QUIC: Quick UDP Internet Connections

David Hasselquist

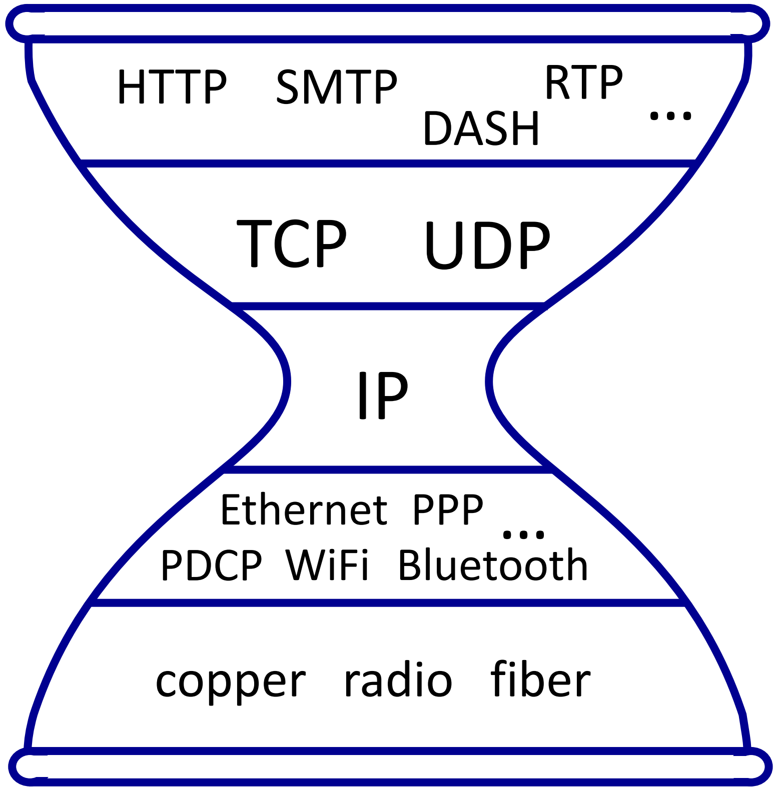
Slides partially borrowed/inspired by [1] – [4]

Recap: UDP

- Unreliable protocol
  - Simple, connectionless best effort service
  - Packets may be lost or duplicated
  - Packets may be delivered out of order

- Used by
  - Real-time communication
  - Multimedia applications
  - VoIP applications
  - Many protocols
    - DNS (Domain Name Service)
    - SNMP (Simple Network Management Protocol)
    - NTP (Network Time Protocol)
    - …

The IP hourglass: Internet's "thin waist"

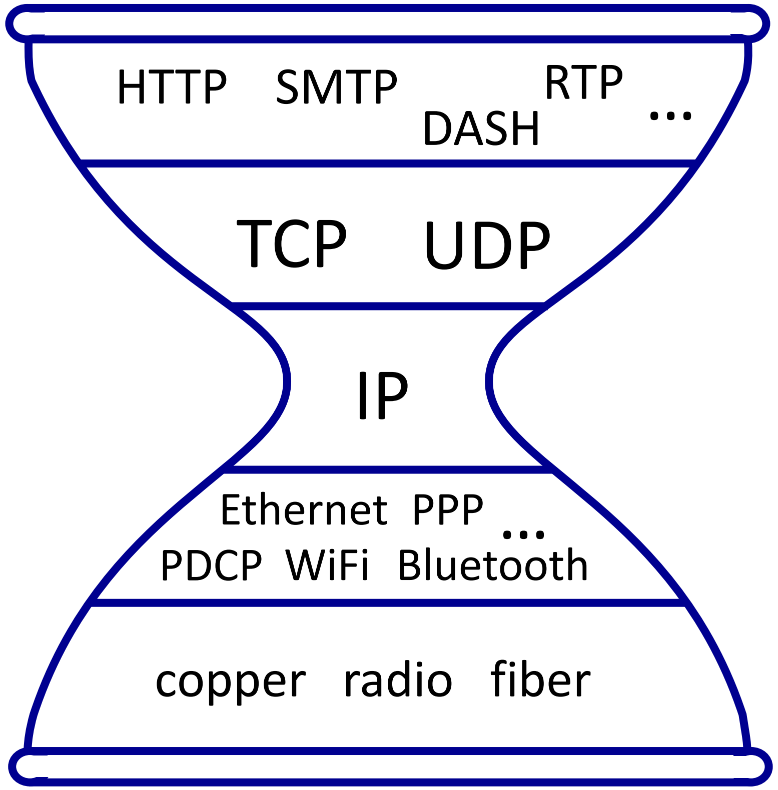


Recap: UDP

» Advantages
  » Simple
  » No setup/handshake needed (no RTT incurred)
  » Helps with reliability (checksum)

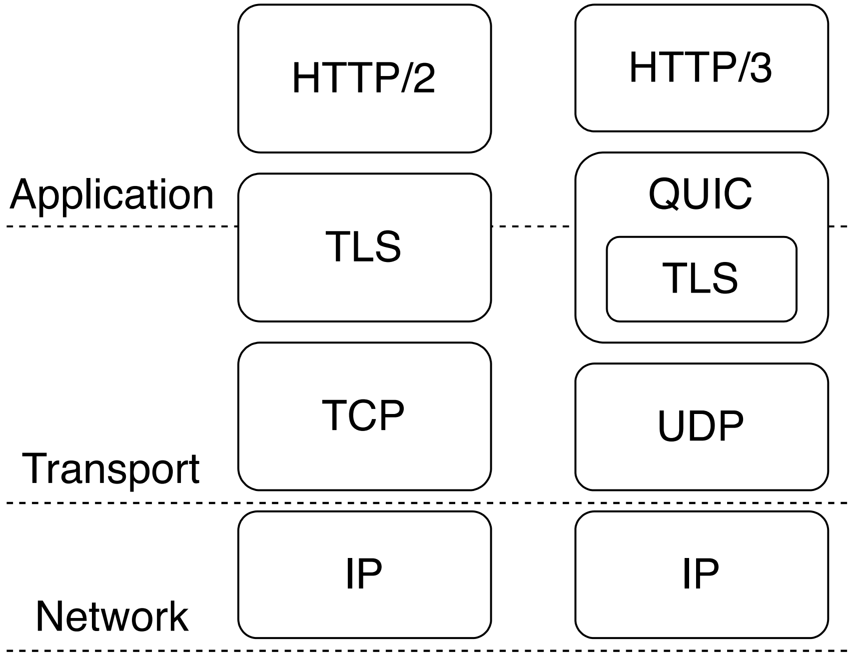
» If reliable transfer needed
  » Add additional functionalities on top of UDP at the application layer (e.g., reliability, congestion control)

The IP hourglass: Internet's "thin waist"



What is QUIC?

- New network transport protocol
  - Introduced by Google for HTTPS transport
- Reliability built in user-space on top of UDP
  - Connection establishment
  - Loss detection and error control
  - Congestion control
  - Authentication and encryption

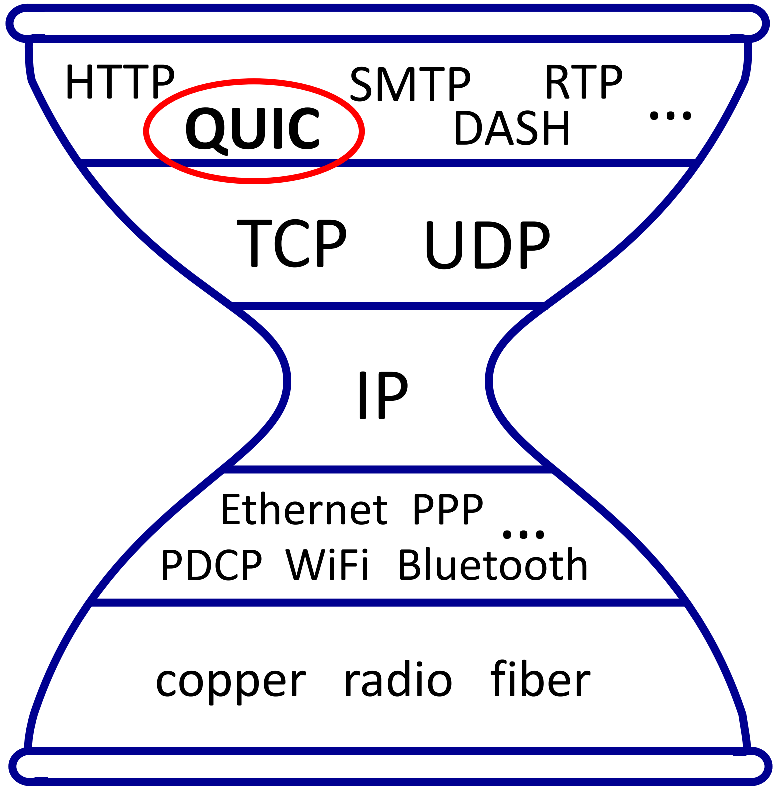


- From QUIC specification:
  - “Readers familiar with TCP’s loss detection and congestion control will find algorithms here that parallel well-known TCP ones.”

What is QUIC?

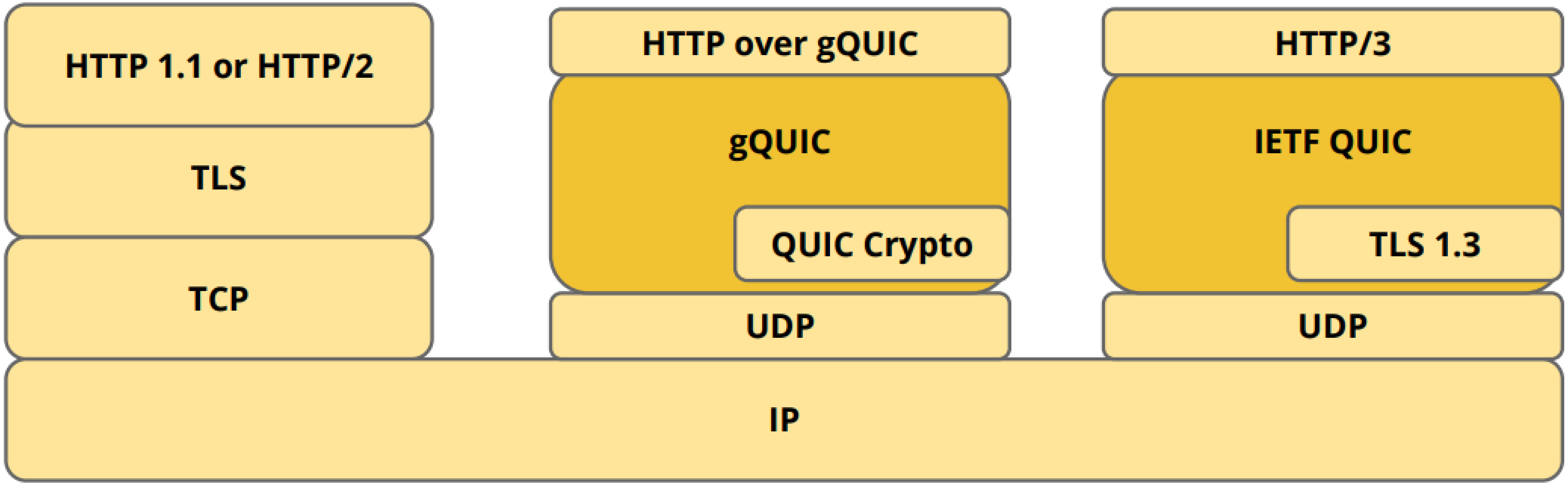
- Application layer protocol
  - Often said to be multilayer
- Improves application and HTTP performance
  - YouTube Video Rebuffers: 15 – 18%
  - Google Search Latency: 3.6 – 8%
  - Mostly for high latency users
- Widespread adoption
  - Google, Apple, Facebook, Microsoft
  - YouTube, Facebook, Instagram, Teams, Office
  - Chrome, Firefox, Edge, Safari
  - iOS, MacOS
  - …

The IP hourglass: Internet's "thin waist"

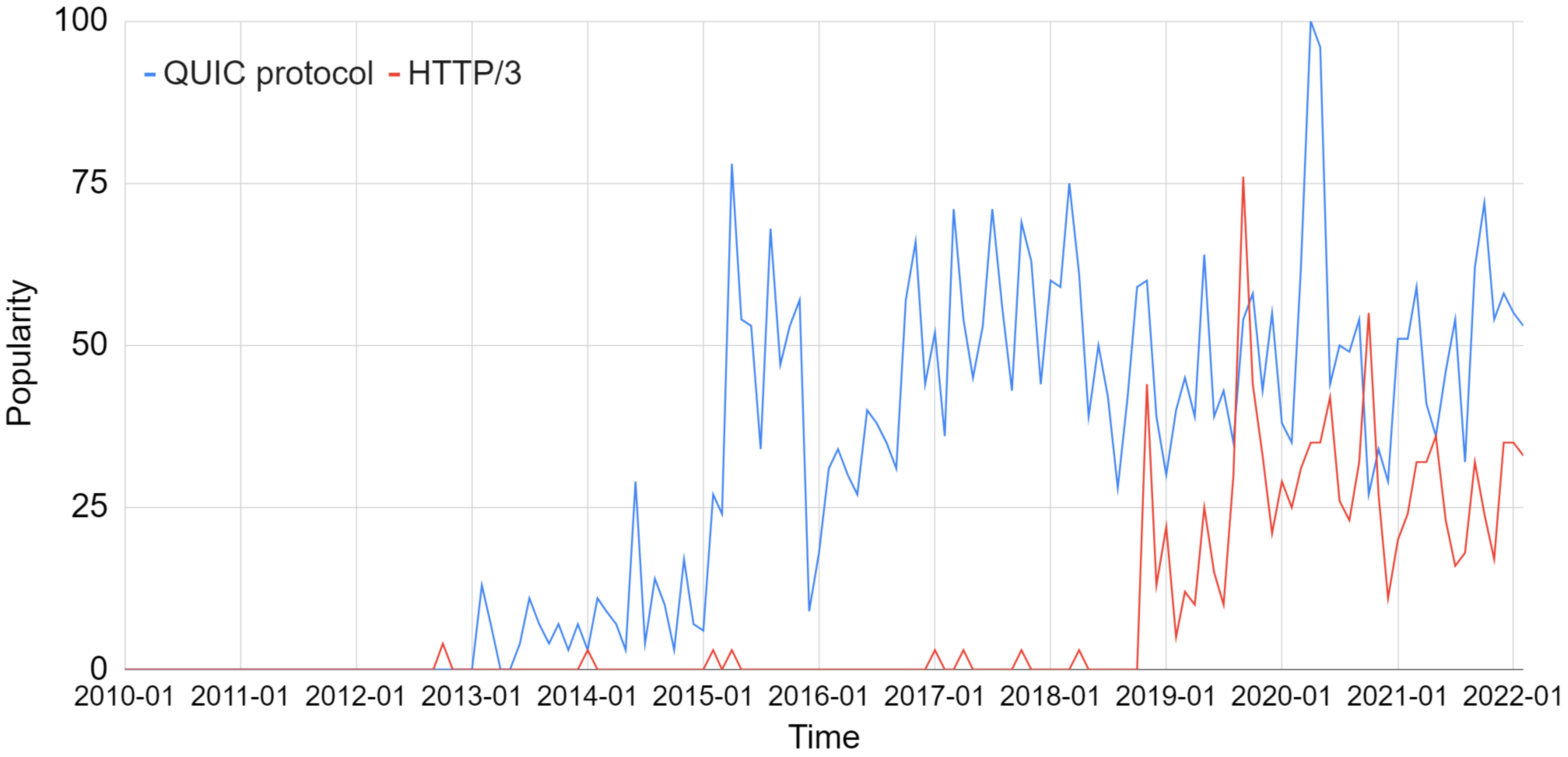


What is QUIC?

- Originally gQUIC
- Continued by the IETF to become HTTP/3 standard



Google Search trends



Issues with TCP

- Improvements have not seen wide deployment
  - Middleboxes key issue
    - Packet headers not protected end-to-end
    - Firewalls block unfamiliar traffic
    - Network Address Translators (NATs) rewrite transport header
  - Built into OS, requires new OS version

- Updates take long time to cascade
  - Many different versions in use
  - Simple protocol changes take up to a decade to see significant deployment

Issues with TCP

- Handshake delay (long connection setup)
  - Costs of TCP+TLS layering
  - Network bandwidth has increased, but speed of light remains constant
  - Most connections on the internet are short transfers
- Head-of-line blocking

QUIC features

- Deployability and evolvability
- Low-latency secure connection establishment
- Streams and multiplexing
- Better loss recovery and flexible congestion control
- Resilience to NAT-rebinding

QUIC features

- Deployability and evolvability
- Low-latency secure connection establishment
- Streams and multiplexing
- Better loss recovery and flexible congestion control
- Resilience to NAT-rebinding

Deployability and evolvability

- Google controls both server and client software
  - Google services and Chrome / mobile apps
  - Enables rapid deployment
  - Internet as the testbed

- Important to be deployable on the internet as is
  - Nodes, routers, firewalls view a QUIC packet as a regular UDP packet
  - Choice of UDP → user-space

- Important to be able to quickly change the protocol
  - Avoid dependence on vendors and network operators
  - Encrypt and authenticate as much as possible → middleboxes cannot tamper
  - Rapidly deploy new versions
  - Rapidly fallback to TCP if something goes wrong

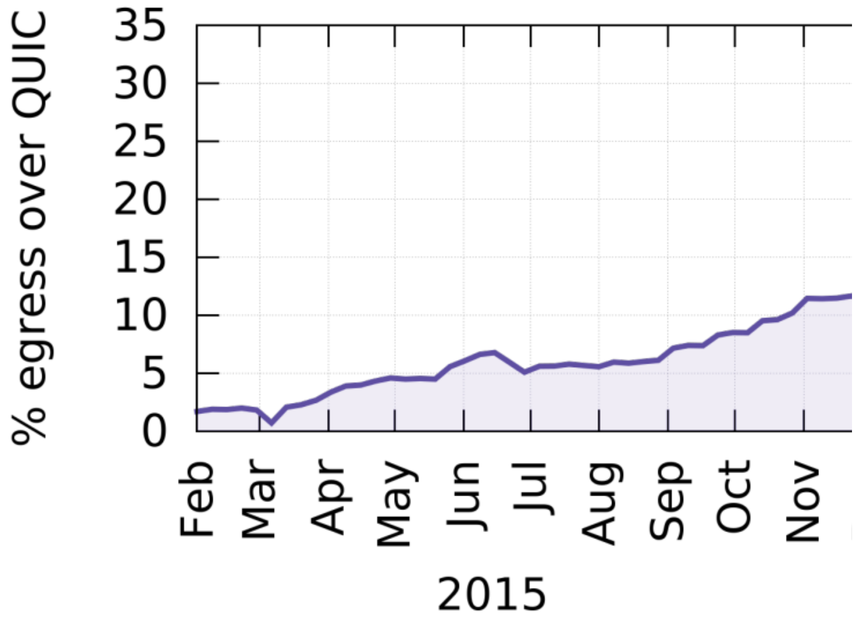
QUIC deployment

- 2013 – Introduction by Google
  - Enabled for Google dev team

- 2014 – Small scale testing
  - Enabled for less than 0.025% of Google Chrome users

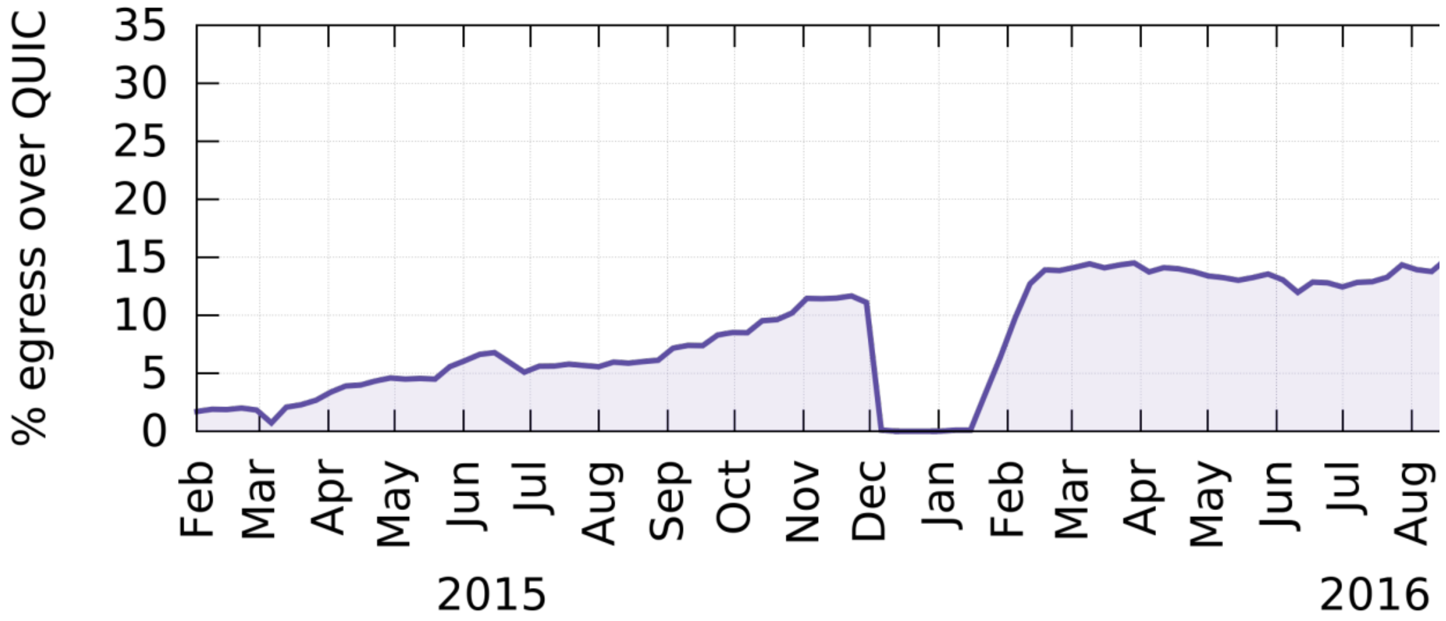
QUIC deployment

- 2013 – Introduction by Google
  - Enabled for Google dev team
- 2014 – Small scale testing
  - Enabled for less than 0.025% of Google Chrome users
- 2015 – Large scale deployment
  - 11% of Google's egress traffic



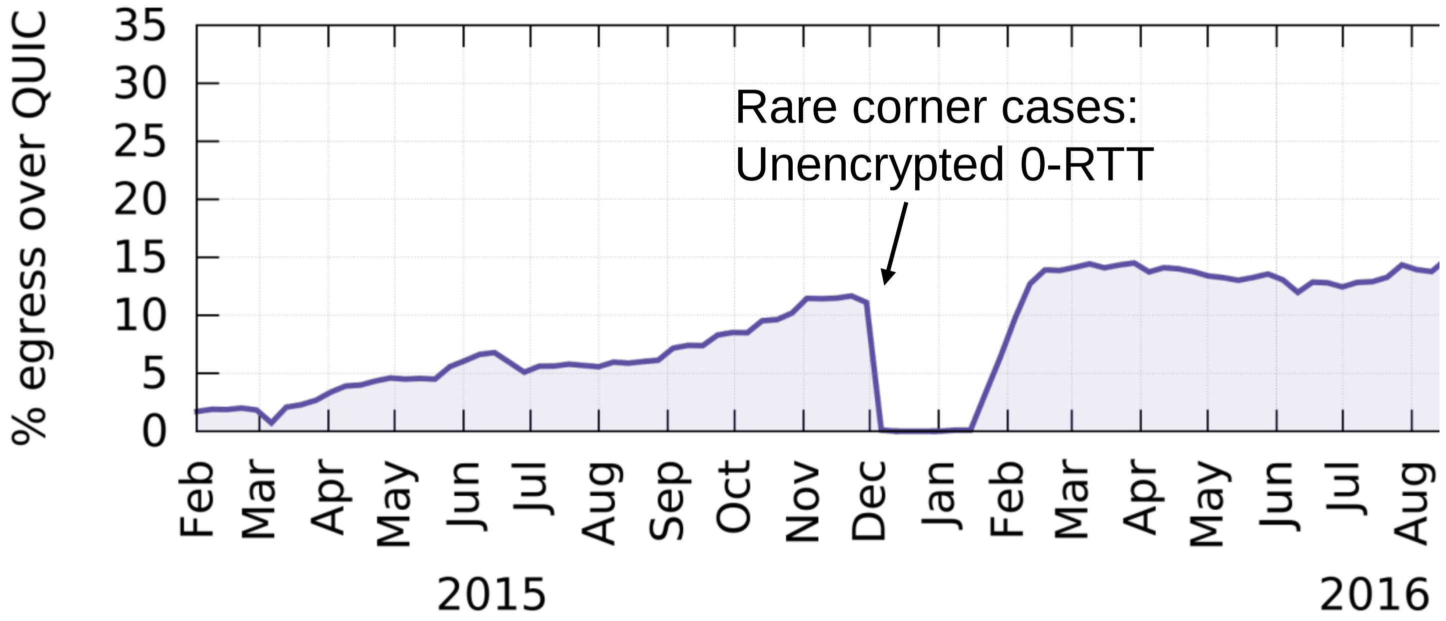
QUIC deployment

- 2013 – Introduction by Google
  - Enabled for Google dev team
- 2014 – Small scale testing
  - Enabled for less than 0.025% of Google Chrome users
- 2015 – Large scale deployment
  - 11% of Google's egress traffic



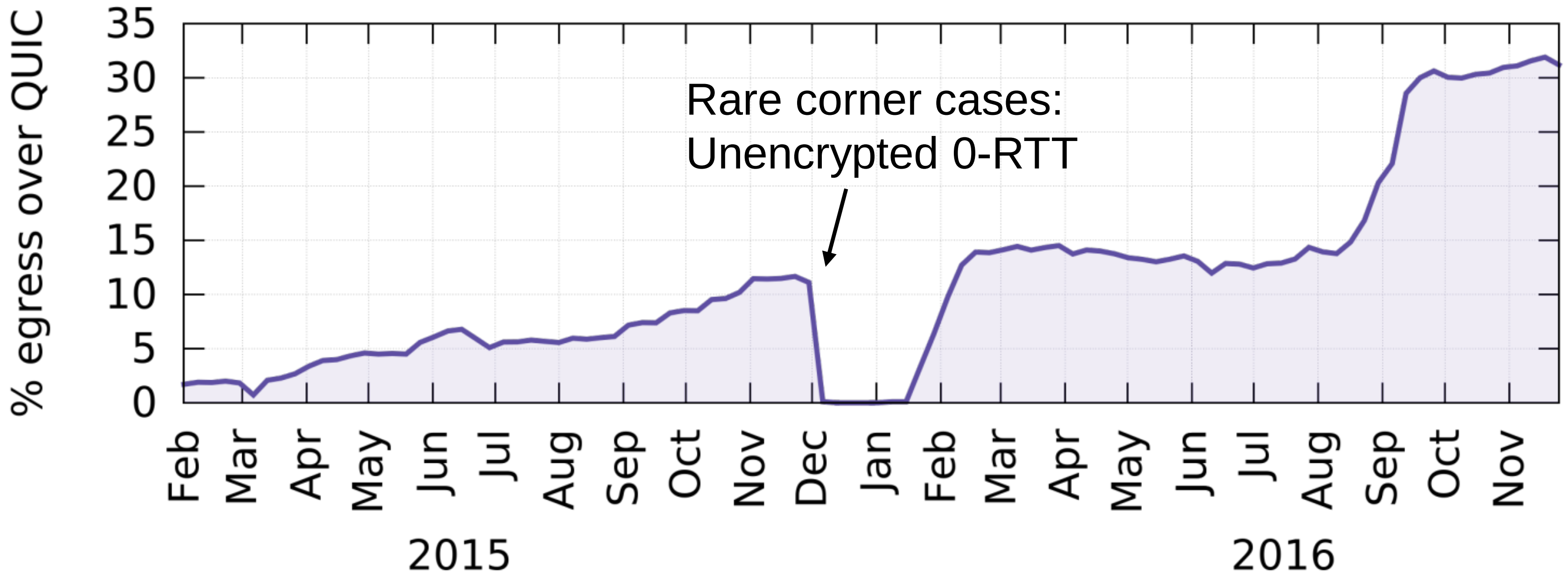
QUIC deployment

- 2013 – Introduction by Google
  - Enabled for Google dev team
- 2014 – Small scale testing
  - Enabled for less than 0.025% of Google Chrome users
- 2015 – Large scale deployment
  - 11% of Google's egress traffic



QUIC deployment

- 2013 – Introduction by Google
  - Enabled for Google dev team
- 2014 – Small scale testing
  - Enabled for less than 0.025% of Google Chrome users
- 2015 – Large scale deployment
  - 11% of Google's egress traffic
- 2016 – Larger scale deployment
  - Over 30% of Google's egress traffic (estimated to be 7% of all internet)

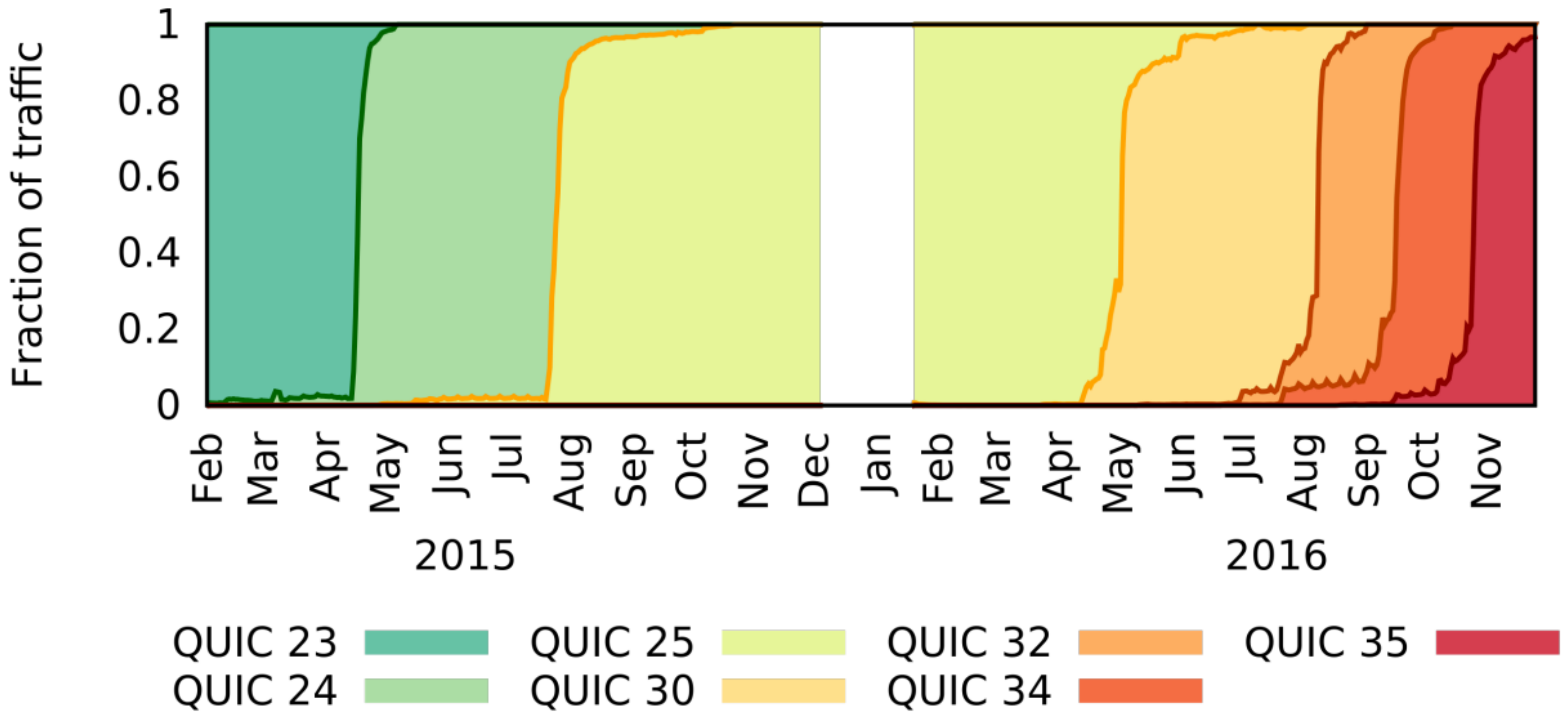


QUIC deployment

- 2019
  - Over 50% of Google’s egress traffic
  - Over 80% of Facebook’s API requests from the primary mobile application
  - Support for cURL
  - …
- 2020
  - Support for Safari
  - Facebook migration of apps and server infrastructure – 75% of all its traffic
  - …
- 2021
  - Support for Firefox
  - …
- 2022: Widespread adoption
  - Google, Apple, Facebook, Microsoft
  - YouTube, Facebook, Instagram, Teams, Office
  - Chrome, Firefox, Edge, Safari
  - iOS, MacOS
  - …

QUIC deployment versions

» Rapid deployment and evolution



QUIC features

- » Deployability and evolvability
- » Low-latency secure connection establishment
- » Streams and multiplexing
- » Better loss recovery and flexible congestion control
- » Resilience to NAT-rebinding

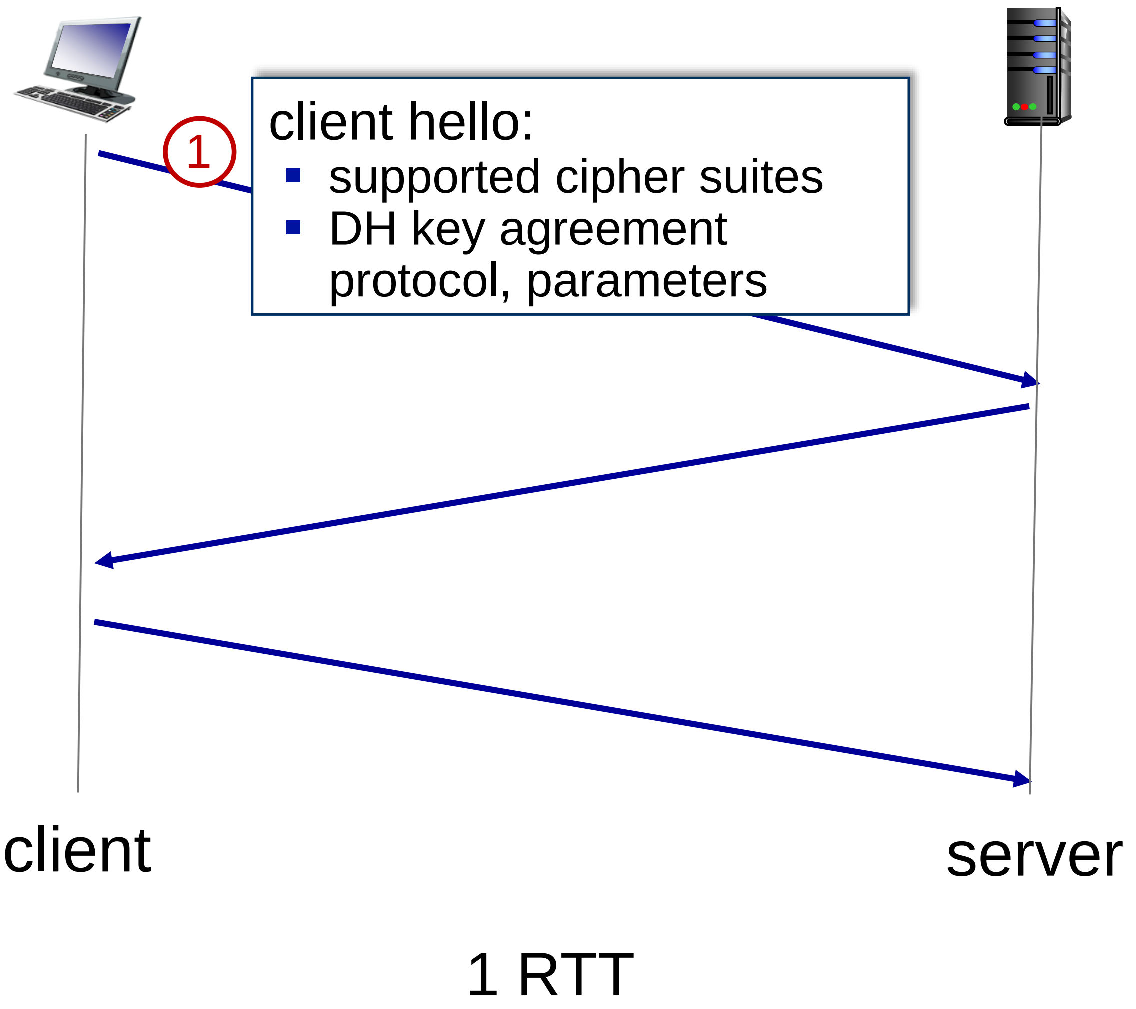
Recap: TLS

- Widely deployed security protocol above the transport layer
- Supported by all major browsers and web servers
- Provides an API that any application can use
- Provides
  - Confidentiality: via symmetric encryption
  - Integrity: via cryptographic hashing
  - Authentication: via public key cryptography
- Evolution from SSL

Recap: TLS 1.3 handshake (1 RTT)

1 » Client TLS hello msg

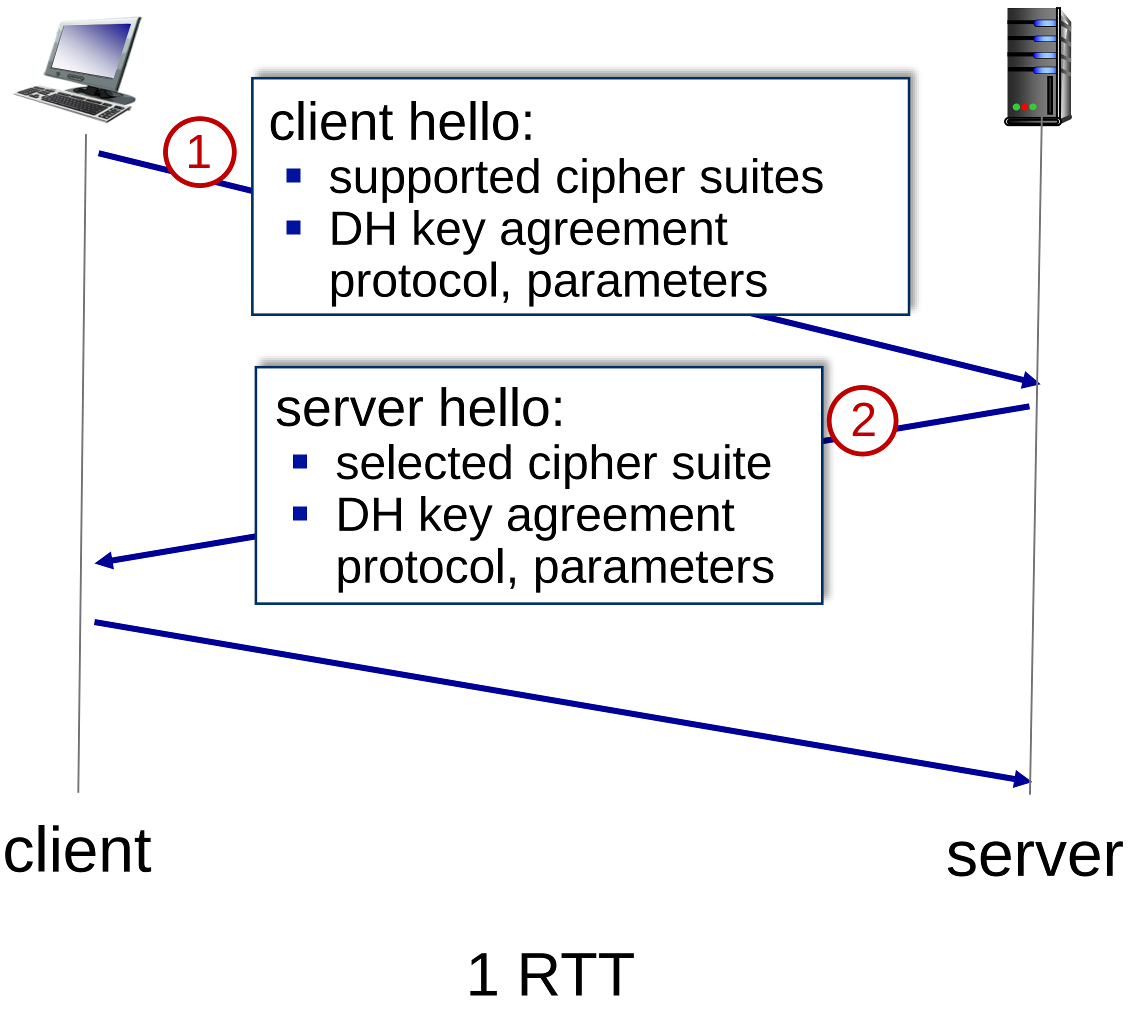
- » Key agreement protocol, parameters
- » Indicates cipher suites it supports



Recap: TLS 1.3 handshake (1 RTT)

1. Client TLS hello msg
   - Key agreement protocol, parameters
   - Indicates cipher suites it supports

2. Server TLS hello msg chooses
   - Key agreement protocol, parameters
   - Cipher suite
   - Server-signed certificate

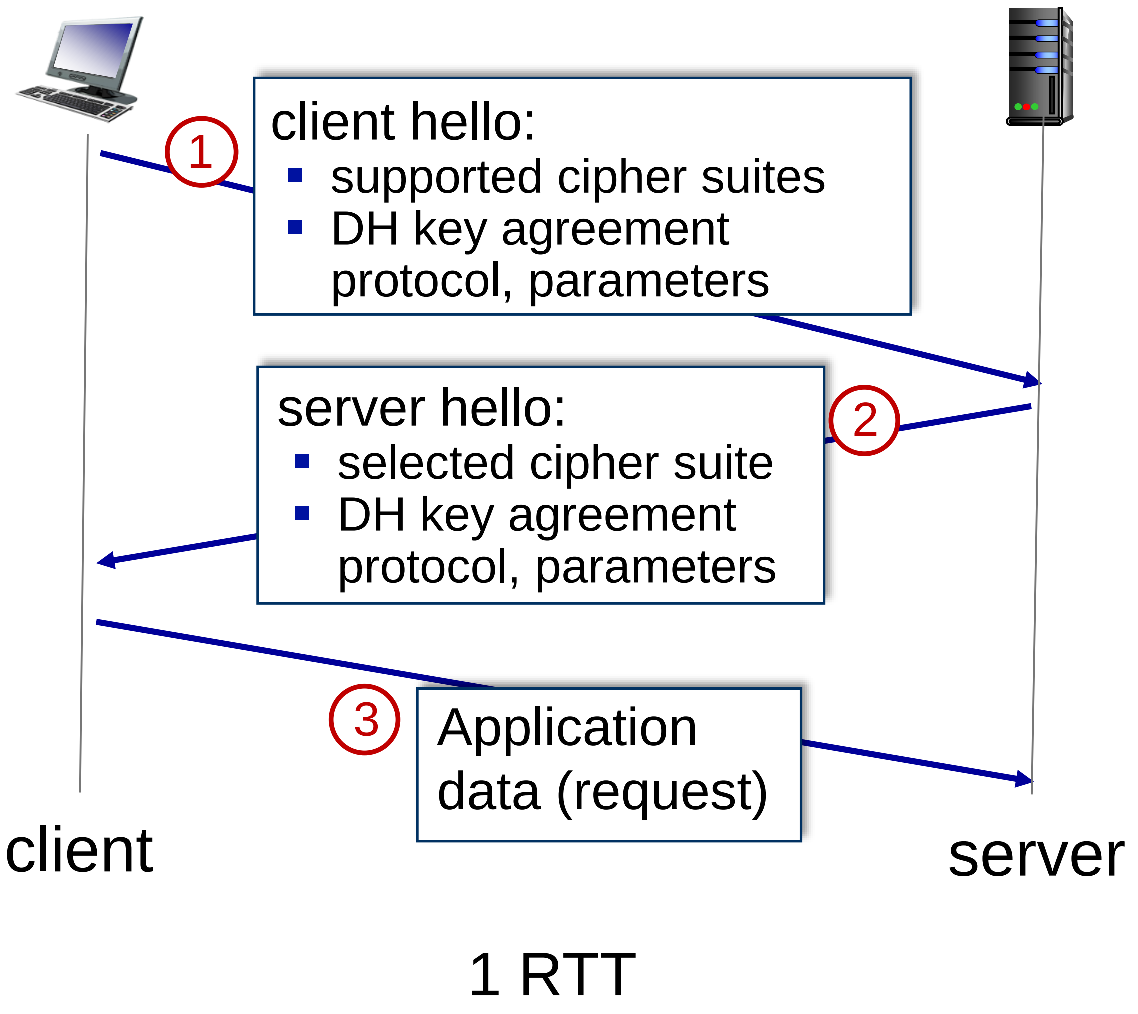


Recap: TLS 1.3 handshake (1 RTT)

1 » Client TLS hello msg
  - » Key agreement protocol, parameters
  - » Indicates cipher suites it supports

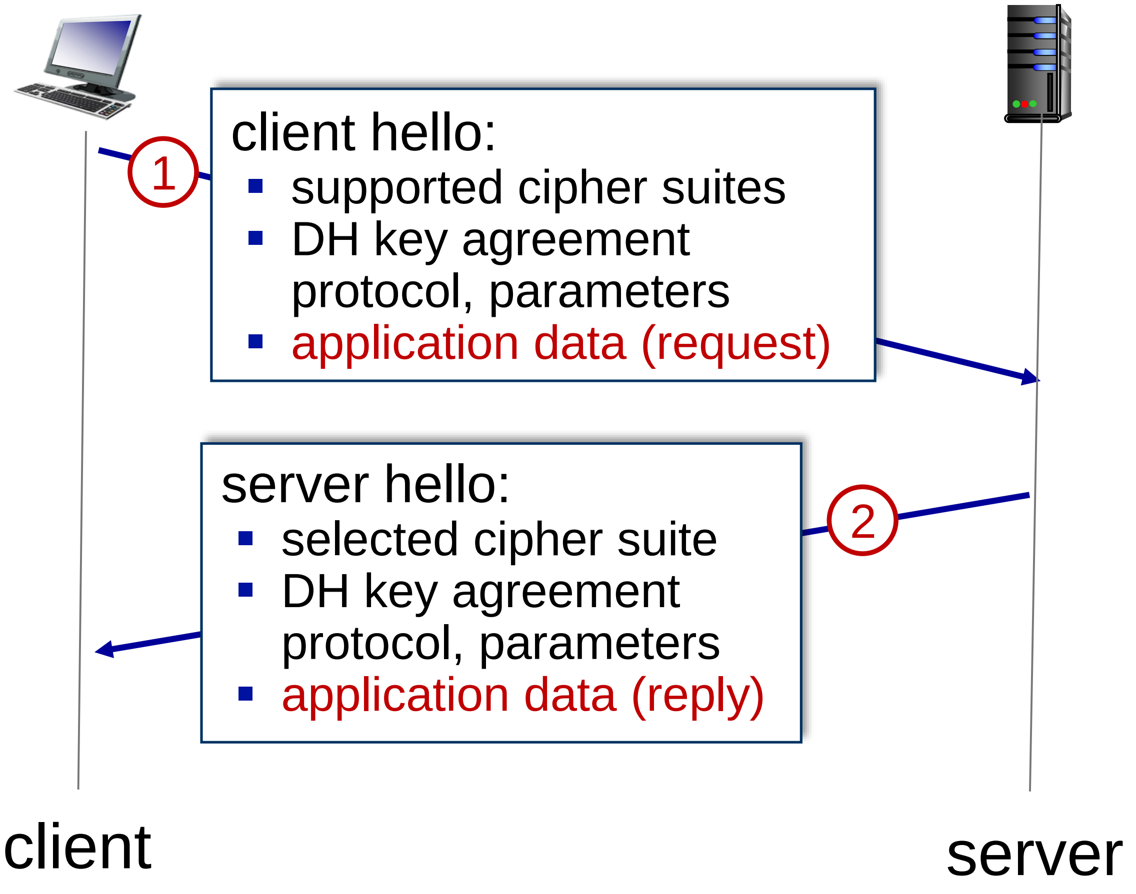
2 » Server TLS hello msg chooses
  - » Key agreement protocol, parameters
  - » Cipher suite
  - » Server-signed certificate

3 » Client
  - » checks server certificate
  - » generates key
  - » can now make application request (e.g., HTTPS GET)



Recap: TLS 1.3 handshake (0 RTT)

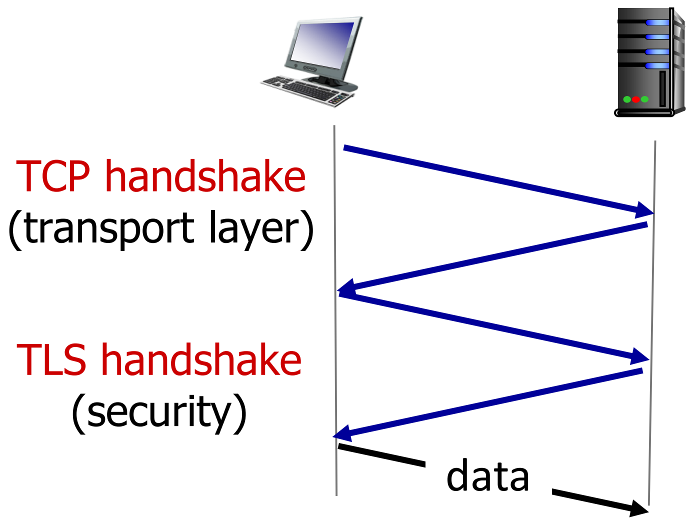
- Initial hello message contains encrypted application data
- "Resuming" earlier connection between client and server
  - Using cached data
  - Calculate initial session encryption keys before setting up new connection
  - Expired cache causes 1 RTT
- Vulnerable to replay attacks
  - Acceptable for HTTP GET or client requests not modifying server state



0 RTT

TCP + TLS: Connection establishment

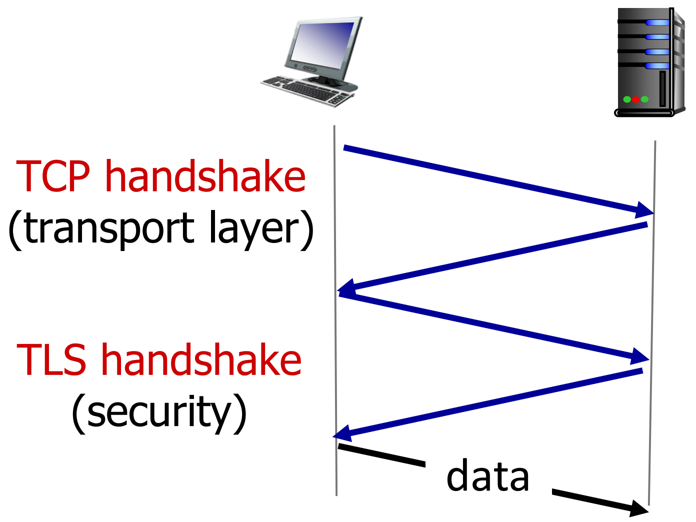
- TCP + TLS
  - 2 serial handshakes before data
  - TCP (reliability, congestion control)
  - TLS (authentication, cryptography)



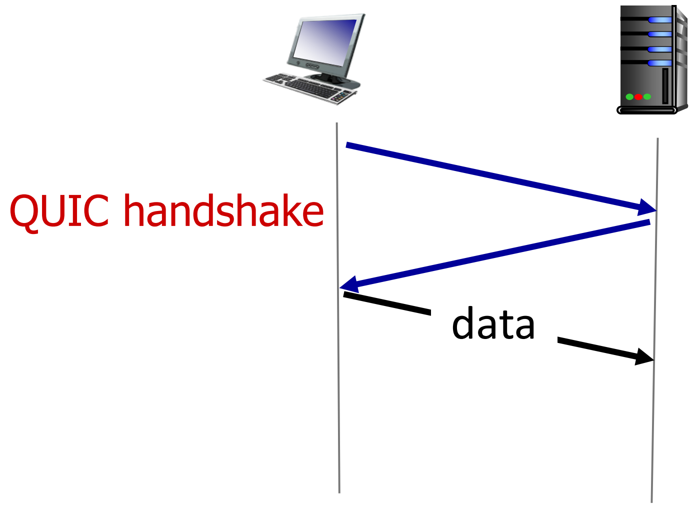
2 RTT

QUIC: Connection establishment

- TCP + TLS
  - 2 serial handshakes before data
  - TCP (reliability, congestion control)
  - TLS (authentication, cryptography)
- QUIC
  - Reliability, congestion control, authentication, cryptography
  - Combines cryptographic and transport handshakes (1 layer)
  - 1 handshake at worst



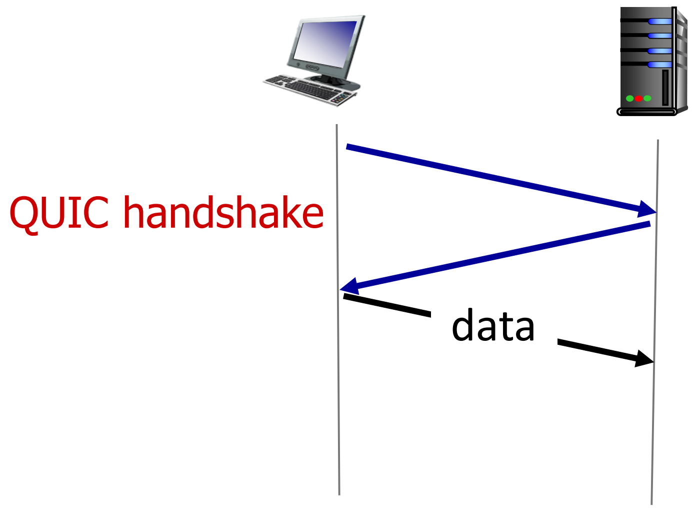
2 RTT



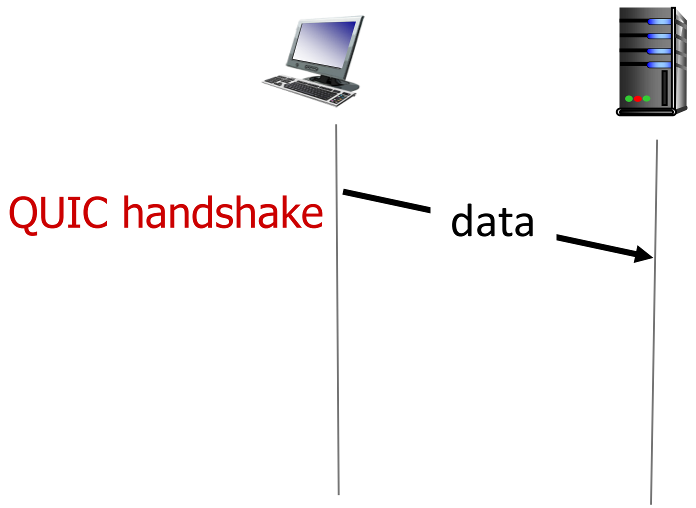
1 RTT

QUIC: Connection establishment

» Mostly 0-RTT (~88 %), sometimes 1 RTT

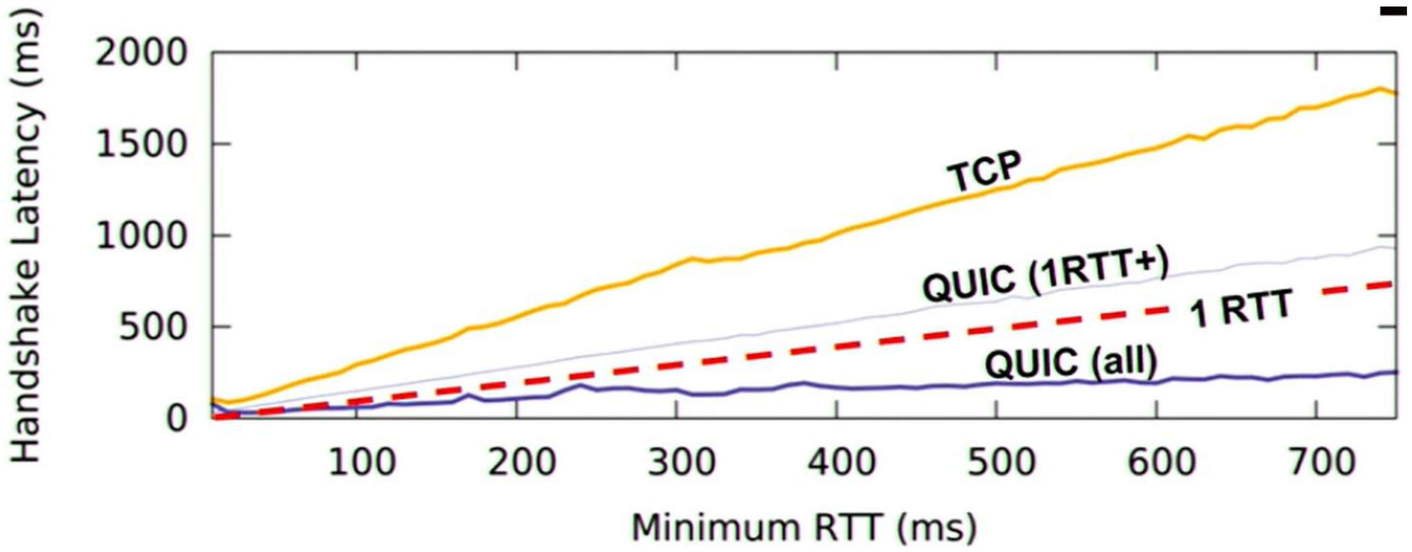


1 RTT



0 RTT

Handshake latency



QUIC features

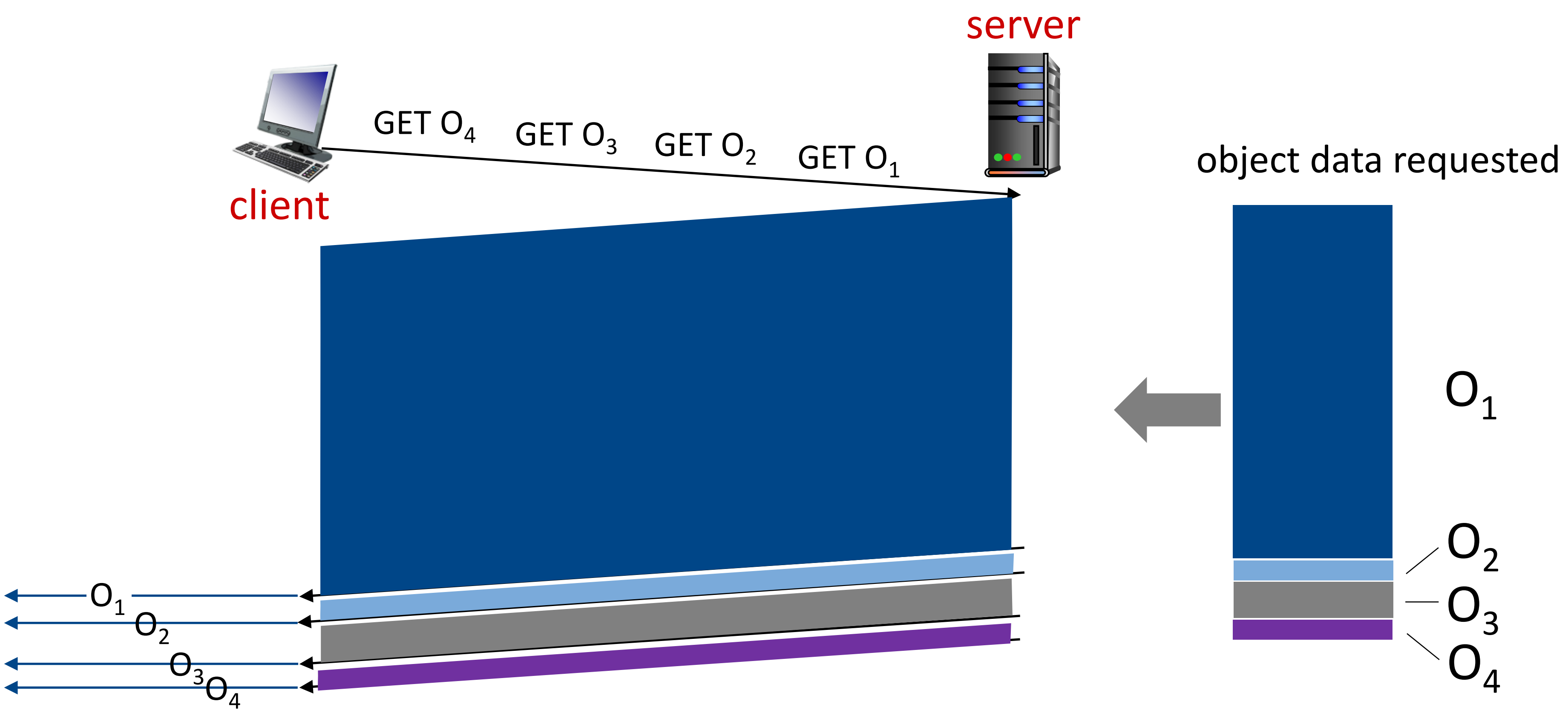
- Deployability and evolvability
- Low-latency secure connection establishment
- Streams and multiplexing
- Better loss recovery and flexible congestion control
- Resilience to NAT-rebinding

HTTP/1.1

- Introduced multiple, pipelined GETs over single TCP connection
- Server responds in-order
  - Uses FCFS (first-come-first-served) scheduling for GET requests
- FCFS results in small object may have to wait for transmission
  - Head-of-line (HOL) blocking behind large objects
- Loss recovery (retransmitting lost TCP segments) stalls object transmission

HTTP/1.1: HOL blocking

- Client requests 1 large object (e.g., video file) and 3 smaller objects
- Objects delivered in order → O_2, O_3, O_4 wait behind O_1 → HOL blocking



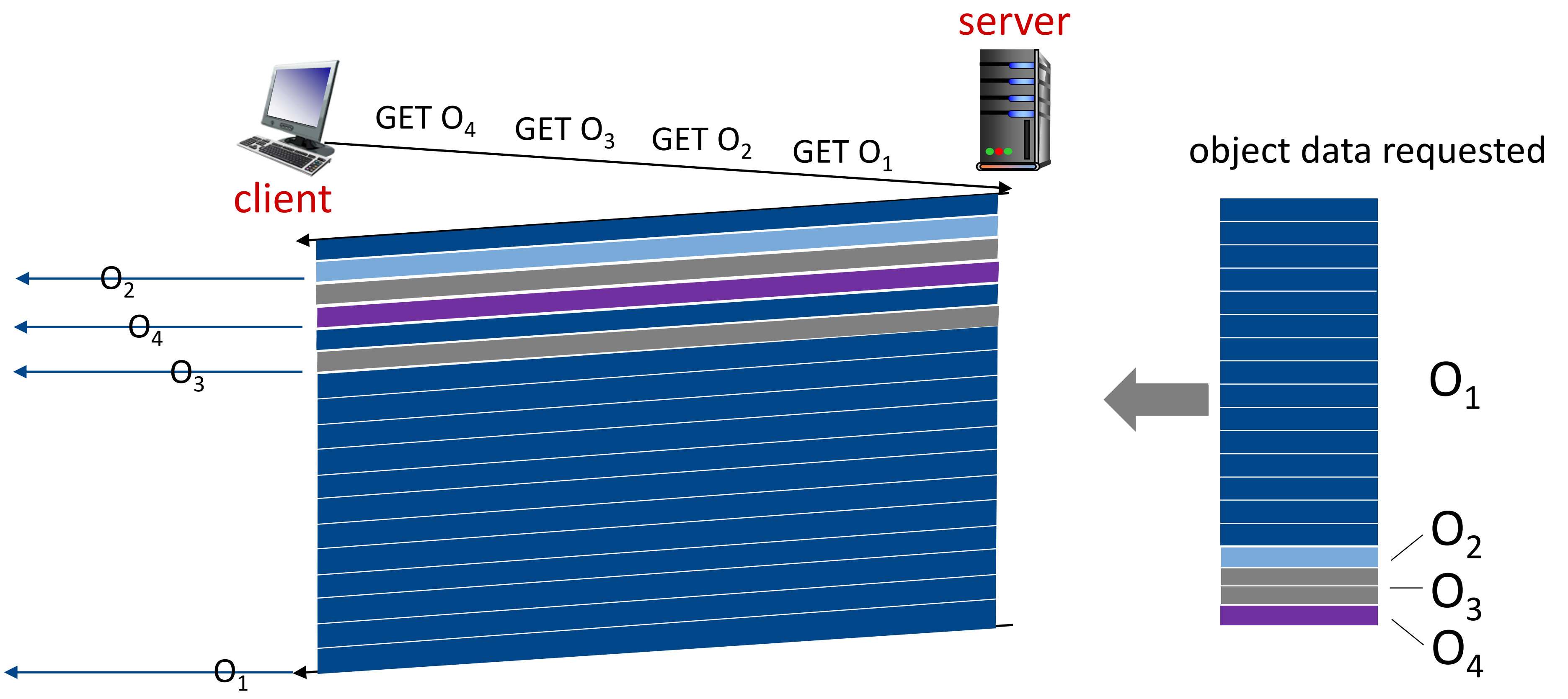
HTTP/2

- Key goal: decreased delay in multi-object HTTP requests

- Increased flexibility at server in sending objects to client
  - Methods, status codes, most header fields unchanged from HTTP 1.1
  - Transmission order of requested objects based on client-specified object priority (not necessarily FCFS)
  - Push unrequested objects to client
  - Divide objects into frames, schedule frames to mitigate HOL blocking

HTTP/2: mitigating HOL blocking

- HTTP/2: objects divided into frames, frame transmission interleaved
- O_2, O_3, O_4 delivered quickly, O_1 slightly delayed



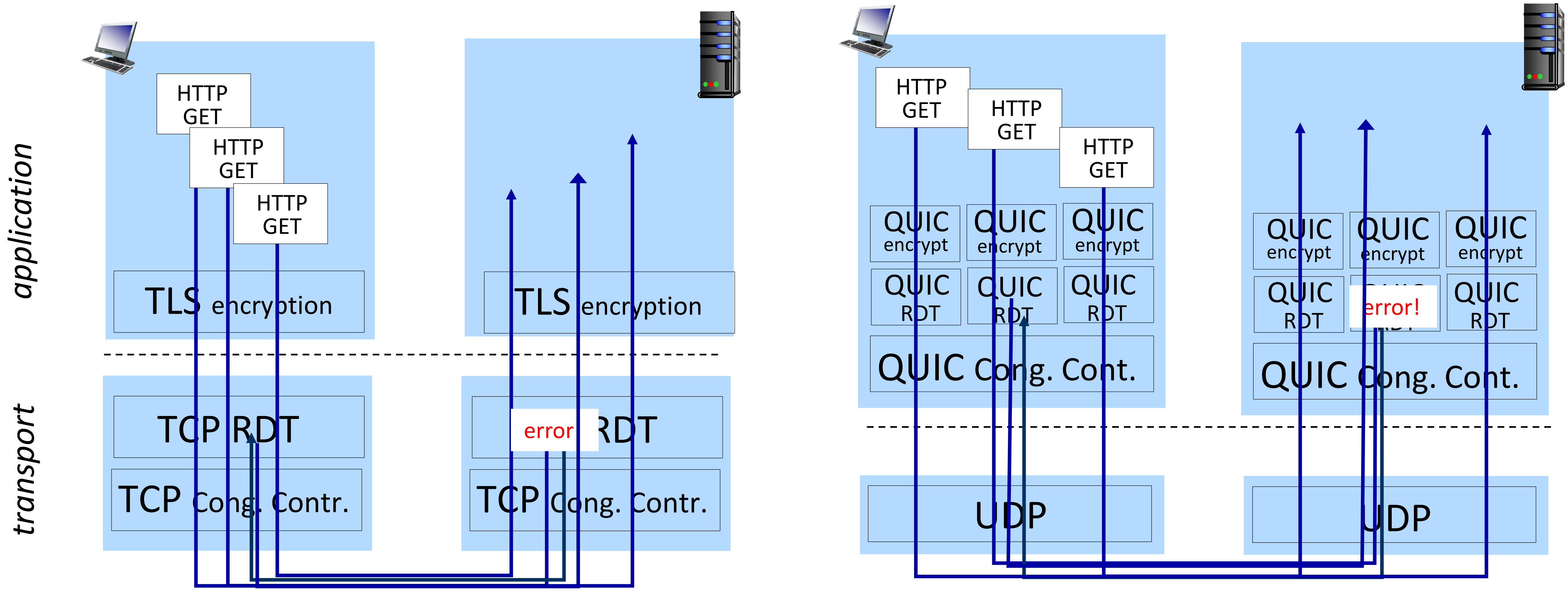
HTTP/2 to HTTP/3

- HTTP/2 over single TCP connection means:
  - recovery from packet loss still stalls all object transmissions
  - as in HTTP 1.1, browsers have incentive to open multiple parallel TCP connections to reduce stalling, increase overall throughput
- HTTP/2 provides no security over vanilla TCP connection
- HTTP/3 adds security, per object error- and congestion-control (more pipelining) over UDP

Streams and multiplexing

- Streams
  - Lightweight abstraction within a connection
  - In context of HTTP/3: different stream for each object on a web page
- Multiple application-level streams multiplexed over single QUIC connection
  - Can quickly add new streams
  - Reliable data transfer for each stream separately
  - Per stream flow control
  - Common congestion control
- Avoids head-of-line blocking in TCP

QUIC: streams: parallelism, no HOL blocking



(a) HTTP 1.1

(b) HTTP/3 (HTTP/2 with QUIC: no HOL blocking)

QUIC features

- Deployability and evolvability
- Low-latency secure connection establishment
- Streams and multiplexing
- Better loss recovery and flexible congestion control
- Resilience to NAT-rebinding

Better loss recovery and flexible congestion control

- Better loss recovery
  - Unique packet number
  - Avoid retransmission ambiguity

- Flexible congestion control
  - Receiver timestamp for better RTT estimates

- No specific congestion control
  - Draft says TCP NewReno, but mostly CUBIC used
  - As fair as TCP

QUIC features

- Deployability and evolvability
- Low-latency secure connection establishment
- Streams and multiplexing
- Better loss recovery and flexible congestion control
- Resilience to NAT-rebinding

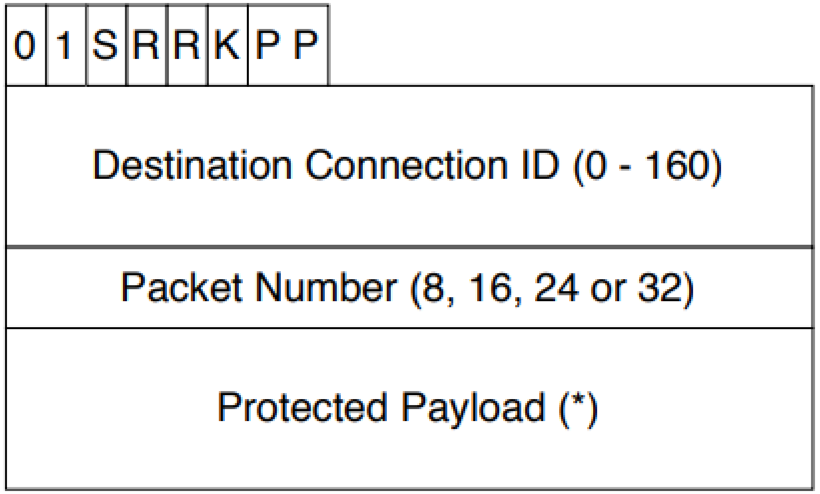
Resilience to NAT-rebinding

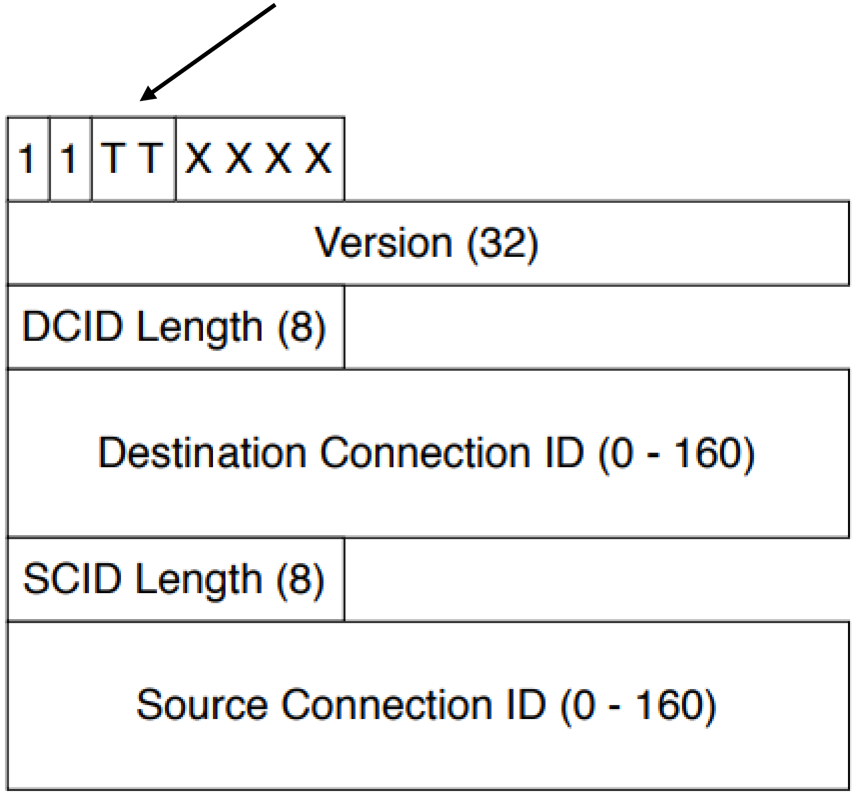
- 64-bit to 160-bit connection ID
- Instead of IP + port pairs
- Survives NAT timeout and NAT rebinding
  - More aggressive for UDP than TCP
- Improves migration, handovers to new IP
- Improves multipath

QUIC packet structure

- Long header packets
  - Initial connection establishment
- Short header packets
  - Transmit data

Bit values	Packet type
00	Initial
01	0-RTT
10	Handshake
11	Retry





Research on QUIC

- Langley et al., The QUIC Transport Protocol: Design and Internet-Scale Deployment, Proc. ACM SIGCOMM, 2017
  - 600 citations

- ACM SIGCOMM Workshop on Evolution, Performance, and Interoperability of QUIC (EPIQ)

- Zhilong Zheng et al., Xlink: Qoe-driven multi-path quic transport in large-scale video services, Proc. ACM SIGCOMM, 2021.

- Johannes Zirngibl, et al., It's over 9000: analyzing early QUIC deployments with the standardization on the horizon, Proc IMC, 2021.

EPIQ

- 2018
  - *Moving fast at scale: Experience deploying IETF QUIC at Facebook*
  - Real-time Audio-Visual Media Transport over QUIC
  - The QUIC Fix for Optimal Video Streaming
  - Towards QUIC debuggability
  - Observing the Evolution of QUIC Implementations
  - Interoperability-Guided Testing of QUIC Implementations using Symbolic Execution
  - nQUIC: Noise-Based QUIC Packet Protection
  - A Stream-Aware Multipath QUIC Scheduler for Heterogeneous Paths

EPIQ

- 2019
  - 0 papers accepted out of 15 submitted

EPIQ

- 2020
  - *As QUIC as TCP, Optimizing QUIC and HTTP/3 CPU usage*
  - Testing QUIC with packetdrill
  - Automating QUIC Interoperability Testing
  - Same Standards, Different Decisions: A Study of QUIC and HTTP/3 Implementation Diversity
  - Making QUIC Quicker With NIC Offload
  - Scalable High Efficiency Video Coding based HTTP Adaptive Streaming over QUIC
  - Analyzing the Adoption of QUIC From a Mobile Development Perspective

EPIQ

- 2021
  - *QUIC usage at Microsoft*
  - *QUIC usage at Apple*
  - Verifying QUIC implementations using Ivy
  - Days of Future Past: An Optimization-based Adaptive Bitrate Algorithm over HTTP/3
  - Tracking the QUIC Spin Bit on Tofino
  - The Search of the Path MTU with QUIC
  - Evaluation of QUIC-based MASQUE Proxying
  - Congestion Control for Real-time Media over QUIC

References

- [1] Langley et al., The QUIC Transport Protocol: Design and Internet-Scale Deployment, Proc. ACM SIGCOMM, 2017.
- [2] David Hasselquist et al., QUIC Throughput and Fairness over Dual Connectivity, Proc. IEEE MASCOTS Workshop, 2020.
- [3] Ian Swett. As QUIC as TCP, Optimizing QUIC and HTTP/3 CPU usage, Proc. EPIQ keynote, 2020.
- [4] James Kurose and Keith Ross, Computer networks: A top down approach featuring the internet, 2021.