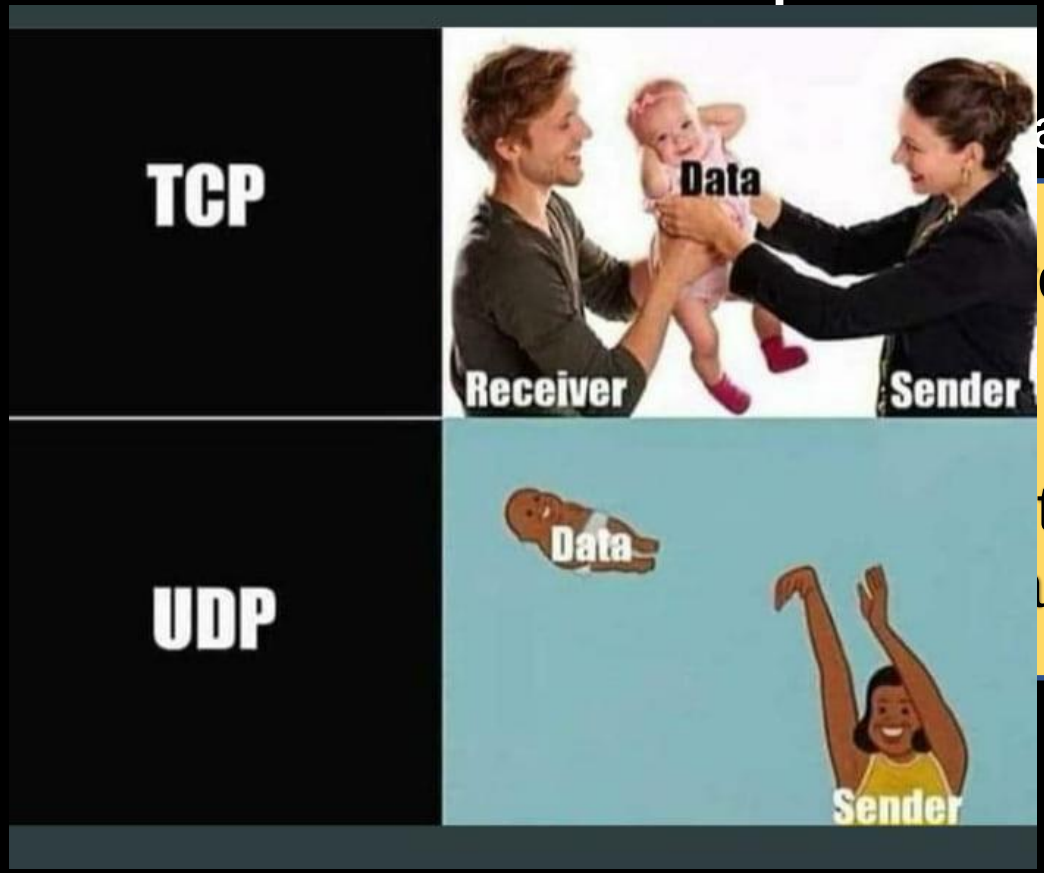
The responsibilities of the Transport Layer might include:

- Data
- Mult
- Sam
- Con
- Byte
- Flow Control
- Traffic Control

The popular transport protocols are your old pals:

TCP: Transport Control Protocol
UDP: User Datagram Protocol

Transport Layer Responsibilities

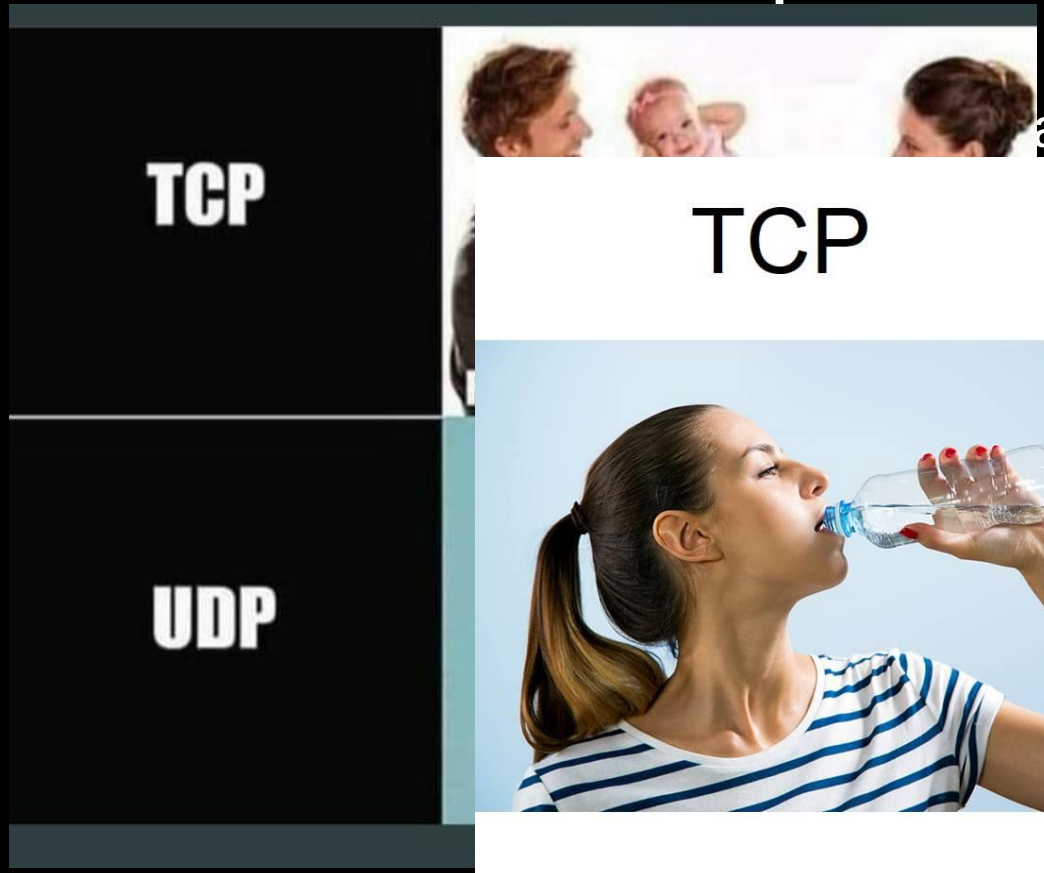


ayer might include:

otocols are your

trol Protocol
m Protocol

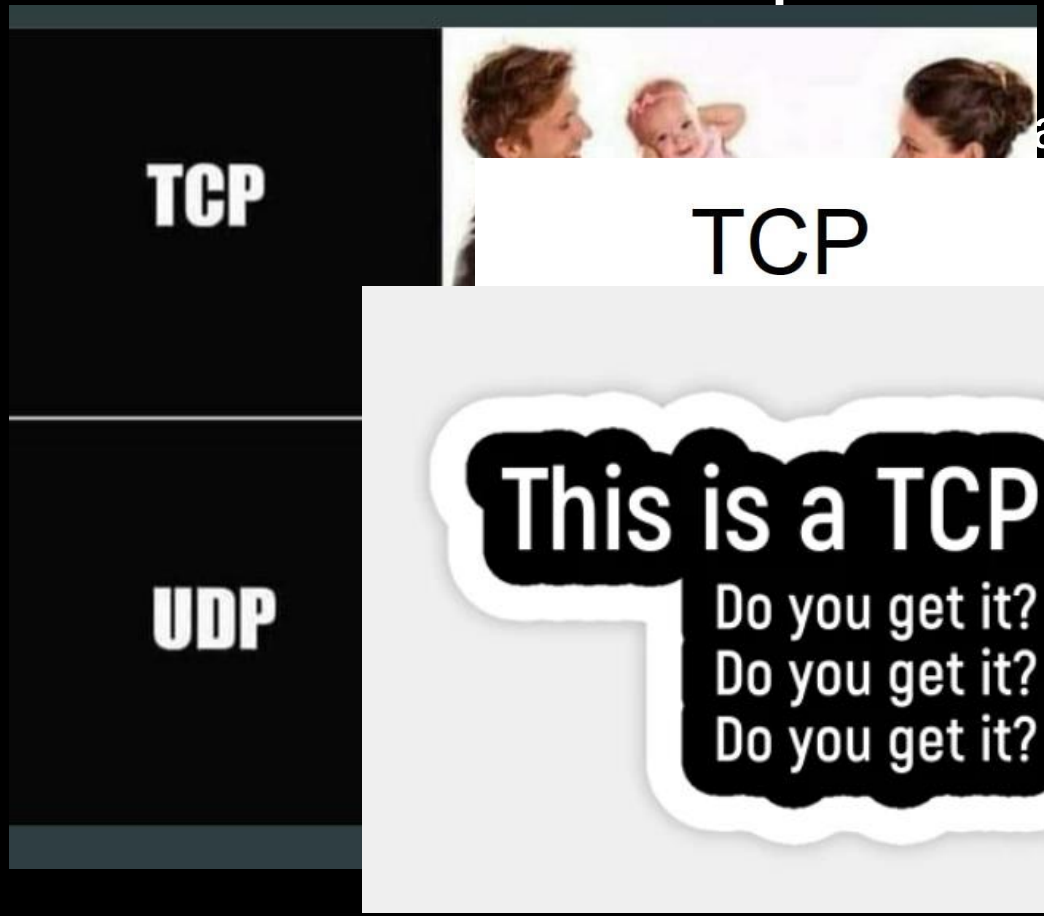
Transport Layer Responsibilities



ayer might include:

Transport Layer Responsibilities

ayer might include:



Transport Layer Responsibilities

The responsibilities of the Transport Layer might include:

- Data Integrity
- Multiplexing
- Same Order Delivery
- Connection-Oriented Communication
- Byte Orientation
- Flow Control
- Traffic Control

Data Integrity

Data Integrity refers to the ability to ensure that the data that is received is the same as the data that was originally sent.

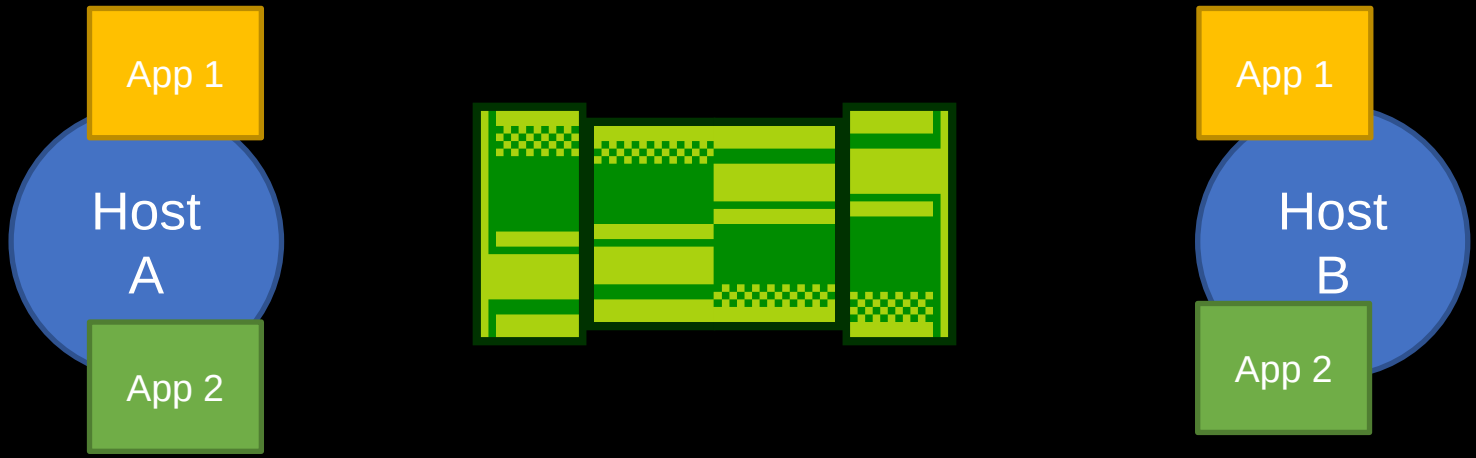
This is achieved by having a checksum within the transport protocol layer, which can be verified by the receiver. If the verification fails, the packed can be dropped (which transforms a commission (e.g., corruption) failure in an omission failure).

Multiplexing

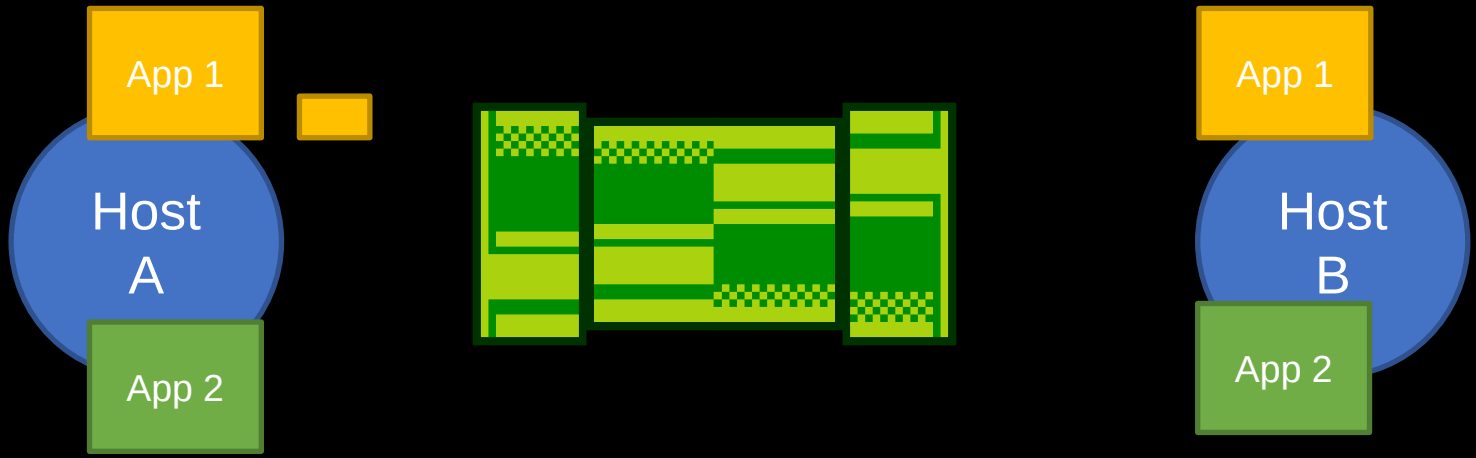
Maybe the most underrated task of transport protocols is the ability to multiplex communication between two hosts.

Multiplexing here means the capacity for different applications across two hosts to communicate simultaneously while sharing the communication medium.

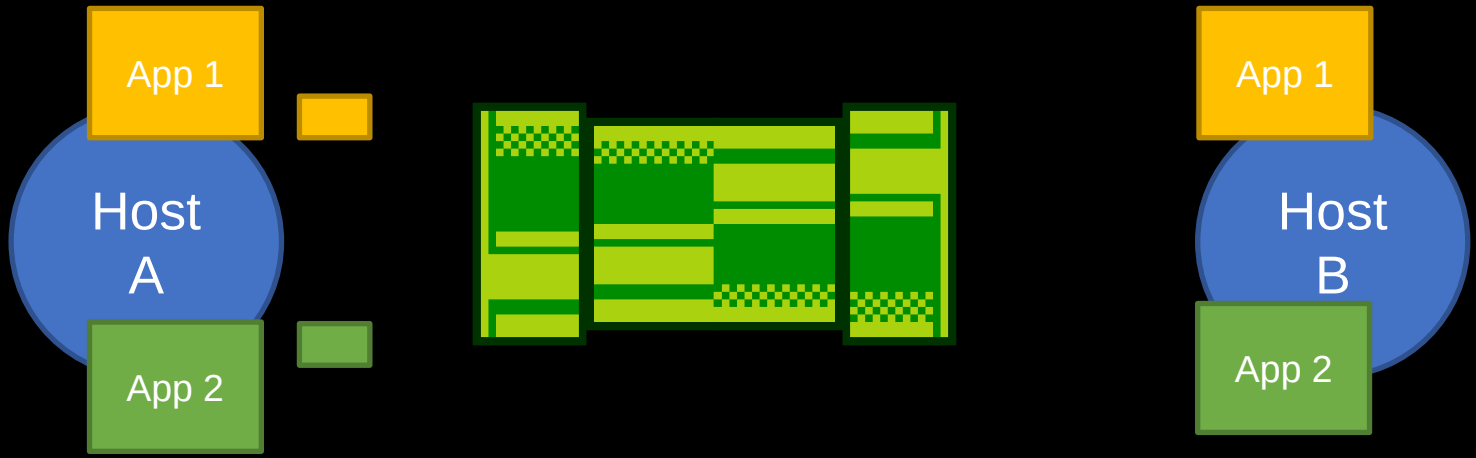
Multiplexing



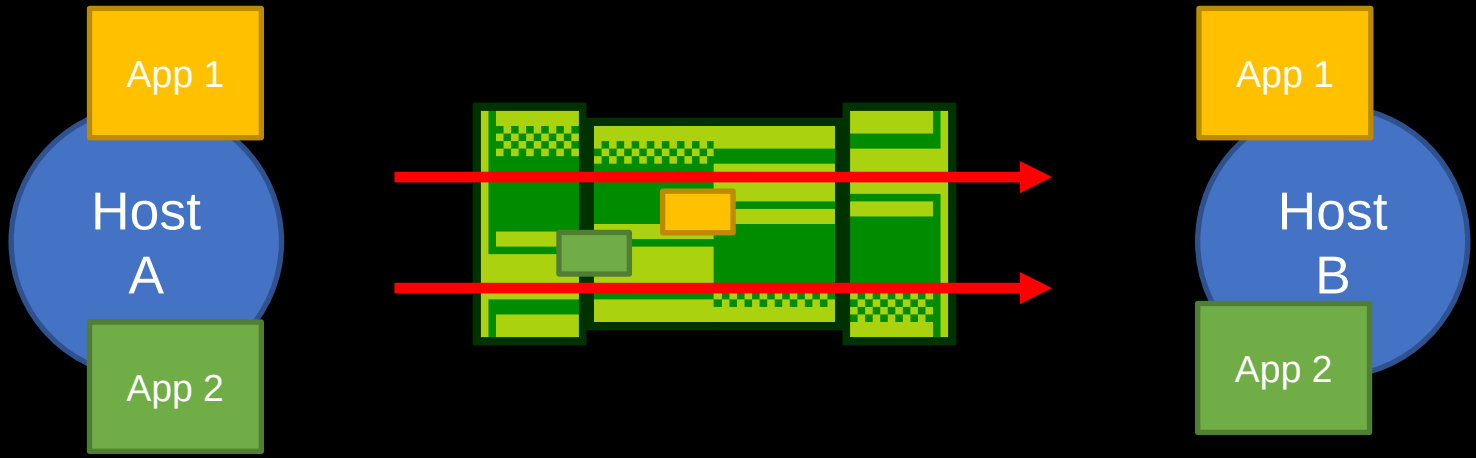
Multiplexing



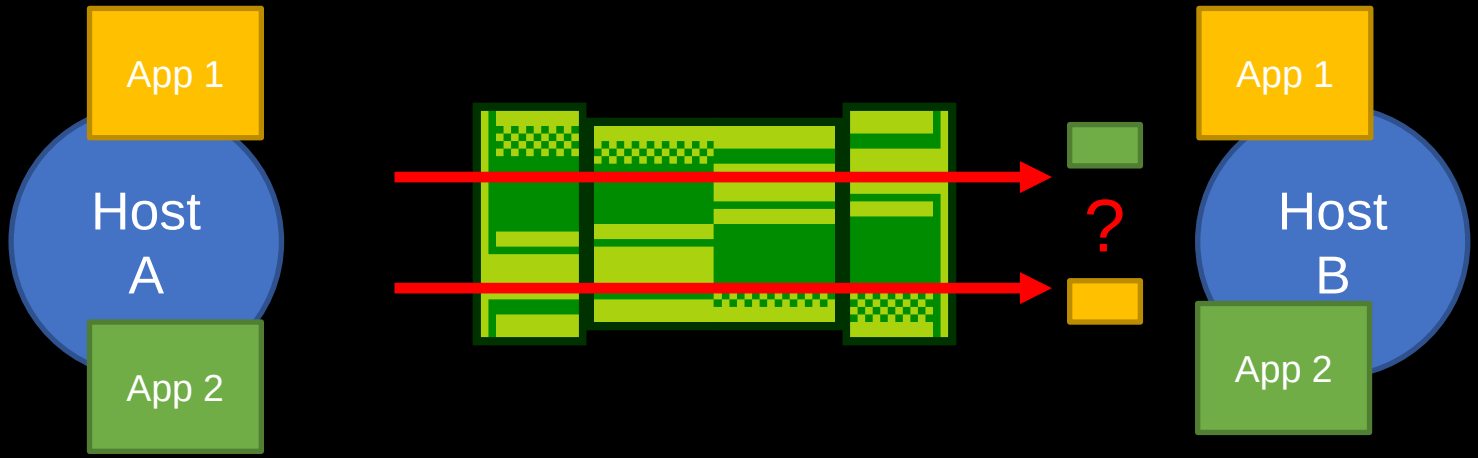
Multiplexing



Multiplexing



Multiplexing



Multiplexing

Multiplexing is achieved by the introduction of network ports (in the header of the transport protocol).

In some sense, like how a MAC address identifies a network card, an IP address identifies a host, a network port will identify an application within a given host (or more precisely, a data flux for a particular application).

Same Order Delivery

Ensuring that packets are delivered in the same strict sequence as they were originally sent.

This is achieved by using some scheme for numbering packets and requesting retransmissions of packets that might have been lost in the network (or discarded due to corruption).

Connection-Oriented Communication

Ensuring that a stable communication channel has been correctly established between two end-points (applications) before starting to transmit applications data, and that transmitted data is effectively received.

This is achieved by resorting to some form of handshake performed between the two communicating end-points, and using explicit acknowledgments for received data. Naturally, this will introduce additional delays in data exchange and additional load on the network.

Byte Orientation

Enabling applications to communicate using streams of bytes instead of finite units of data (such as packets or messages).

Flow Control

Different hosts that communicate might have different capabilities for communicating (e.g., bandwidth, RAM, CPU) which might lead to scenarios where the sender sends data faster that the capability of the receiver for buffering and processing data, which can lead to communication failures (or vice-versa, leading to a suboptimal use of the network).

Flow control allows end-points to adjust their rates of transmission automatically in a way that is acceptable for both endpoints of a data flow.

Traffic Control

The network capacity might be exhausted (due to limitations on bandwidth or processing power of routing equipment) by the existence of multiple (concurrent) flows of data leading to the congestion on the network.

Some transport protocols can identify symptoms of network congestion (usually due to drop of packets in transit) and adjust their transmission rates to compensate such scenarios.

Transport Layer Responsibilities

The responsibilities of the Transport Layer might include:

- Data Integrity
- Multiplexing
- Same Order Delivery
- Connection-Oriented Communication
- Byte Orientation
- Flow Control
- Traffic Control

Transport Layer Responsibilities

The responsibilities of the Transport Layer might include:

• Data Integrity	TCP/UDP
• Multiplexing	TCP/UDP
• Same Order Delivery	TCP
• Connection-Oriented Communication	TCP
• Byte Orientation	TCP
• Flow Control	TCP
• Traffic Control	TCP

Transport Layer Responsibilities

The responsibilities of the Transport Layer might include:

• Data Integrity	TCP/UDP
• Multiplexing	TCP/UDP
• Same Order Delivery	TCP
• Connection-Oriented Communication	TCP
• Byte Orientation	TCP
• Flow Control	TCP
• Traffic Control	TCP

Checksums

Ports

Transport Layer Responsibilities

The responsibilities of the Transport Layer might include:

• Data Integrity	TCP/UDP
• Multiplexing	TCP/UDP
• Same Order Delivery	TCP
• Connection-Oriented Communication	TCP
• Byte Orientation	TCP
• Flow Control	TCP
• Traffic Control	TCP

Checksums

Ports

Sequence numbers

3-way handshake and acks

Transport Layer Responsibilities

The responsibilities of the Transport Layer might include:

• Data Integrity	TCP/UDP	Checksums
• Multiplexing	TCP/UDP	Ports
• Same Order Delivery	TCP	Sequence numbers
• Connection-Oriented Communication	TCP	3-way handshake and acks
• Byte Orientation	TCP	Transmission/reception windows and fragmentation
• Flow Control	TCP	+ Slow start and exponential backoff
• Traffic Control	TCP	A bit of all the of the above

Popular Transport Protocols

TCP: Connection oriented, network friendly, and reliable.

UDP: Datagram oriented, allows to send large burst of data, best effort (no guarantees of delivery).

Popular Transport Protocols

IPv4 pseudo header format

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source IPv4 Address																															
4	32	Destination IPv4 Address																															
8	64	Zeroes								Protocol								UDP Length															
12	96	Source Port																Destination Port															
16	128	UDP Length																Checksum															
20	160+	Data																															

Popular Transport Protocols

IPv6 pseudo header format

Offsets Octet	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source IPv6 Address																															
4	32																																
8	64																																
12	96																																
16	128	Destination IPv6 Address																															
20	160																																
24	192																																
28	224																																
32	256	UDP Length																															
36	288	Zeroes																								Next Header = Protocol[11]							
40	320	Source Port																Destination Port															
44	352	Length																Checksum															
48	384+	Data																															

Popular Transport Protocols

TCP pseudo-header for checksum computation (IPv4)

Bit offset	0–3	4–7	8–15	16–31
0	Source address			
32	Destination address			
64	Zeros		Protocol	TCP length
96	Source port			Destination port
128	Sequence number			
160	Acknowledgement number			
192	Data offset	Reserved	Flags	Window
224	Checksum			Urgent pointer
256	Options (optional)			
256/288+	Data			

Popular Transport Protocols

TCP pseudo-header for checksum computation (IPv4)

Bit offset	0–3	4–7	8–15	16–31
0	Source address			
32	Destination address			
64	Zeros		Protocol	TCP length
96	Source port			Destination port
128	Sequence number			
160	Acknowledgement number			
192	Data offset	Reserved	Flags	Window
224	Checksum			Urgent pointer
256	Options (optional)			
256/288+	Data			

Popular Transport Protocols

TCP pseudo-header for checksum computation (IPv6)

Bit offset	0–7	8–15	16–23	24–31
0	Source address			
32				
64				
96				
128	Destination address			
160				
192				
224				
256	TCP length			
288	Zeros			Next header = Protocol
320	Source port		Destination port	
352	Sequence number			
384	Acknowledgement number			
416	Data offset	Reserved	Flags	Window
448	Checksum			Urgent pointer
480	Options (optional)			
480/512+	Data			

Popular Transport Protocols

Independently of the concrete transport protocol being employed, it is obvious that many responsibilities are delegated either to the transport protocol (TCP case) or to layers above, potentially the application (UDP case).

Could we not move some of these functionalities to the lower layers?

L2 and L3 already do too much…

Layer 2 (Data Link) and in particular Layer 3 (Network) are already addressing a set of very complex problems, that requires a significant amount of computational resources to handle.

Delegating more functionalities to lower layers would make the complexity of routers (and switches) unbearable (and equipment too expensive).

The End-to-End Argument

J. Saltzer, D. Reed ad D. Clark, “End-to-End Arguments in System Design,” ACM Transactions on Computer Systems (4):277-288, November 1984.

END-TO-END ARGUMENTS IN SYSTEM DESIGN

J.H. Saltzer, D.P. Reed and D.D. Clark*

M.I.T. Laboratory for Computer Science

This paper presents a design principle that helps guide placement of functions among the modules of a distributed computer system. The principle, called the end-to-end argument, 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. Examples discussed in the paper include bit error recovery, security using encryption, duplicate message suppression, recovery from system crashes, and delivery acknowledgement. Low level mechanisms to support these functions are justified only as performance enhancements.

The End-to-End Argument

J. Saltzer, D. Reed ad D. Clark, “End-to-End Arguments in System Design,” ACM Transactions on Computer Systems (4):277-288, November 1984.

END-TO-END ARGUMENTS IN SYSTEM DESIGN

J.H. Saltzer, D.P. Reed and D.D. Clark*

M.I.T. Laboratory for Computer Science

This paper presents a design principle that helps guide placement of functions among the modules of a distributed computer system. The principle, called the end-to-end argument, 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. Examples discussed in the paper include bit error recovery, security using encryption, duplicate message suppression, recovery from system crashes, and delivery acknowledgement. Low level mechanisms to support these functions are justified only as performance enhancements.

The End-to-End Argument

A more direct explanation:

- Ideally, most complexity should be implemented in the edge, i.e., in the upper layers of the system.
- However, when a common and popular feature can be implemented with lower cost and higher efficiency in lower layers, it is preferable to implement it directly at these layers.
- Additionally, a well-designed system should be extensible and should easily support the introduction of new features in upper layers, without requiring modification of the lower ones.

The End-to-End Argument

The principles established by the end-to-end argument are quite notorious on the Networking layer.

- The way that the network layer operates, is by simply store and forward individual packets that are treated independently (no need to keep state on data flows).
- Delivery at the network layer is best-effort (packets can be lost, corrupted, delivered out of order).
- The network never has to say its sorry for any anomaly that happens.
- Easier to ensure network correctness during failures.

Back to Transport

So TCP and UDP are the only Transport protocols that exist right?

Back to Transport

So TCP and UDP are the only Transport protocols that exist right?

TCP: Originally published on RFC 675 (Dec 1974)

UDP: Originally published on RFC 768 (Aug 1980)

Back to Transport

So TCP and UDP are the only Transport protocols that exist right?

TCP: Origir Dec 1974)

UDP: Origi Aug 1980)

QUIC Transport Protocol

- General-purpose Transport Layer Network Protocol.
- Its design is highly motivated by the need to speed up HTTP traffic that happens on top of TCP (and most of the times TLS at the Session Layer – L5).
- It can be seen as the combination of UDP speed with TCP's reliability.
- Provides new features that answer current time needs.

QUIC Transport Protocol

- Originally designed by Jim Roskind at Google, it started to operate in Google in 2012, and it was publicly announced in 2013.

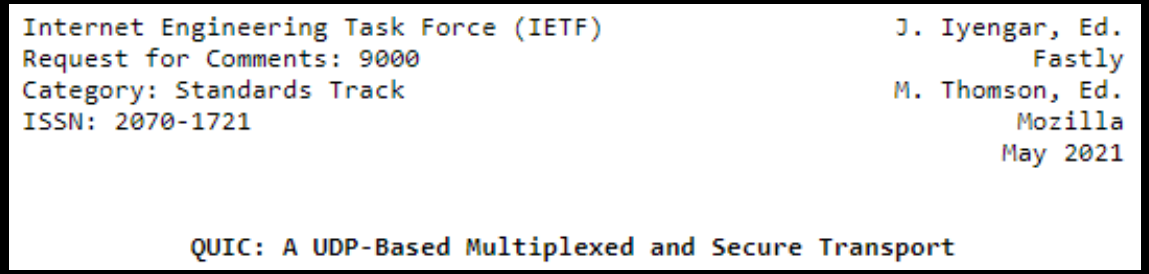
- Originally its name was an acronym for “Quick UDP Internet Connections”.

QUIC Transport Protocol

- In June 2014, an Internet Draft of the QUIC specification was submitted to the IETF for standardization.
- A QUIC workgroup within IETF was established on 2016.
- In October 2018, the IETF's HTTP and QUIC Working groups decided to call the HTTP over QUICK mapping HTTP/3.

QUIC Transport Protocol

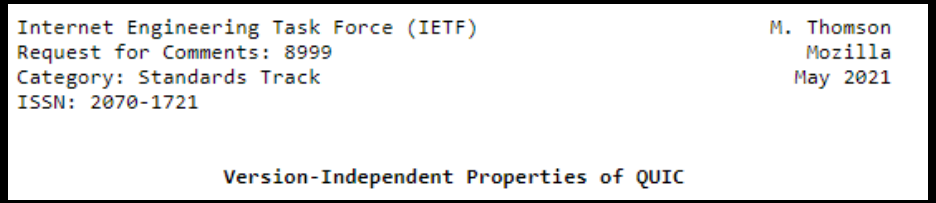
- In May 2021, the IETF standardized QUIC in RFC 9000, supported by RFCs 8999, 9001, and 9002.



Internet Engineering Task Force (IETF)
Request for Comments: 9000
Category: Standards Track
ISSN: 2070-1721

J. Iyengar, Ed.
Fastly
M. Thomson, Ed.
Mozilla
May 2021

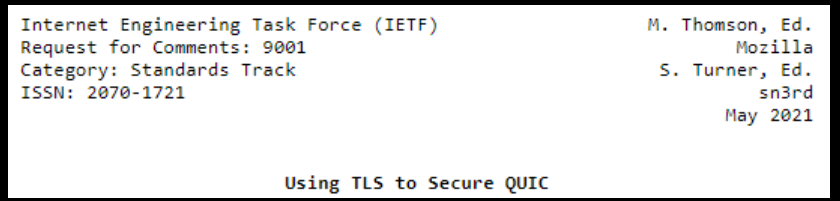
QUIC: A UDP-Based Multiplexed and Secure Transport



Internet Engineering Task Force (IETF)
Request for Comments: 8999
Category: Standards Track
ISSN: 2070-1721

M. Thomson
Mozilla
May 2021

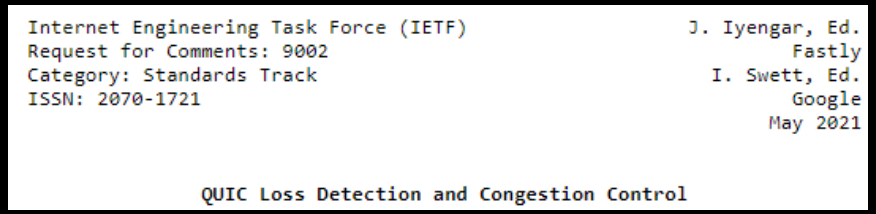
Version-Independent Properties of QUIC



Internet Engineering Task Force (IETF)
Request for Comments: 9001
Category: Standards Track
ISSN: 2070-1721

M. Thomson, Ed.
Mozilla
S. Turner, Ed.
sn3rd
May 2021

Using TLS to Secure QUIC



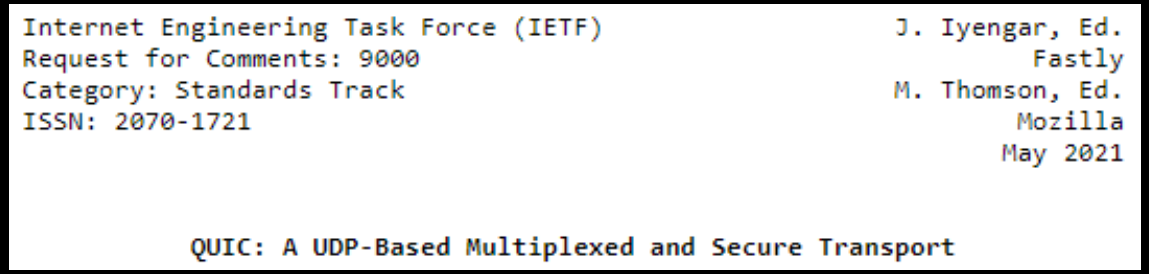
Internet Engineering Task Force (IETF)
Request for Comments: 9002
Category: Standards Track
ISSN: 2070-1721

J. Iyengar, Ed.
Fastly
I. Swett, Ed.
Google
May 2021

QUIC Loss Detection and Congestion Control

QUIC Transport Protocol

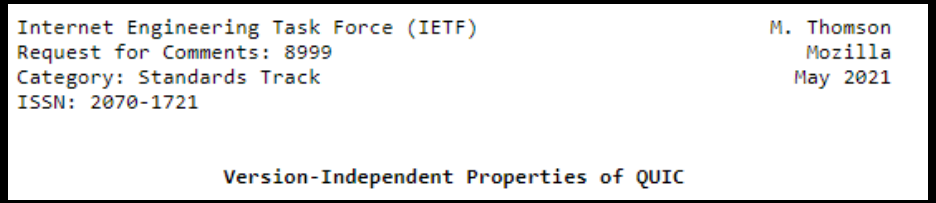
- In May 2021, the IETF standardized QUIC in RFC 9000, supported by RFCs 8999, 9001, and 9002.



Internet Engineering Task Force (IETF)
Request for Comments: 9000
Category: Standards Track
ISSN: 2070-1721

J. Iyengar, Ed.
Fastly
M. Thomson, Ed.
Mozilla
May 2021

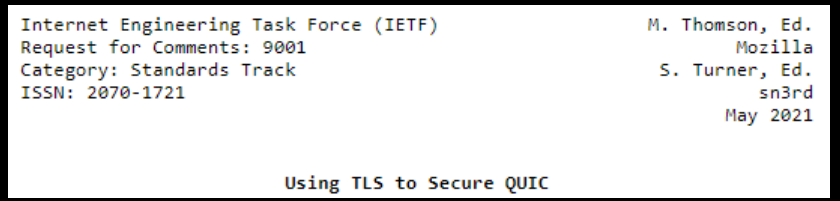
QUIC: A UDP-Based Multiplexed and Secure Transport



Internet Engineering Task Force (IETF)
Request for Comments: 8999
Category: Standards Track
ISSN: 2070-1721

M. Thomson
Mozilla
May 2021

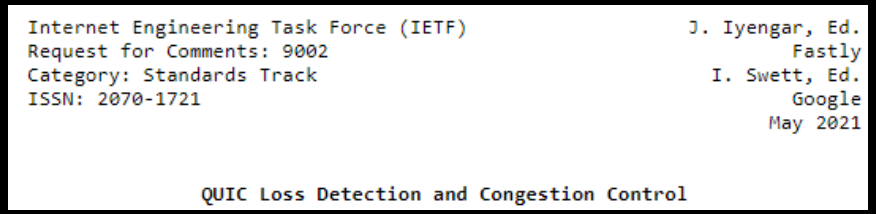
Version-Independent Properties of QUIC



Internet Engineering Task Force (IETF)
Request for Comments: 9001
Category: Standards Track
ISSN: 2070-1721

M. Thomson, Ed.
Mozilla
S. Turner, Ed.
sn3rd
May 2021

Using TLS to Secure QUIC



Internet Engineering Task Force (IETF)
Request for Comments: 9002
Category: Standards Track
ISSN: 2070-1721

J. Iyengar, Ed.
Fastly
I. Swett, Ed.
Google
May 2021

QUIC Loss Detection and Congestion Control

QUIC Transport Protocol

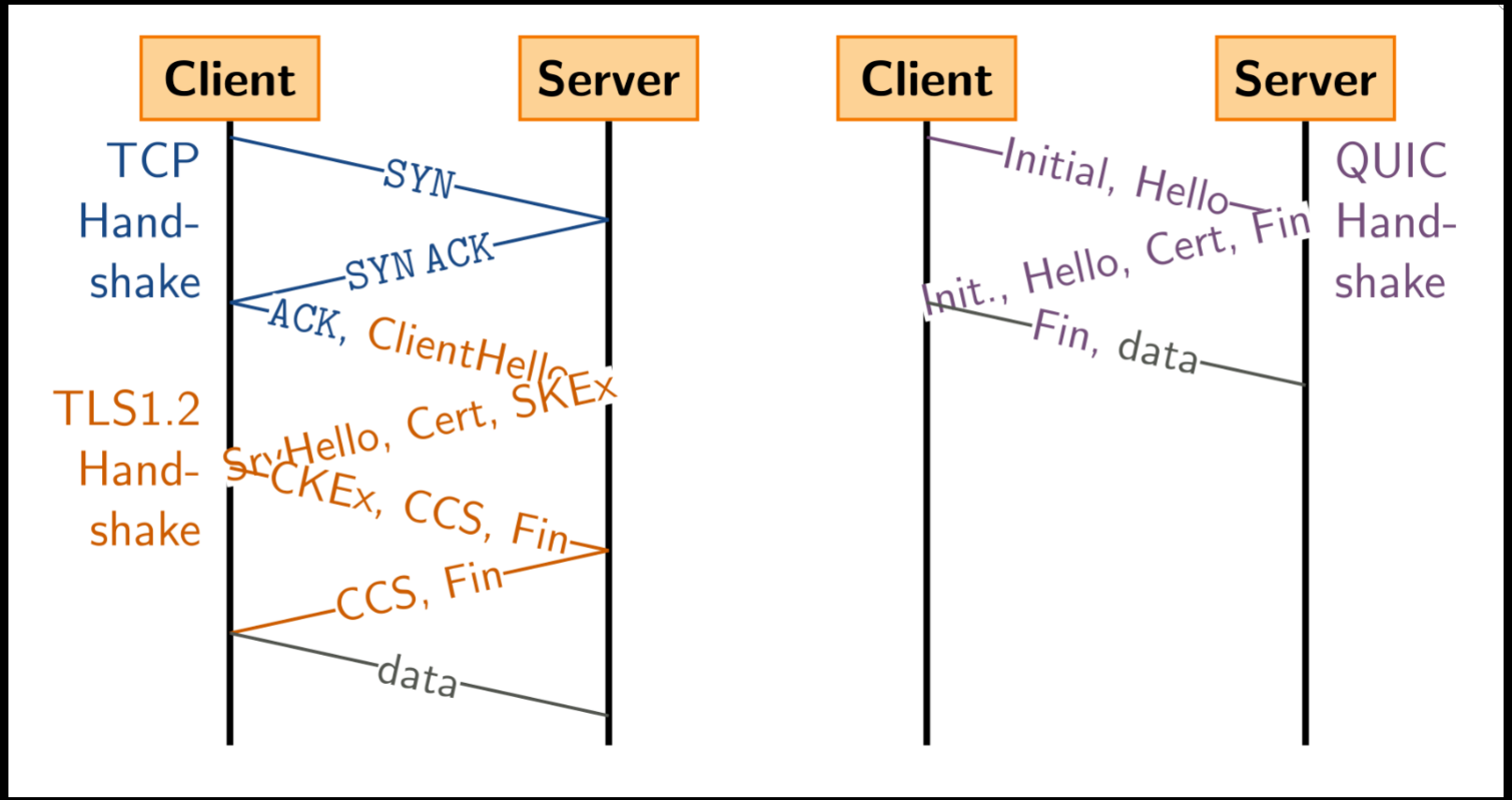
- Currently about 15 implementations of QUIC exist:
  - 14 in User space
  - 1 in Kernel space

- Why is this good (and not a huge source of overhead).
  - QUIC is designed thinking on HTTP/2, where most of the implementation and libraries is already user space.
  - This circumvents the hardships of evolving TCP, that requires updates to kernels (this way QUIC can evolve faster and with minimal effort).
  - Operating at user space allows for more sophisticated mechnisms to be employed.

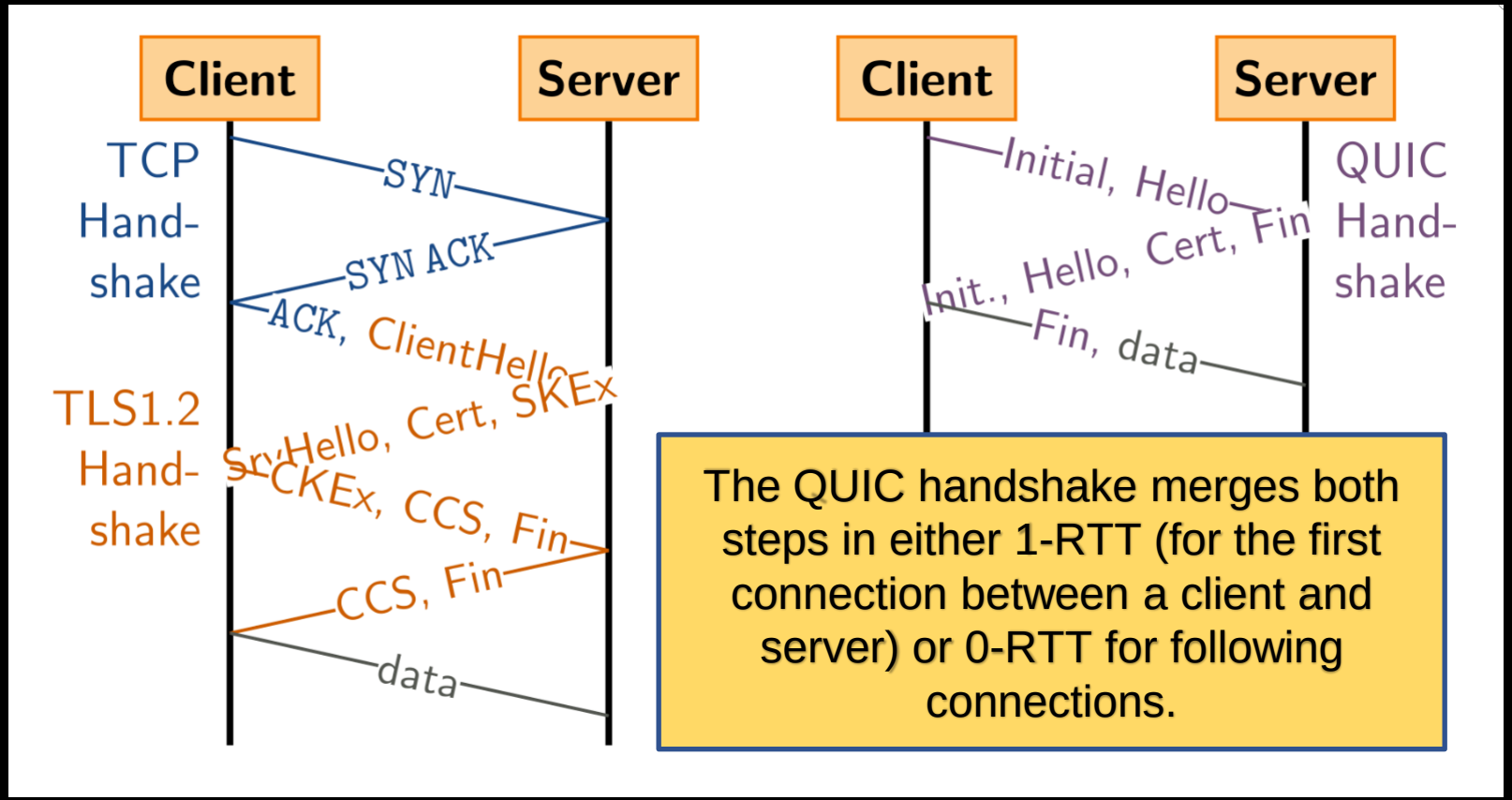
QUIC Handshake

- Most of the Web now-a-days is based on HTTP/2 and TCP.
- 3-Way handshake of TCP creates delays whenever a new connection is established.
- Most of the Web operate with https, which requires additional handshake packets for establishing the TLS connection.

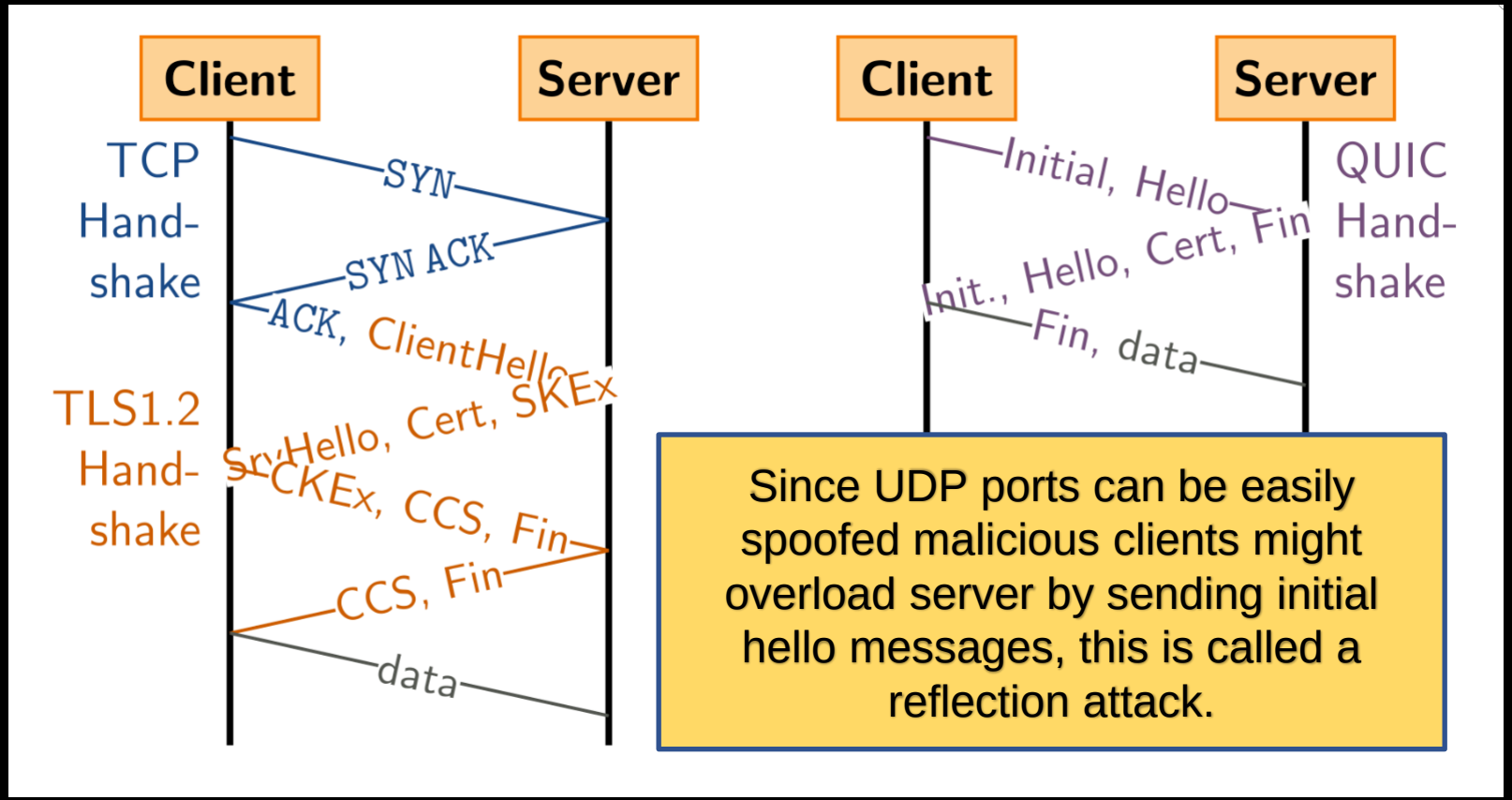
QUIC Handshake



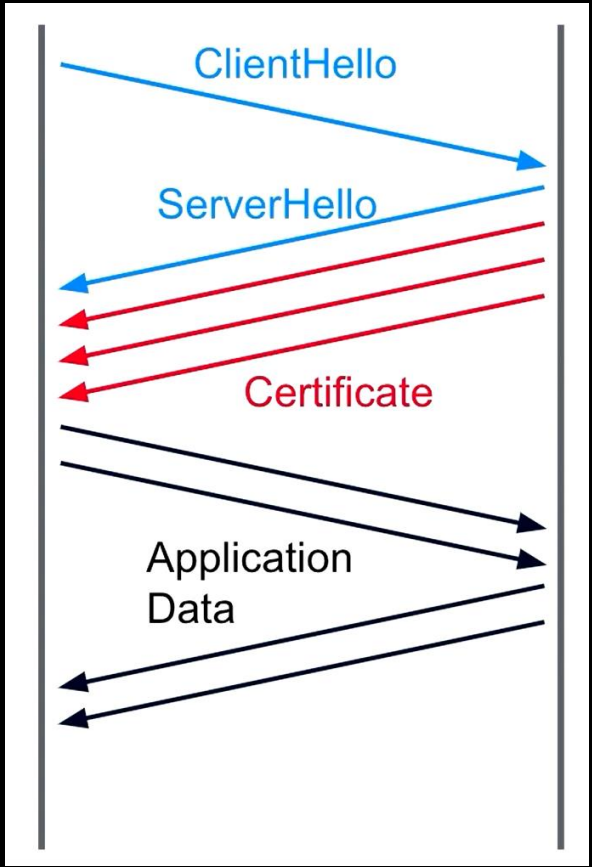
QUIC Handshake



QUIC Handshake

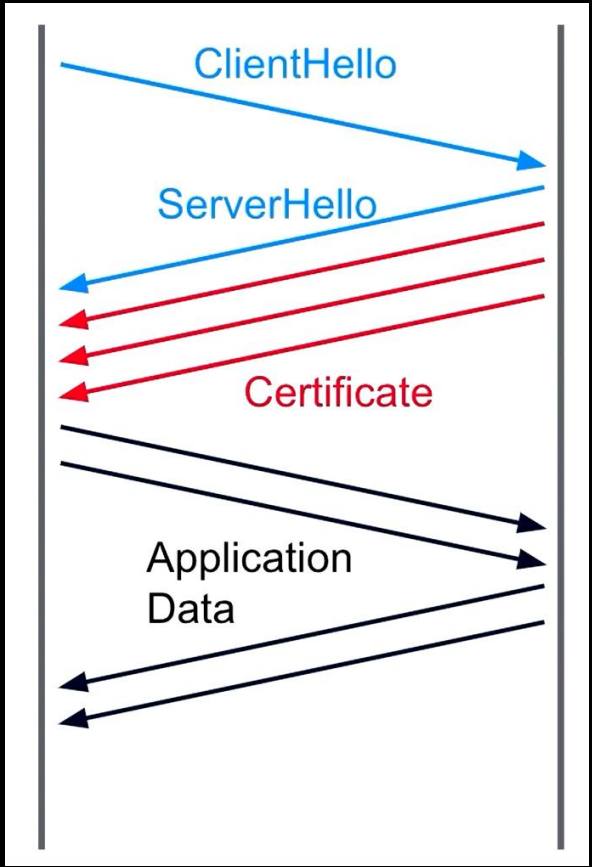


QUIC Handshake



A Reflection attack happens when a client can, with minimal effort, lead the server to consume a significant amount of resources (in this cases sending the certificates of the server).

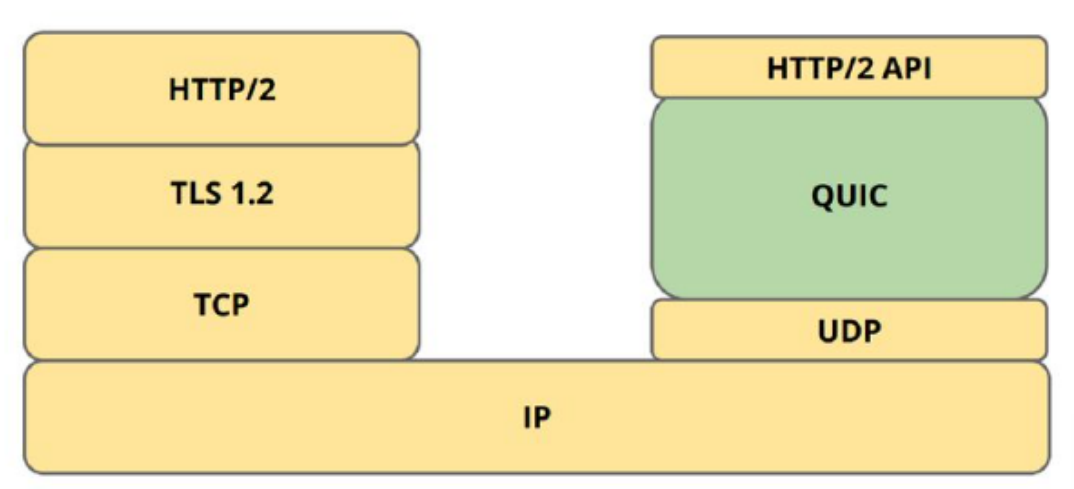
QUIC Handshake



Quick addresses this by forcing the client to always sending (random) data on its first message…

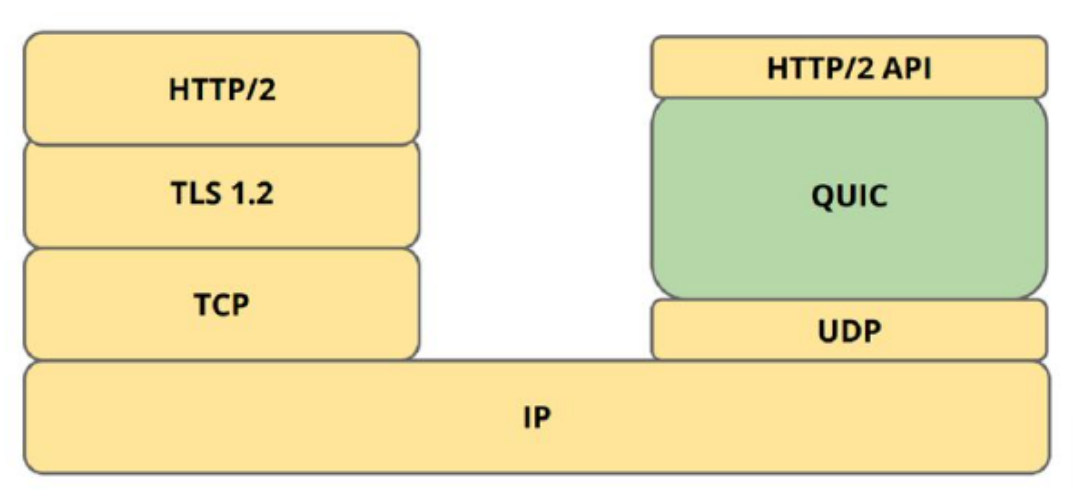
…and restricting the server to send at most 3 times the amount of data that was received from the client.

QUIC Common Stack



Quick effectively replaces the need for TLS 1.2, it multiplexes HTTP/2 requests over a single connection on its own and relies on UDP instead of TCP.

QUIC Common Stack



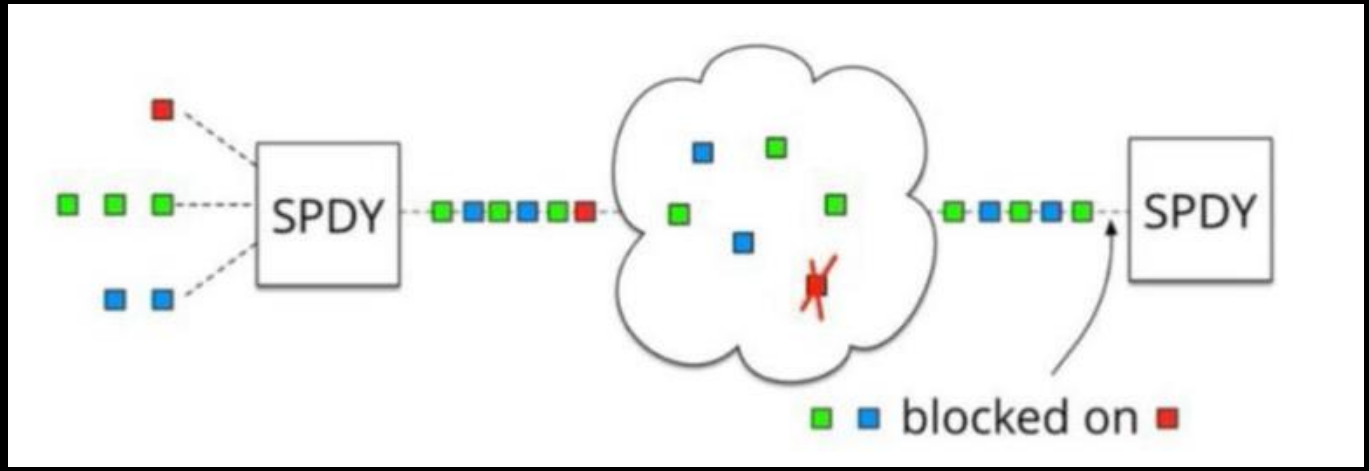
Quick effectively replaces the need for TLS 1.2, it **multiplexes HTTP/2 requests over a single connection** on its own and relies on UDP instead of TCP.

Why is this important?

QUIC Multiplexing

It’s all related with SPDY:

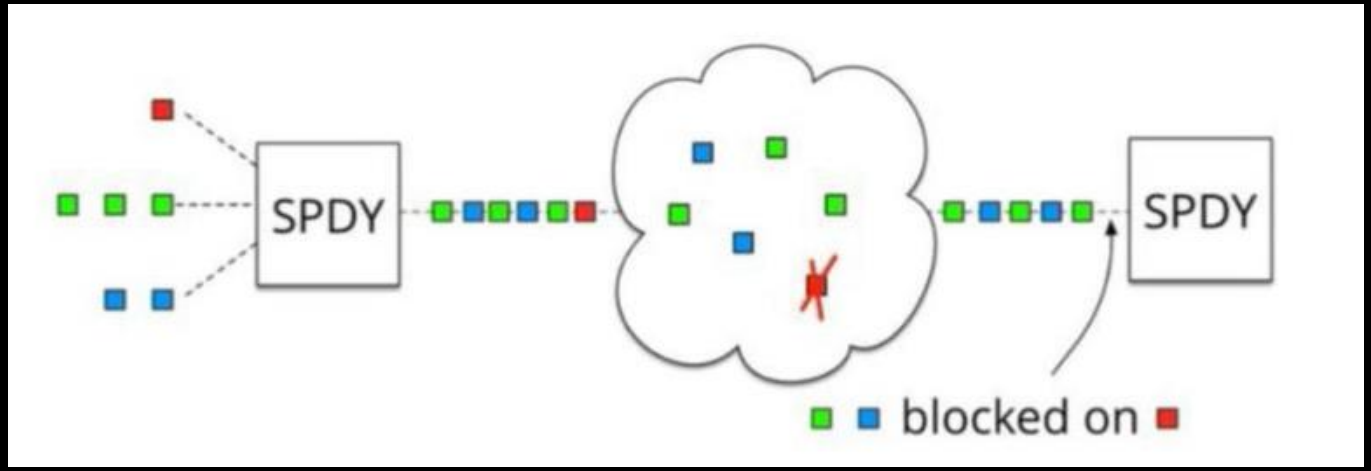
- SPDY + HTTP/2: A single TCP connection to fetch multiple data objects.



QUIC Multiplexing

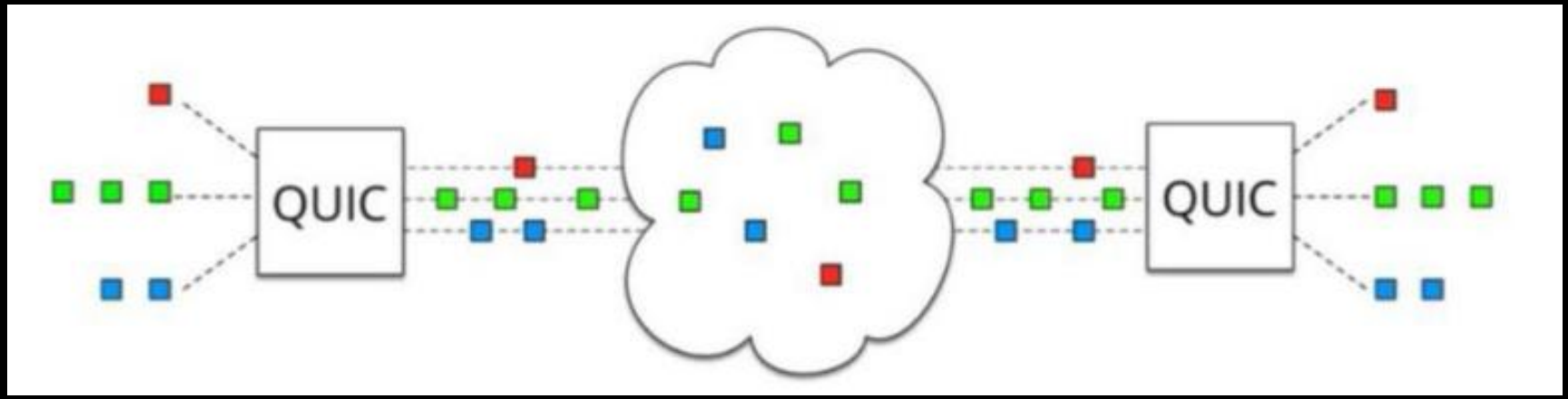
It’s all related with SPDY:

- But TCP requires packets to be delivered and processed in order, so if one packet is lost, all data objects fetches are blocked: head of line blocking.



QUIC Multiplexing

UDP does not require ordering of packets on delivery. Hence QUIC allows for a similar behaviour to SPDY but with multiplexed streams being fully independent (and one packet loss will only affect its own stream)



QUIC Optional Forward Error Correction (FEC)

Forward Error Correction is a technique that relies on the upfront transmission of redundant data when the connection is expected to show frequent packet losses.

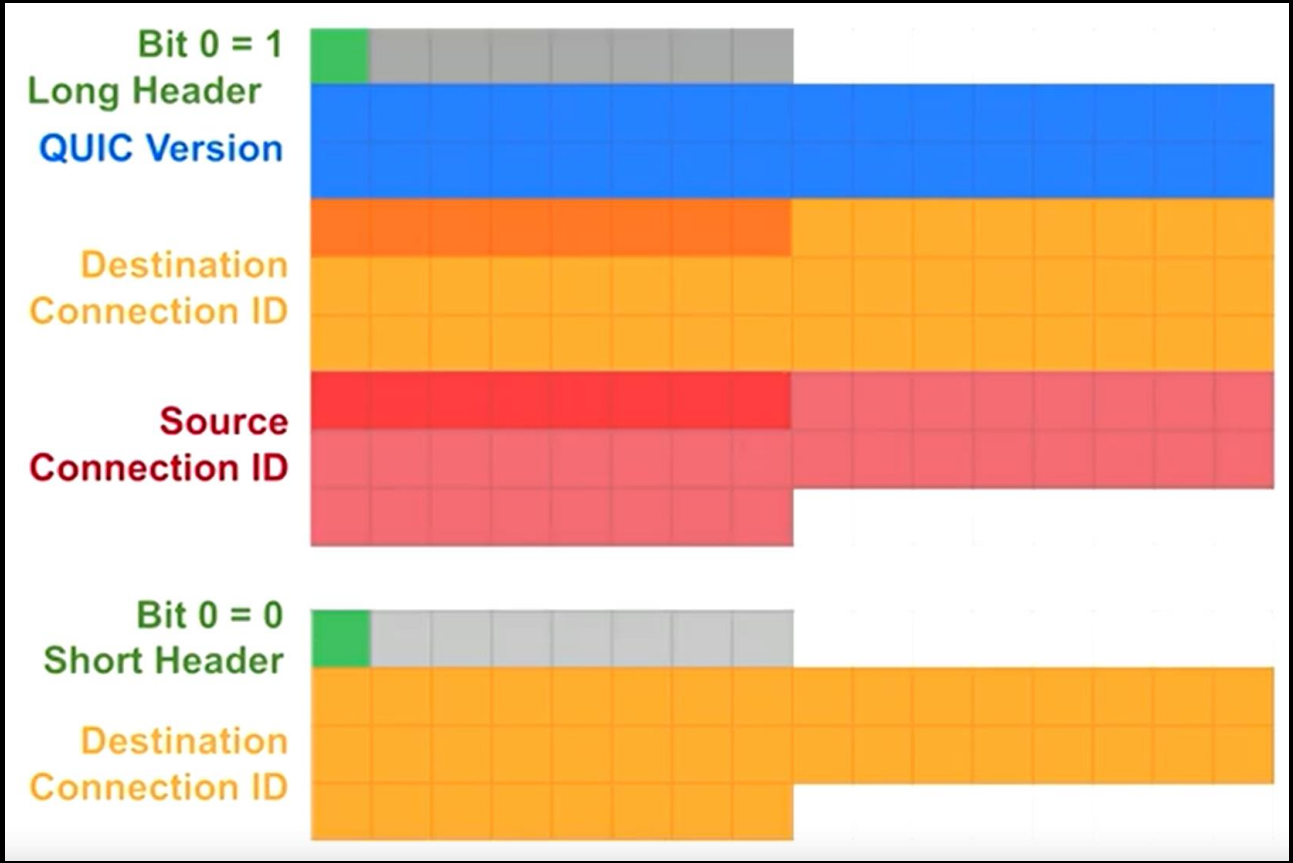
In QUIC this can be achieved by encoding redundant information on packets, such that, for instance, any 9 consecutive packets out of 10, allows to rebuild the entirety of the data sent on all packets.

QUIC Optional Forward Error Correction (FEC)

This operates akin to RAID but at the transport layer.

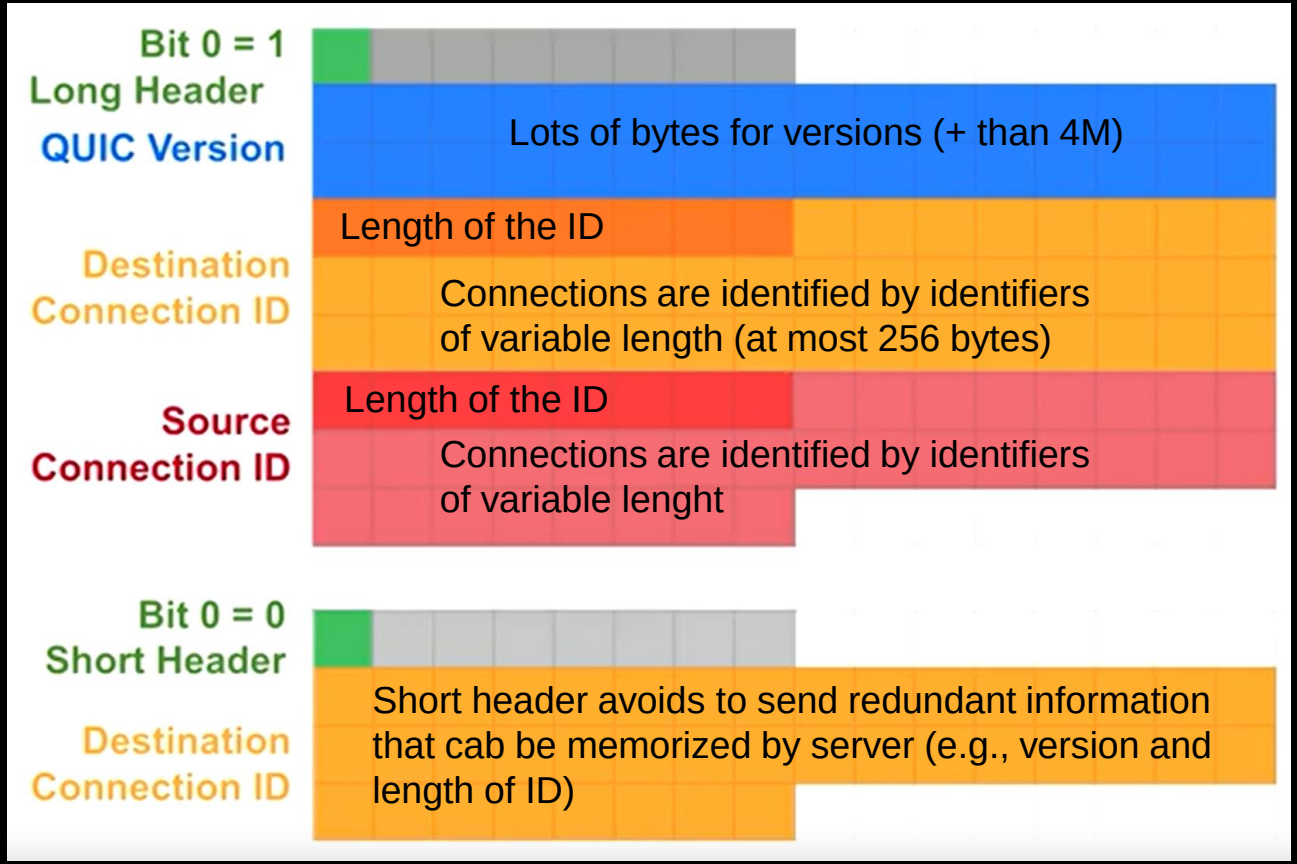
Trade-off: more payload transmitted VERSUS less retransmissions.

QUIC Header(s)



- QUIC relies on two different headers.
- A Long header that is used when a new connection is being established.
- A shorted header used for following packets.

QUIC Header(s)



- QUIC relies on two different headers.
- A Long header that is used when a new connection is being established.
- A shorted header used for following packets.

QUIC Header(s)

- Why is it that connection IDs are useful?

QUIC Header(s)

- Why is it that connection IDs are useful?

- First it allows the endpoints to maintain information about the data flow (identified by the connection ID), avoiding that information to be retransmitted.

QUIC Header(s)

- Why is it that connection IDs are useful?

- First it allows the endpoints to maintain information about the data flow (identified by the connection ID), avoiding that information to be retransmitted.

- **More importantly, it decouples the data flow from IP and port, enabling transparent migration of connections.**

QUIC Client Migration

- Imagine that you are using your cellphone connected to a WiFi access point to interact with a site, transferring multiple data objects…
- …and that you get far from the access point and lose the WiFi connection, leading your phone to migrate to your 4G/5G connection. What happens to your TCP connections?

QUIC Client Migration

- Imagine that you are using your cellphone connected to a WiFi access point to interact with a site, transferring multiple data objects…
- …and that you get far from the access point and lose the WiFi connection, leading your phone to migrate to your 4G/5G connection. What happens to your TCP connections?
- They will all timeout, and for each one you will have to restart it, going through the entire handshake process (lots of time and round trips).

QUIC Client Migration

- Imagine that you are using your cellphone connected to a WiFi access point to interact with a site, transferring multiple data objects…
- …and that you get far from the access point and lose the WiFi connection, leading your phone to migrate to your 4G/5G connection. What happens to your QUIC connections?

QUIC Client Migration

- Imagine that you are using your cellphone connected to a WiFi access point to interact with a site, transferring multiple data objects…
- …and that you get far from the access point and lose the WiFi connection, leading your phone to migrate to your 4G/5G connection. What happens to your QUIC connections?
- You will just retransmit your packed with your new IP/Port, and the server will answer, since it will identify the data flow based on the connection ID.

QUIC Client Migration

- Imagin
  connec t with a
  site, tra
- …and t t and
  lose th ne to
  migrate ppens to
  your QUIC connections?
- You will just retransmit your packed with your new IP/Port, and the server will answer, since it will identify the data flow based on the connection ID.

This is also only possible because QUIC operates on top of UDP datagrams, which allows the encryption of data to be independent from Datagram to Datagram (which does not happen with TCP which is stream based).

QUIC and TCP

- QUIC naturally was built following years of lessons learned from observing TCP in operation.
- It also features RTT prediction mechanisms, explicit acknowledgements, and retransmissions.
- But these were revised to become more flexible, for instance, QUIC supports many more different ways to encode acknowledges and negative acknoledgments.

QUIC and TCP

- QUIC r… essons learned…
- It also f… s, explicit acknow…
- But these were revised to become more flexible, for instance, QUIC supports many more different ways to encode acknowledges and negative acknoledgments.

We will not go into details on those mechanisms.

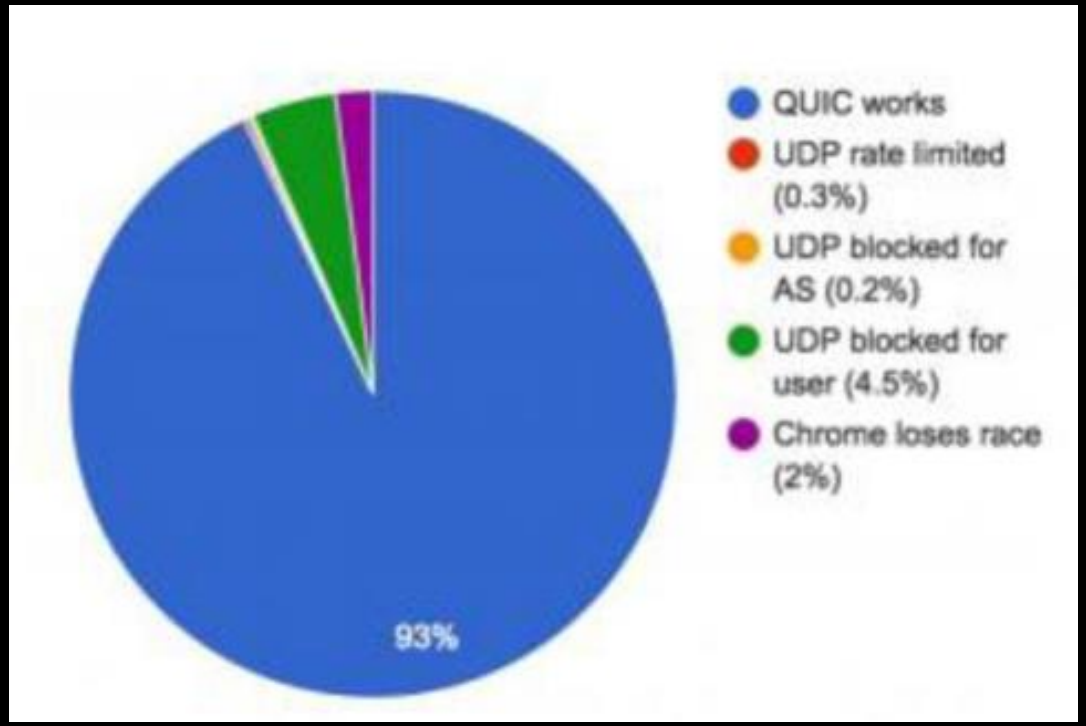
But network operators don’t like UDP

- Since UDP is not network friendly, many operators lower the priority of UDP traffic (meaning that those packets are the first to be dropped).
- Firewalls also not big fans of UDP packets…

- This is also reasonable since UDP does not provides any guarantees on delivery.

- What is the impact of these policies on QUIC?

But network operators don’t like UDP



Experiment conducted by Google in 2018.

But you are saving a few round trips only… so what?

But you are saving a few round trips only… so what?

- Experiments from Google, show that under poor network conditions, QUIC is significantly better than TCP.
- 1 second faster loading the Google Search Page for the slowest 1% connections.
- Reports also indicate that using QUIC allows youtube to show less 30% rebuffers (Video stalls).

The impact of QUIC

- Lots of traffic in the Internet is now supported by QUIC.
- Adopted by most browsers.
- Adopted by most big operators (Google, Facebook)

The impact of QUIC

- Lots of traffic in the Internet is now supported by QUIC.
- Adopted by most browsers.
- Adopted by most big operators (Google, Facebook)

- More importantly, QUIC has the potential to evolve faster now, since it operates within user space.

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

Bibliography:

- RFCs referenced throughout the slides.
- J. Saltzer, D. Reed ad D. Clark, “End-to-End Arguments in System Design,” ACM Transactions on Computer Systems (4):277-288, November 1984.
- Wikipedia for details on QUIC.

Videography:

- QUIC DEEP DIVE by Marten Seemann (Paris P2P Festival April 2022)

https://www.youtube.com/watch?v=6SyDP7xKqZk&list=PLNeNFYqVeWnNy8KdZOdOTlzSkKoBWyfqO&index=22