Advanced Computer Networking

CYBR 230 – Jeff Shafer – University of the Pacific

QUIC

It’s a Google thing….

(Originally)

Google Engineering Motivations

- **Goal: Decrease end-user latency on web**
  - *To increase user engagement…*
  - *So they see more ads…*
- Approaches
  - Expand Google's content delivery network to be physically closer to audience
    - Fewer network hops, fewer wire delays
  - Develop and optimize web browser (Chrome)
  - Update HTTP protocol (HTTP/2)

Google Engineering Motivations

- **HTTP/2** (based on Google SPDY)
- Decrease end-user latency via
  - Compressing HTTP headers (fewer bits)
  - Pipelining requests
  - Multiplexing many HTTP requests onto single TCP connection
  - Allowing server to push *anticipated* content to users

How do you make the web faster?

Google

User-perceived latency

$BROWSER

HTTP/1.1

TLS 1.2

TCP

IP

Physical Network

google.com

Launch your own browser

Update HTTP

Chrome

HTTP/2

What about this part of the stack?

Build a carrier-grade network

Google CDN

google.com

https://www.nanog.org/sites/default/files/meetings/NANOG64/1051/20150603_Rogan_Quic_Next_Generation_v1.pdf

Google Engineering Motivations

- Problems demonstrated in HTTP/2 testing
  - TCP is **in-order delivery** protocol
    - All packets are precious!
    - **Head-of-Line problem:** Loss of a single packet prevents delivery of all behind it until (slow) retransmission occurs
  - If multiple streams are being sent through *single* TCP connection, all are delayed
- Can we do better?
  - Challenge: TCP is baked into the operating system kernel (Windows, OS X, Linux) – Difficult for even Google to modify

Transmission Control Protocol (TCP)

- TCP is connection-oriented
  - 3-way handshake used for connection setup
- TCP provides a stream-of-bytes service
- TCP is reliable
  - Acknowledgements indicate delivery of data
  - Checksums are used to detect corrupted data
  - Sequence numbers detect missing, or mis-sequenced data
  - Corrupted data is retransmitted after a timeout
  - Mis-sequenced data is re-sequenced
  - (Window-based) Flow control prevents over-run of receiver
- TCP uses congestion control to share network capacity among users

QUIC Overview

- **Q**uick **U**DP **I**nternet **C**onnections
  - Roughly, TCP+TLS+HTTP/2 on top of UDP
- Design Goals
  - Provide multiplexed connections between two hosts (without head-of-line blocking)
  - Provide security (equivalent to TLS) – always encrypted
  - Reduce connection establishment latency
  - Improve congestion control
  - Provide bandwidth estimation to applications to avoid congestion
  - "Innovate" at the userspace (not constrained by OS kernel, legacy clients, middleboxes)

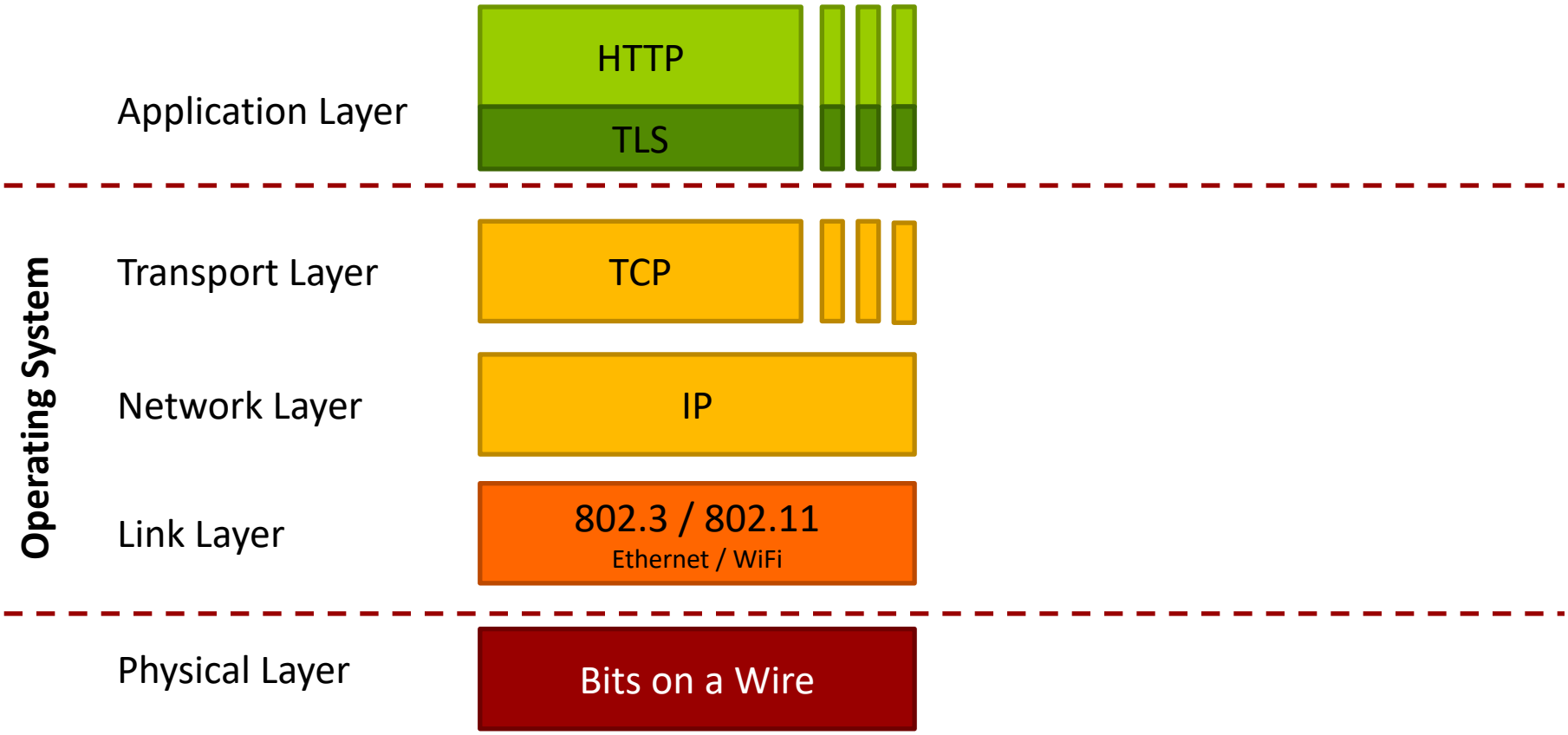
User Datagram Protocol (UDP)

- UDP is a connectionless datagram service
  - There is no connection establishment: packets may show up at any time
- UDP packets are self-contained
- UDP is unreliable
  - No acknowledgements to indicate delivery of data
  - Checksums cover the header, and only optionally cover the data
  - Contains no mechanism to detect missing or mis-sequenced packets
  - No mechanism for automatic retransmission
  - No mechanism for flow control or congestion control (sender can overrun receiver or network)

Traditional Architecture

HTTP/1.1 on top of TCP

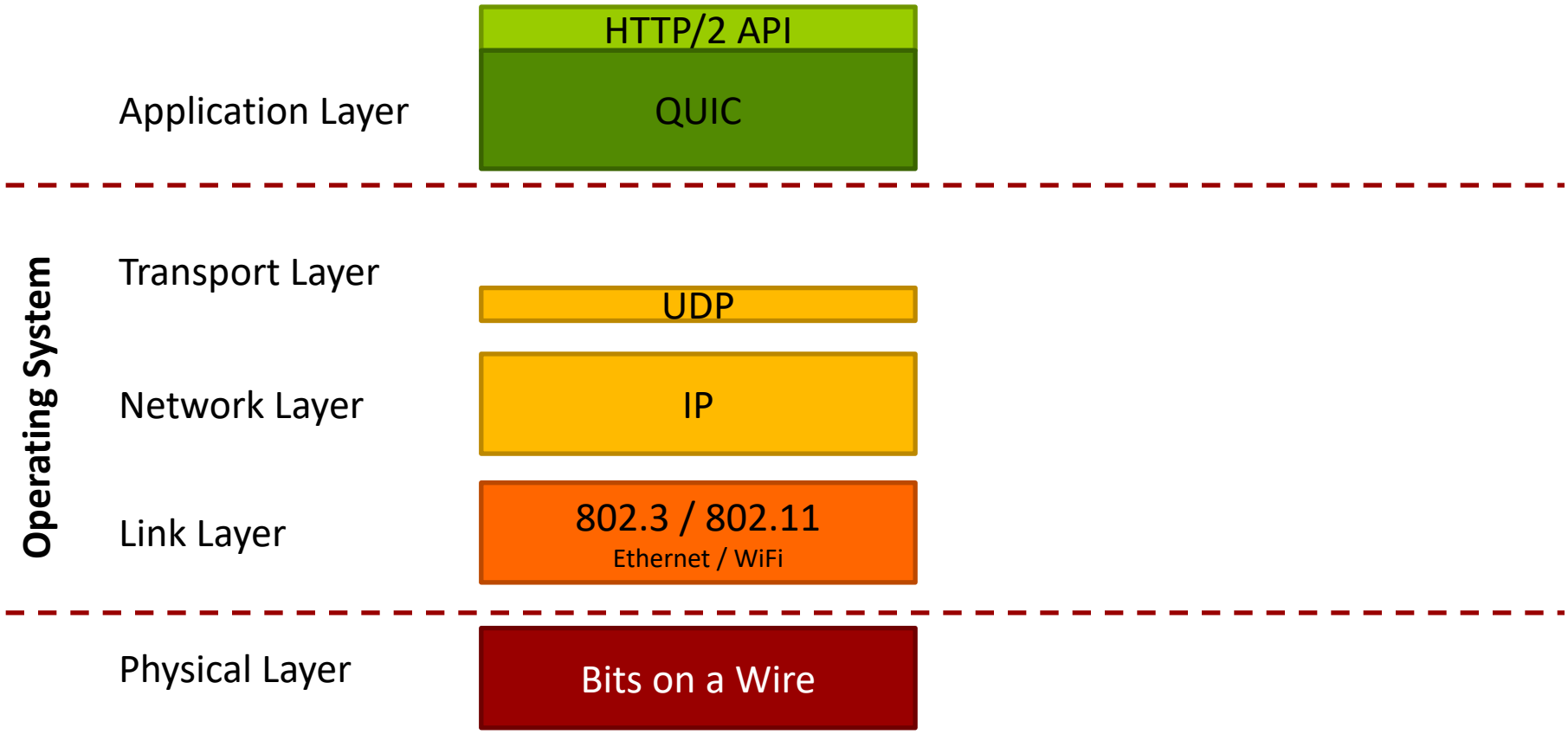
Multiple connections to same server (for parallelism) require multiple TCP sockets



QUIC Architecture

HTTP/2 on top of QUIC

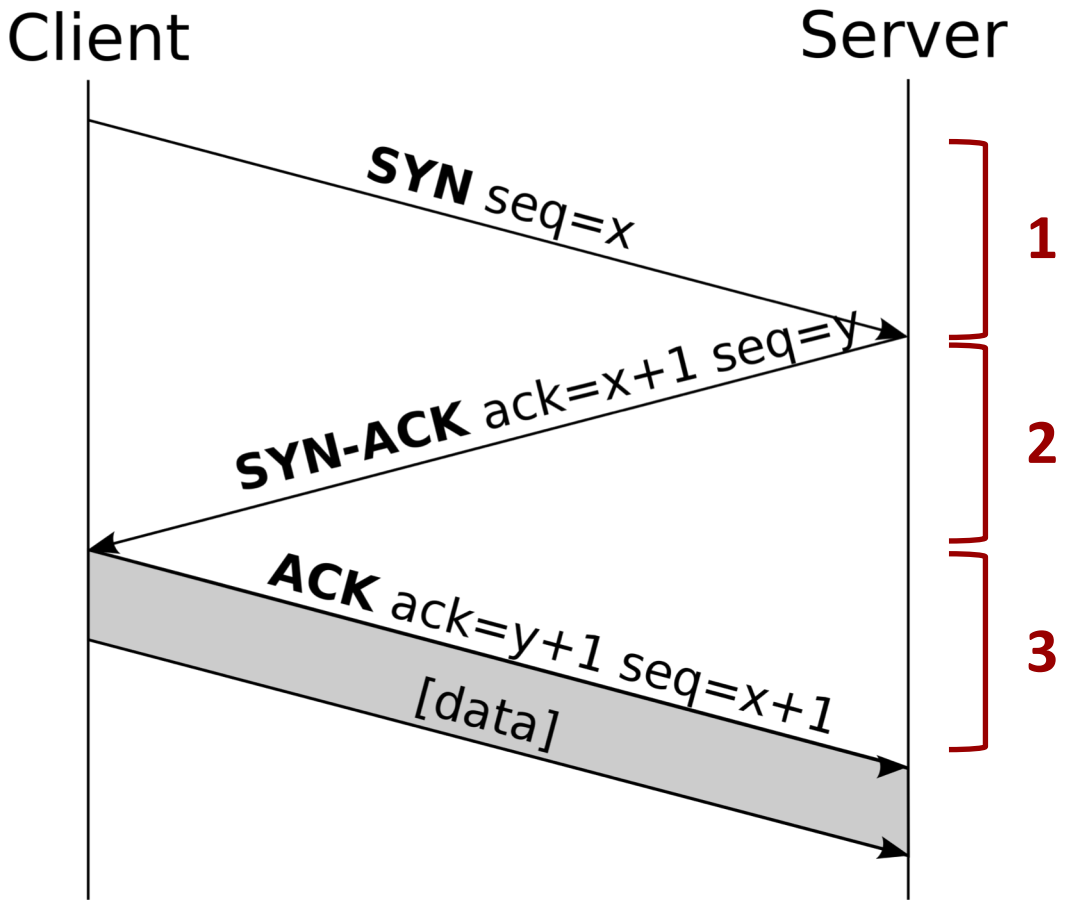
Single UDP stream presented to operating system (multiplexed)



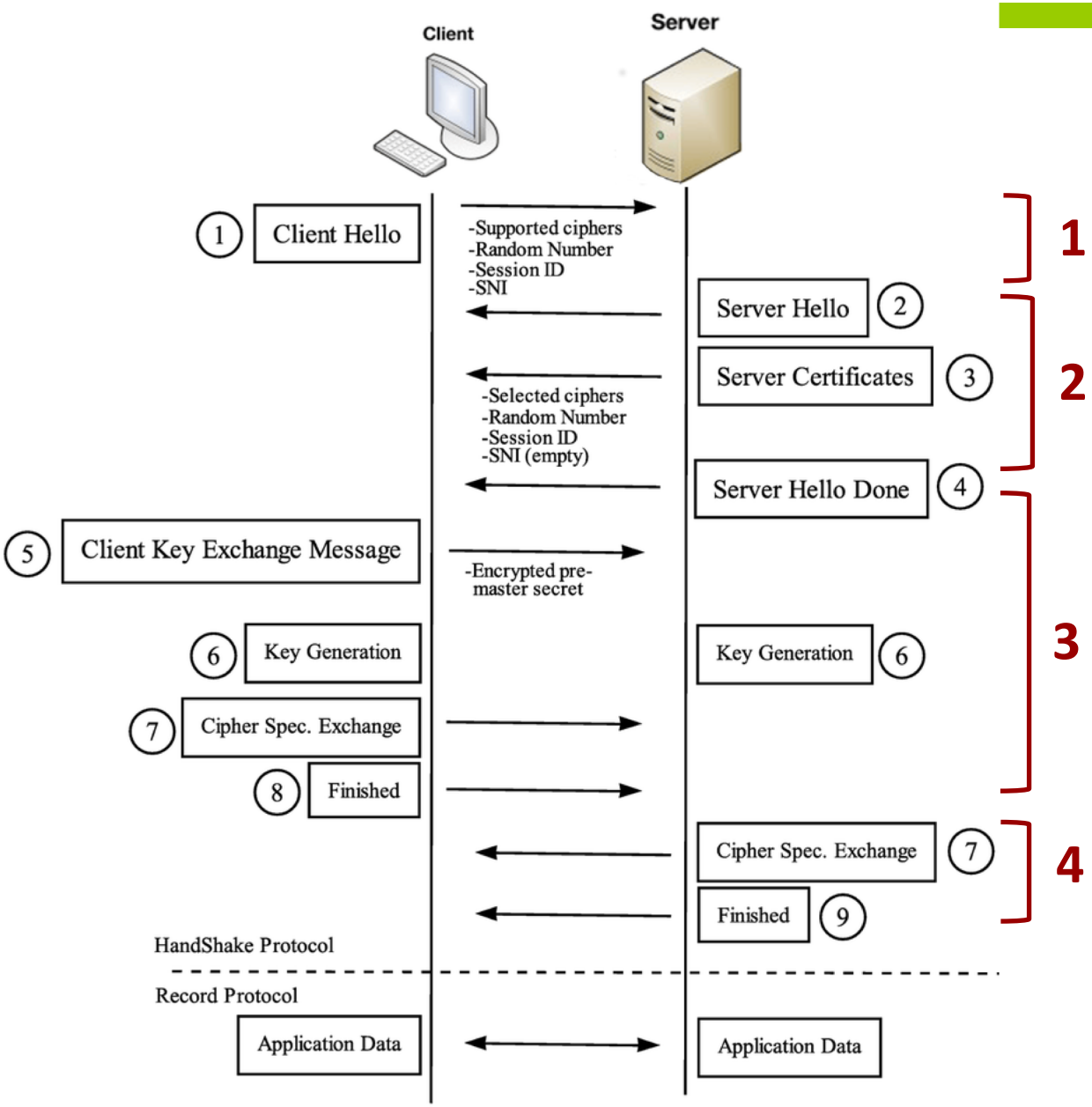
QUIC vs TCP

- QUIC uses server UDP port 443 instead of TCP 443
- QUIC is fully encrypted by default
  - Except for flags, connection ID, and sequence number
  - Avoids network ossification by “helpful” network operators and middleware boxes
- QUIC retransmits data with new sequence numbers and re-encrypts them
  - Improves loss recovery and RTT measurement
- QUIC has no head of line blocking
  - Only the stream with the missing packet is blocked
  - All other streams can continue

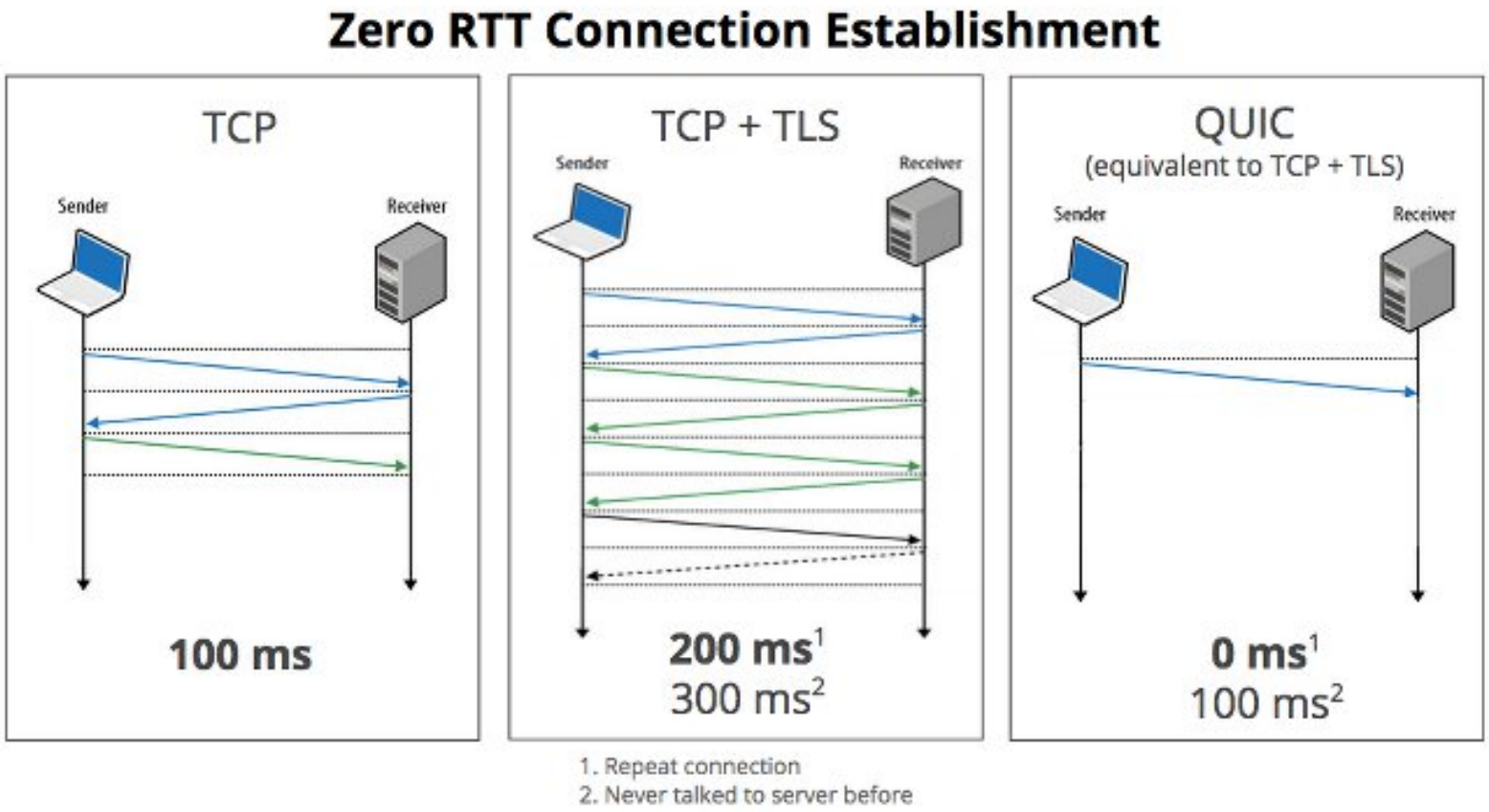
TCP 3-Way Handshake



TLS Handshake



QUIC vs TCP



https://peering.google.com/#/learn-more/quic

QUIC Zero RTT Connections

- QUIC Connection Setup
  - 0 round-trips to a known server (common)
  - 1 round-trip if crypto keys are not new
  - 2 round-trips if QUIC version negotiation needed
  - After setup, HTTP requests/responses flow over connection
- QUIC inspired TLS 1.3 Zero RTT handshake

https://datatracker.ietf.org/meeting/98/materials/slides-98-edu-sessf-quic-tutorial-00.pdf

QUIC Congestion Control

- QUIC builds on decades of experience with TCP
- Incorporates TCP best practices
  - TCP Cubic - fair with TCP FACK, TLP, F-RTO, Early Retransmit...
- Adds signaling improvements that can't be done to TCP
  - Retransmission uses a new sequence number
  - Avoid ambiguity about which packets have been received

Mobile

- QUIC better supports mobile clients
  - Handing off between WiFi and cell network
  - Switching apparent IP addresses
- QUIC token allows a client to continue with an established connection even if the IP address changes

Implementation

- Clients
  - Google Chrome (from 2012+)
  - Google mobile apps (YouTube, 2016+)
  - Opera 16
- Servers
  - Google servers (proprietary)
  - Other servers in development
- Default transport protocol between Google clients and Google servers

Performance

- Strong network connection?
  - 3% latency improvement
- Weak network connection?
  (99% percentile of connections to Google search)
  - Reduced page loading time by **1 second**
  - Strong benefit over TCP on marginal internet connections (third world/emerging markets, high latency satellite Internet, lousy mobile devices over weak WiFi, etc…)
- YouTube
  - 30% fewer rebuffers (video pauses)

Standardization

- Internet Engineering Task Force (IETF) Working Group formed in 2016
  - https://quicwg.org/
  - Development/standardization ongoing
- Not just adopting Google QUIC
  - Google's QUIC was an experiment, tested on a large scale, yielding valuable data
  - Replacing "Google crypto" with TLS 1.3
  - Standardizing APIs
  - New packet format (long headers and short headers)
- Need to distinguish between gQUIC (Google QUIC) and upcoming iQUIC (IETF QUIC) standard