BAWMAN

HTTP 1, 2, 3 …

Baw Chng

info@bawman.com

IEEE

First presented at an IEEE event on 2022 November 15

Find this presentation at bawman.com/PRESENTATIONS/

Biography

BAW CHNG @ BAWMAN LLC

- Proud Senior Member of IEEE. Founder and President of BAWMAN LLC
- Consult in the Wireless Networking and Telecommunications industry
- **Services offered**:
  - **Strategic Consulting** – business strategy, product positioning, technology roadmap development, RFI/RFP
  - **Standards Consulting** – standards development, standards & trade group representation
  - **Technology Consulting** – network system architecture, technology development, evaluation and trial
  - **Patent, Intellectual Property** – patent evaluation, pre-trial consultation, expert witness services
- **Clientele**: communications service providers, technology suppliers, equipment vendors, start-ups, established enterprises, investors, intellectual property firms, the legal community
- **Expertise**: networking systems, wireless communications, standards and protocols
  - cellular, Wi-Fi, IoT, LPWAN, optical, NFV, SDN, cloud, IEEE, IETF, 3GPP, WBA, CableLabs, Broadband Forum
- **Contact**: www.bawman.com ; info@bawman.com

Outline – “HTTP 1, 2, 3 …”

- Biography
- Background
  - Reference model, UDP and TCP basics
  - HTTP as the basis for “Web API”
- HTTP 0.9, HTTP/1.0, HTTP/1.1
- HTTP/2
- HTTP/3
  - QUIC
- Summary
- Discussion, Q&A …

https://...

BAWMAN

Background

Reference Model, TCP and UDP Basics

HTTP as the basis for "Web API"

Reference Model

7. Application	**HTTP** *is generally considered here.* *Performance is highly influenced by the Transport layer.*
6. Presentation	Character encoding, compression, encryption/decryption, e.g., S/MIME
5. Session	Manage continuous exchange of info, multiple back-and-forth between two nodes
4. Transport	E.g., TCP, UDP
3. Network	E.g., IP (Internet Protocol)
2. Data Link	E.g., Wi-Fi, Ethernet, cellular air interface
1. Physical	E.g., Wi-Fi, Ethernet, cellular air interface

Reference: OSI model, ISO/IEC 7498-1, ITU-T Recommendation X.200

UDP and TCP Basics

UDP

- Uses very little resource
  - Scalability impact
- Connectionless
  - Send data right away
- No guarantee that packets will arrive in order, or that packets will arrive at all
- No congestion/flow control

- **Indirectly used by HTTP/3**

TCP

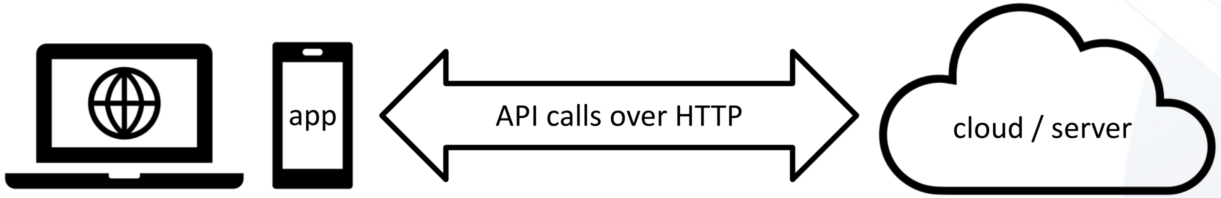
- Resource intensive
  - Scalability impact
- Connection-oriented
  - Establish connection before sending data
- Guarantees* no packet loss, guarantees* that packets arrive in order
- Congestion/flow control

- **Used by HTTP up to HTTP/2**

HTTP as the basis for "Web API"

- (Pre-1990s) "Remote Procedure Call" (RPC)
- (c.1990s) CORBA, Java remote method invocation, **CGI**
- (c.2000 and after) **SOAP/XML, REST/JSON over HTTP**

➔ "Web API" called not only by web browsers, but also by mobile apps



- Examples of popular web applications/services that have published Web APIs:
  - Google, PayPal, Facebook, Twitter, Amazon Web Service, …

HTTP 0.9, HTTP/1.0, HTTP/1.1

From humble beginning to world domination

HTTP 0.9

- "The Original HTTP as defined in 1991" by Tim BL.

- Very simple, very limited
  - Use normal Internet-style telnet protocol on TCP/IP link
    - defaults to port 80 if not otherwise specified
  - Request: "**GET** */address/to/some/web/page.html*"
    - Requests are idempotent (will revisit the concept of idempotence later)
  - Response: a message in HTML as a byte stream of ASCII characters
    - Message is terminated by the closing of the connection by the server
    - Error response are supplied in human readable text in HTML syntax

- Also very forward-looking
  - Mandates backward compatibility into the future
  - Puts the definition of HTTP in the public domain

Reference: https://www.w3.org/Protocols/HTTP/AsImplemented.html

Sidebar

Protocol Stack View (Original HTTP)

HTTP

TCP

IP

HTTP/1.0

- Protocol versioning (e.g., “GET /some/page.html HTTP/1.0”)
- Multimedia support (reference MIME, “Content-Encoding”, “Content-Type”)
- Additional character sets support (reference MIME)
- Message types, message structure (header vs. body)
- Syntax for query (e.g., “GET /search?q=why+is+the+sky+blue”)

- Additional methods (other than “GET”)
  – more in later slides

- Status Codes (e.g., “404” Not Found)
  – more in later slides

- Authentication & Authorization (Access Control)
  – more in later slides

Reference: RFC 1945 (Informational) ; MIME = Multipurpose Internet Mail Extension

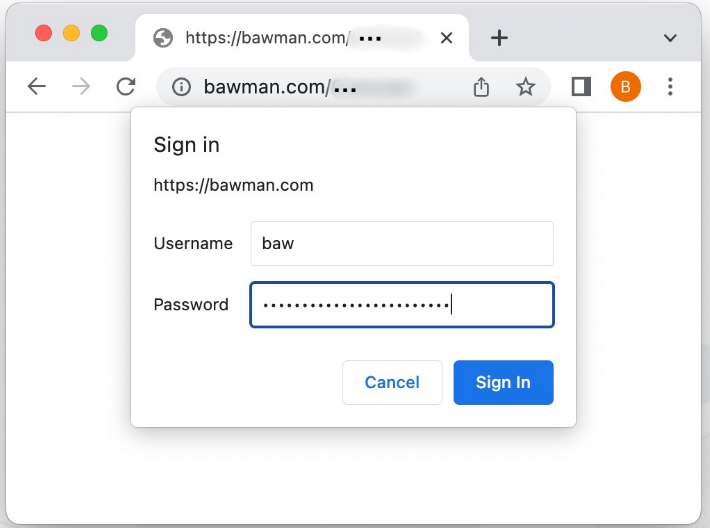
HTTP/1.0 Methods

- **GET**
  - Add support for “conditional GET” (help manage caching)
- **HEAD**
  - Same as the old “GET” except server returns only the “header” part of the HTTP Response (the “body” part is omitted)
- **POST**
  - Allows data to be sent to the server, enables such functions as
    - Filling out a form over the web
    - Posting a message to an electronic bulletin board

HTTP/1.0 Status Codes

- **Informational 1xx**
  - None defined in HTTP/1.0
- **Success 2xx**
  - E.g., “200” OK
- **Redirection 3xx**
  - Redirect to new address if resource has moved
  - Exception “304” Not Modified
    - Response to “conditional GET”
- **Client Error 4xx**
  - E.g., “401” Unauthorized
  - E.g., “403” Forbidden
  - E.g., “404” Not Found
- **Server Error 5xx**
  - E.g., “501” Not Implemented
  - E.g., “502” Bad Gateway
  - E.g., “503” Service Unavailable

HTTP/1.0 Authentication & Authorization



- A way to restrict access to certain resources
  - Authenticate the user before allowing access

- Basic "challenge-response" mechanism

- Typical flow ("basic" authentication) :
  1. Client (browser) requests access to certain restricted resource
  2. Server (seeing no authentication info), returns "401" Unauthorized
  3. Client prompts the user for *username/password*
  4. User enters *username/password*
  5. Client encodes *username*:*password* using base64
  6. Client requests access again with base64-encoded *username*:*password* in the header

➔ **This is not secure, there is no encryption**

What Has Not Changed from HTTP 0.9 to HTTP/1.0

- **Use TCP as transport, one TCP connection per HTTP Request/Response**
  - Client/browser opens a TCP connection to send a HTTP Request to server
  - Server closes the TCP connection at the end of sending HTTP Response
  - Clients/browsers resort to opening multiple TCP connections to the same server for parallelism

Imagine a "photo album" website with 50 thumbnails on one webpage. Will need ≥ 51 TCP connections!

➔ Not efficient, scalability is an issue

- **No encryption**
  - Good enough to disseminate information that is not secret
    - E.g., you can put your product catalogue on the web and even make it "searchable"
  - **Not good enough to enable e-commerce transactions**
    - E.g., prospective customers would not buy things on the web if they are worried that their private information (e.g., credit card numbers) may be easily exposed

HTTP/1.1

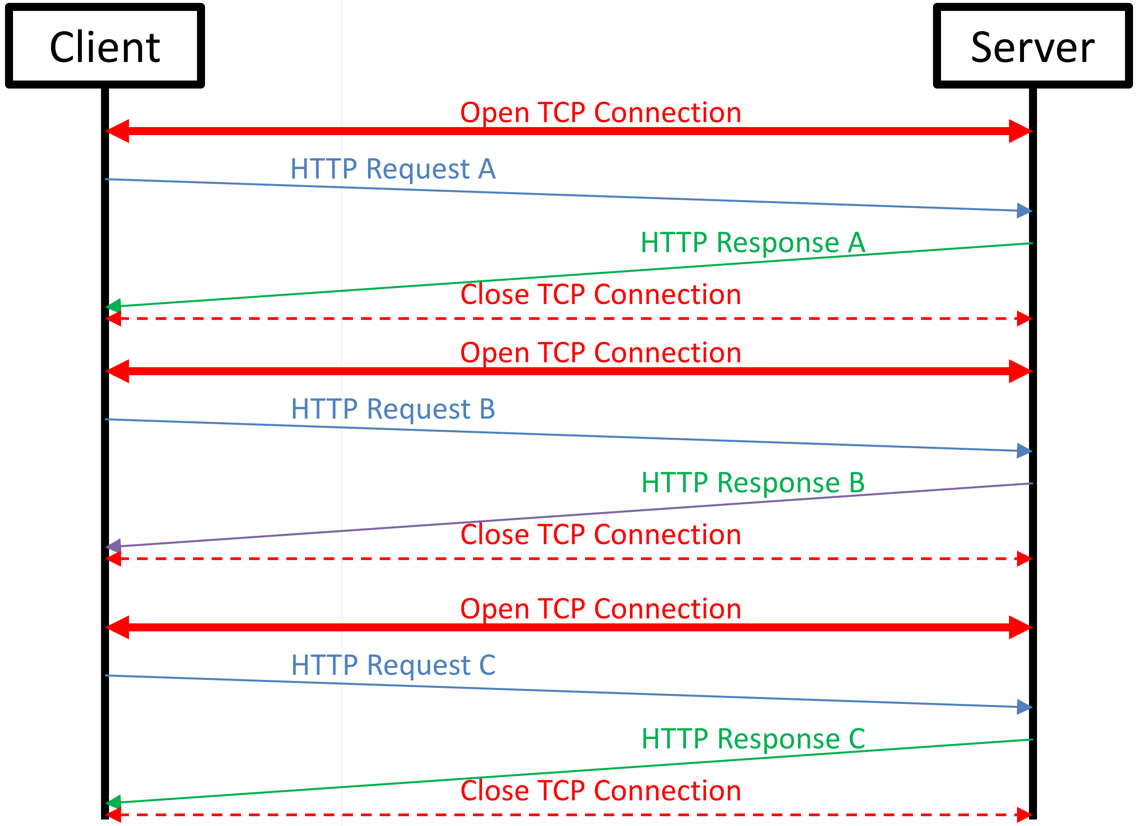
- **More Methods**
  - GET, HEAD, POST, **OPTIONS, PUT, DELETE, TRACE, CONNECT**
- **More Status Codes**
  - Too many to list; still follows the 1xx, 2xx, 3xx, 4xx, 5xx scheme
- More thorough treatment on
  - Method “safety” and “idempotence”
  - “Caching”
- **Efficiency/scalability improvement**
  - more in later slides
- **Encryption support** (“https”)
  - more in later slides

References: RFCs 2068, 2616, 2817 (Standards Track)

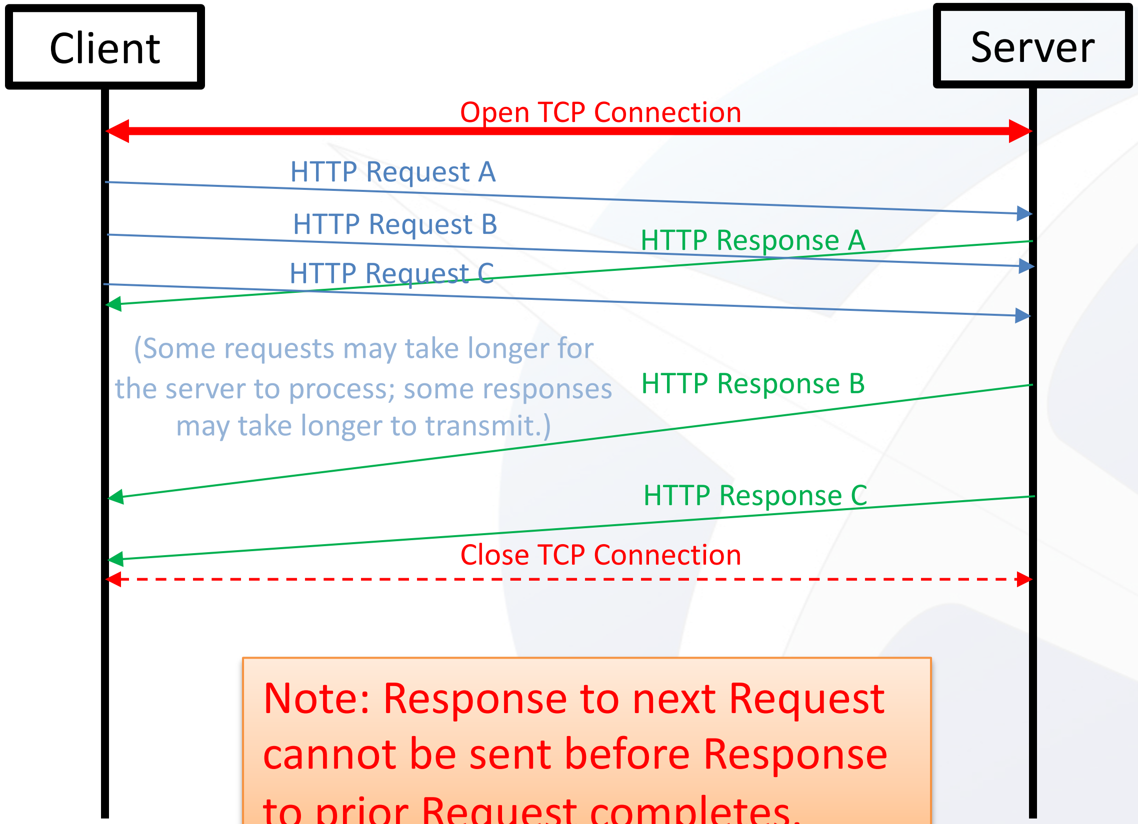
HTTP/1.1 “Persistent Connection” and “HTTP Pipelining”

- Can transact multiple HTTP Requests/Responses over one TCP connection
- Can send subsequent requests without waiting for responses for previous requests
- Server must send Responses to Requests in the same order that the Requests were received

HTTP without Pipelining



HTTP/1.1 with Pipelining



Note: Response to next Request cannot be sent before Response to prior Request completes.

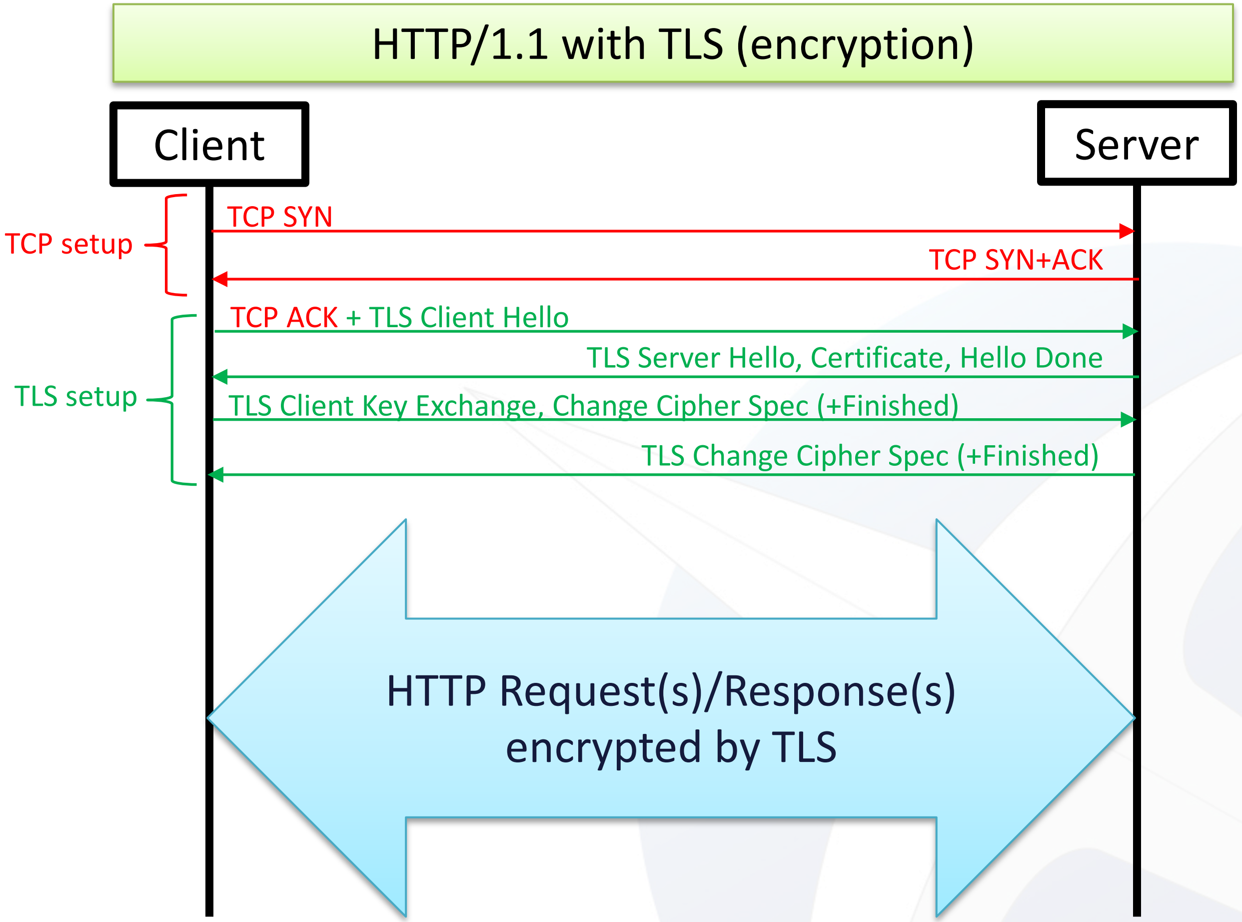
HTTP Pipelining, Method Safety & Idempotence

- “Safe” method ➔ “read-only”
    - E.g., **GET**, **HEAD**
- “Idempotent” method:
    - Achieve same result if the same method is requested multiple times in identical way as if it is requested only once
        - “Safe” methods are inherently “idempotent”
        - E.g., **PUT**, **DELETE** (*x* ← 5, remove member *a* from set ***A***)
- Non-idempotent methods
    - Each request of the same method in identical way leads to a change
        - E.g., **POST** (*x++*, negation, append *str1* to *str2*, post message *x* to a bulletin board)

➔ SHOULD NOT pipeline requests using non-idempotent methods or non-idempotent sequence of methods

HTTP/1.1 Encryption; going from http to https

- Use TLS over TCP
- Enables e-commerce!



Note: It takes three round trips (3 RTTs) to get to sending encrypted HTTP messages (if using TLS ≤ 1.2; can be reduced to 2 RTTs if using TLS 1.3)

Reference: RFC 2817 (Standards Track)

Sidebar

Protocol Stack View (HTTP/1.1 with TLS)

HTTP

TLS (Optional)

TCP

IP

HTTP/2

Latency reduction (faster page load time) & efficiency improvements

HTTP/2 : What's New?

- HTTP full multiplexing within a TCP connection
- Request prioritization
  - the two above are interrelated; more in later slides

- HTTP header compression
  - Introduces HPACK compression format (Huffman coding)
  - Headers get bigger as websites become more sophisticated
  - Compress header to reduce overhead and increase efficiency

- Server Push
  - Server can supply resources not explicitly requested
    - As part of Response to a Request
      - In addition to the resource explicitly requested
      - (When the server expects the client will need those resources)

  ➔ Reduce page load time

References: RFCs 7540, 9113 (Standards Track)

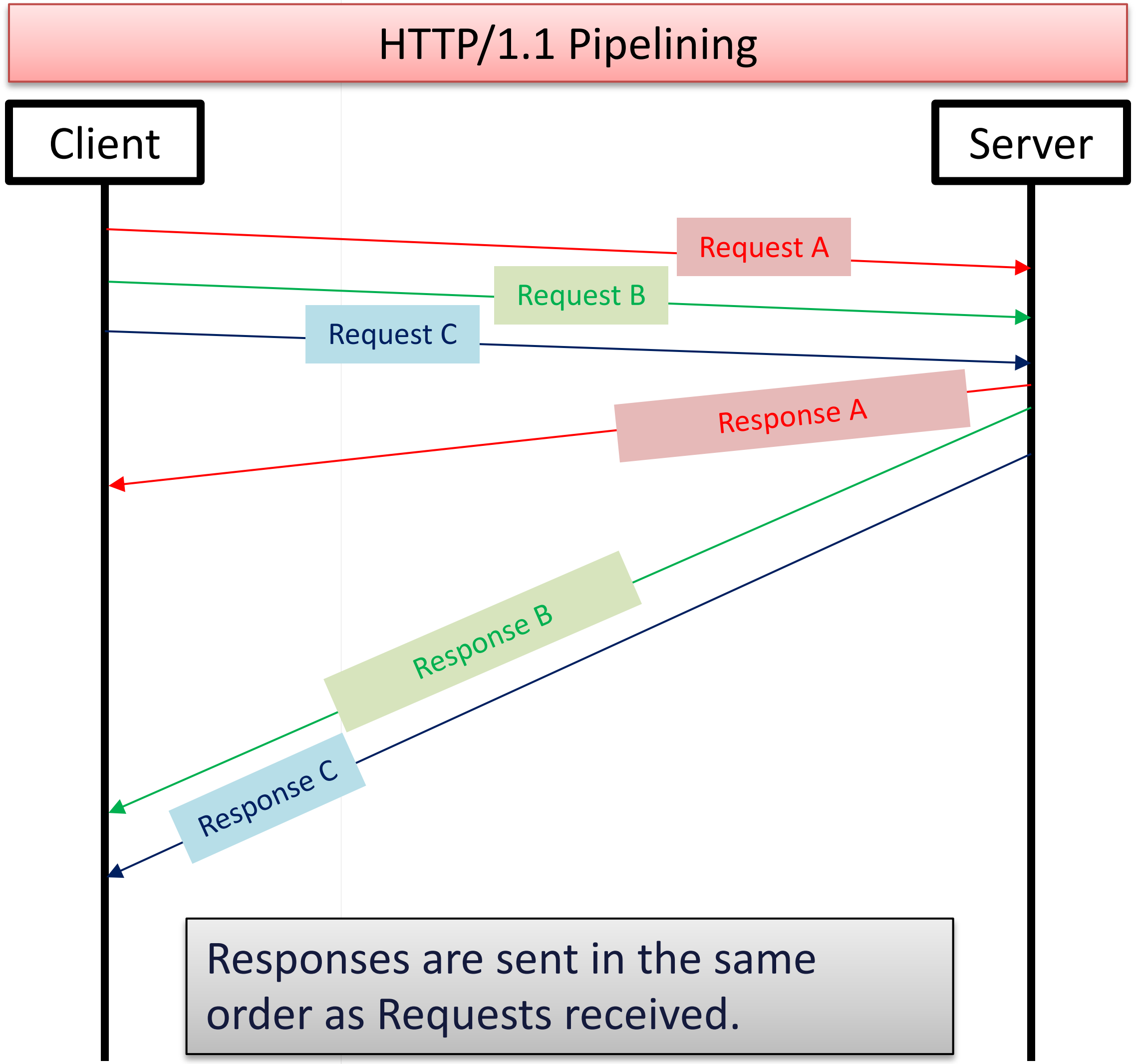
HTTP/2 Multiplexing & Prioritization

- Multiple HTTP Requests/Responses to share a TCP connection
- Allow prioritization of some Requests over others

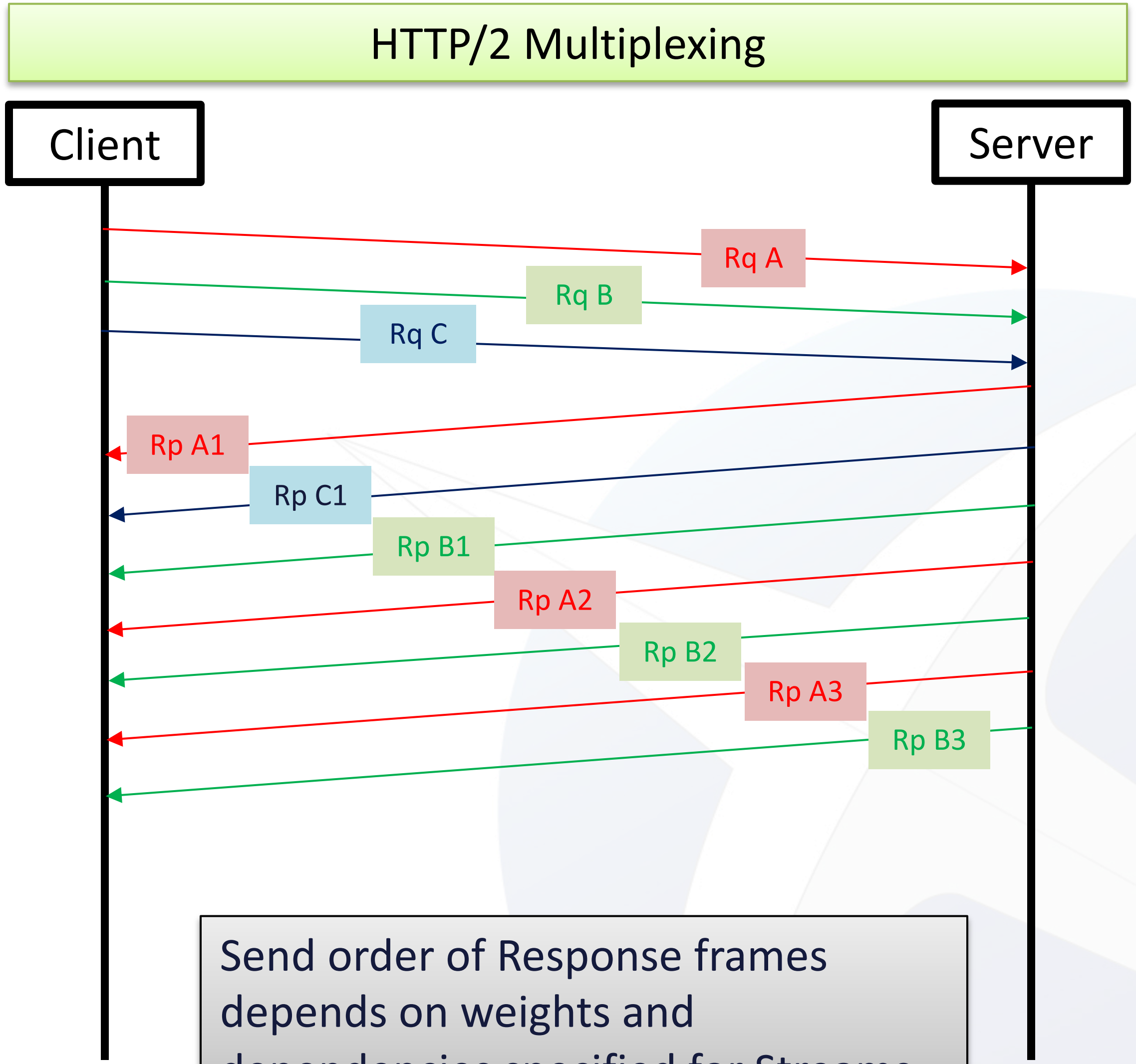
- **HTTP/2's solution**:
  - Assign different HTTP Request/Response transactions to different "Streams"
  - Split HTTP Message into "Frames"
  - Send "Frames" into "Streams"
    - "Frames" for the same HTTP transaction goes into the same "Stream"
  - Can assign different "weights" to different Streams
  - Can specify dependencies among streams
    - Dependencies and weights determine which Requests

➔ Urgent Request need not be "blocked" by prior Requests that may take a long time to complete

HTTP/2 Multiplexing & Prioritization (Example)



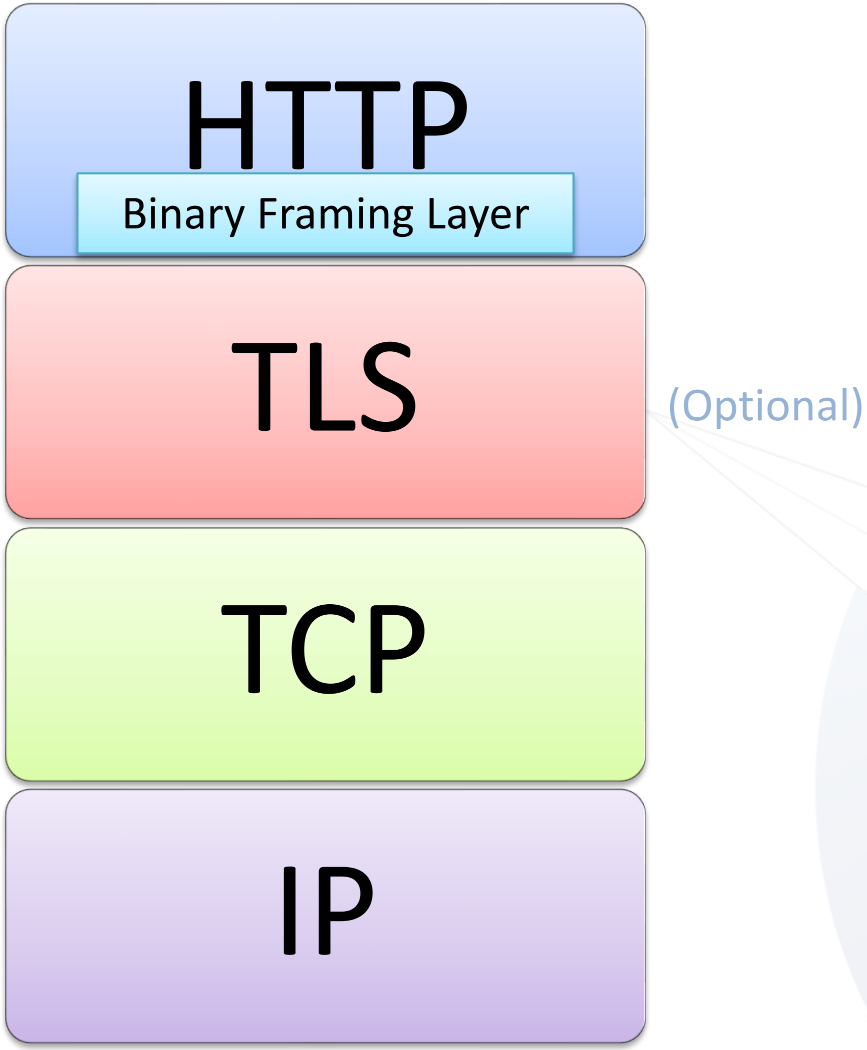
Responses are sent in the same order as Requests received.



Send order of Response frames depends on weights and dependencies specified for Streams.

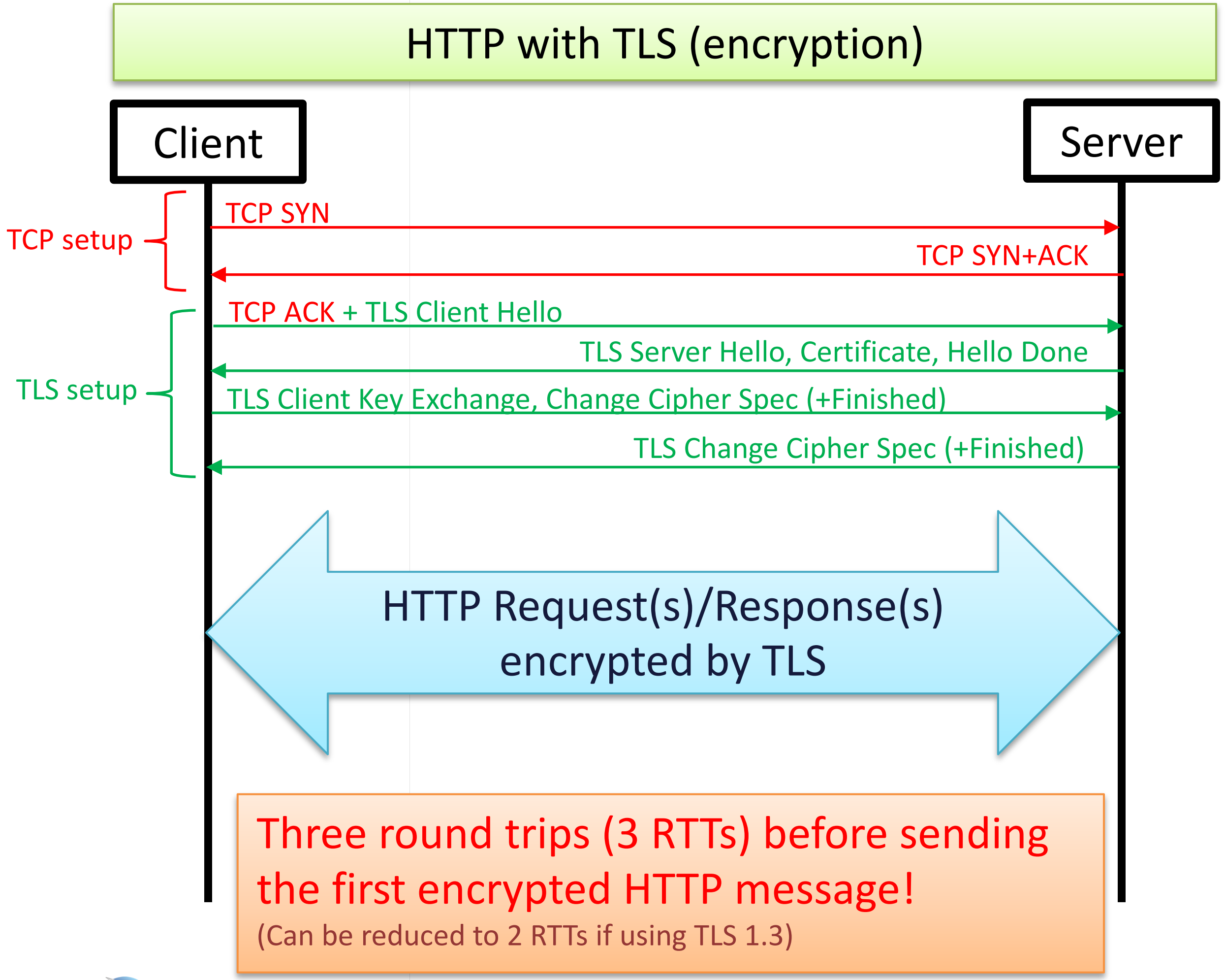
Protocol Stack View (HTTP/2 with TLS)

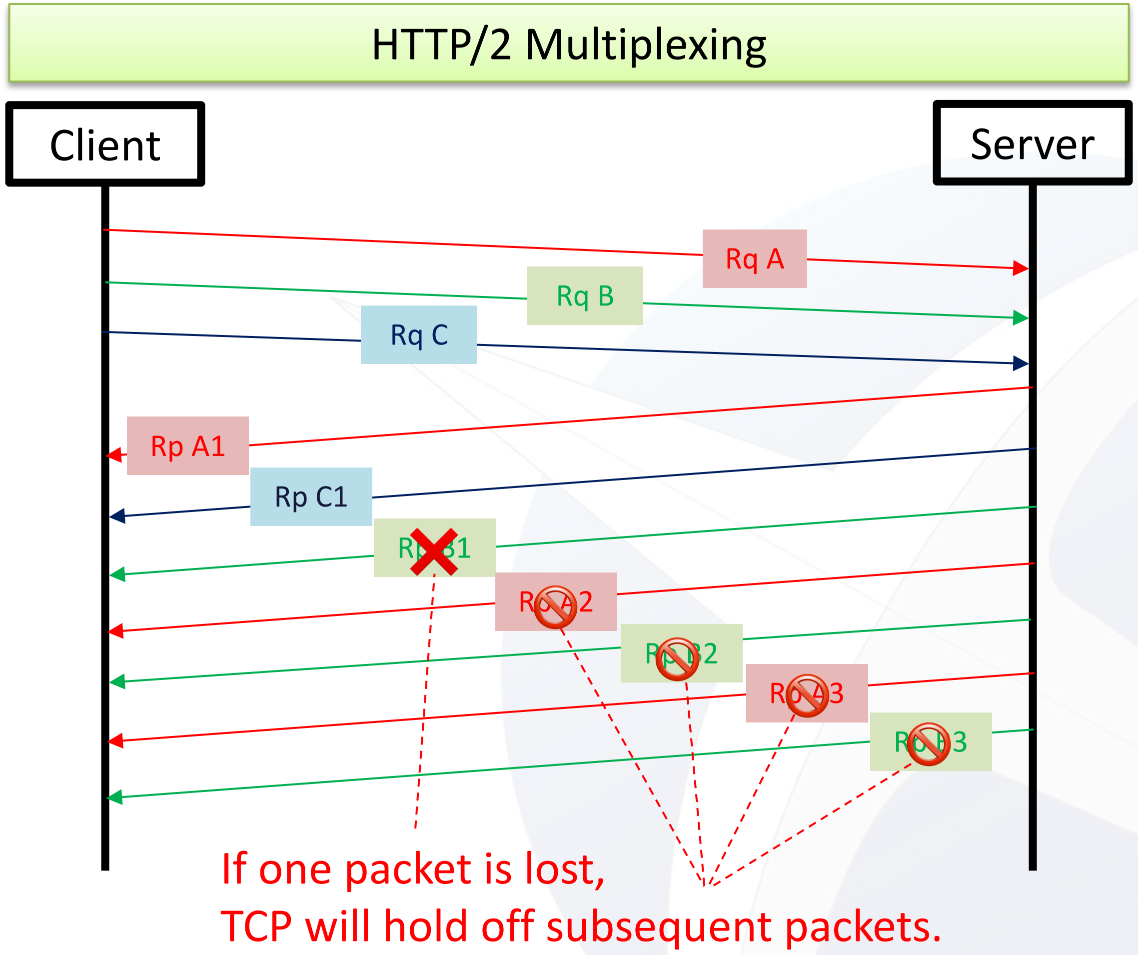
Sidebar



What Problems Are Left Unsolved by HTTP/2 ?

- Transport layer TLS/TCP problems





HTTP/3

Time for a new Transport – QUIC over UDP

HTTP/3 : What's New ?

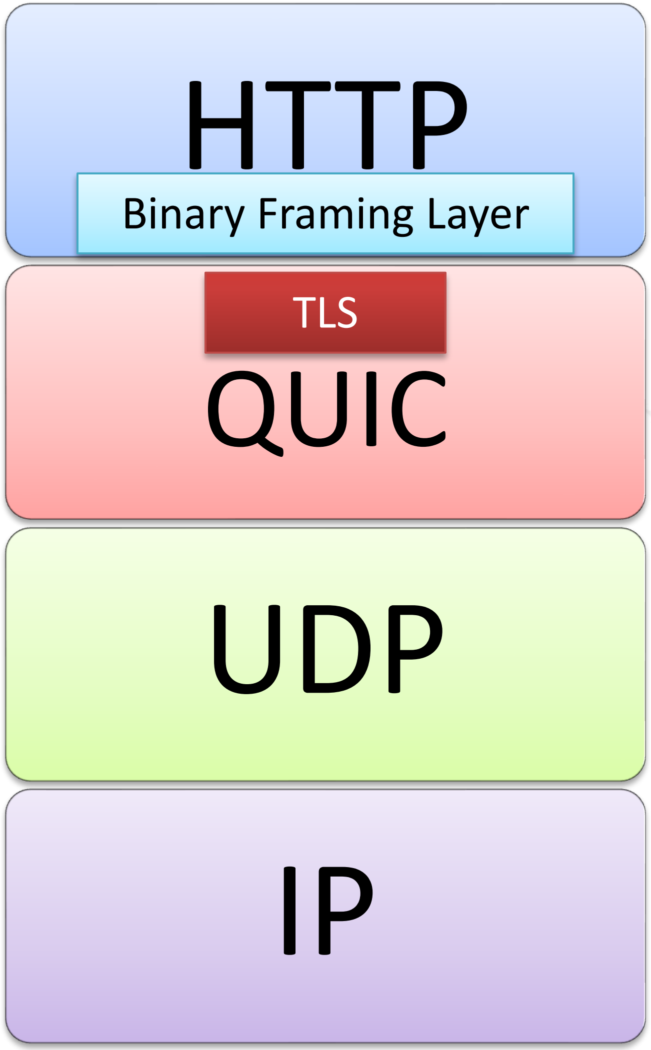
- Recall the two issues left unsolved by HTTP/2
  - Slow start latency: need three round trips (3 RTTs) to set up TCP and TLS before sending the first HTTP message
  - Head-of-line blocking: TCP will hold off subsequent packets if a prior packet is lost

- HTTP/3's answer:
  - Use a different transport protocol that is not TCP
  - HTTP/3's chosen transport protocol is QUIC over UDP

References: RFCs 9114, 9000, 9001 (Standards Track)

Sidebar

Protocol Stack View (HTTP/3 with QUIC over UDP)



BAWMAN

QUIC Basics

- QUIC natively supports multiple “Streams” per connection
  – Reliability and ordered delivery at the Stream level
  – Streams do not block one another
    - Loss of a packet for one Stream will not affect delivery of other Streams

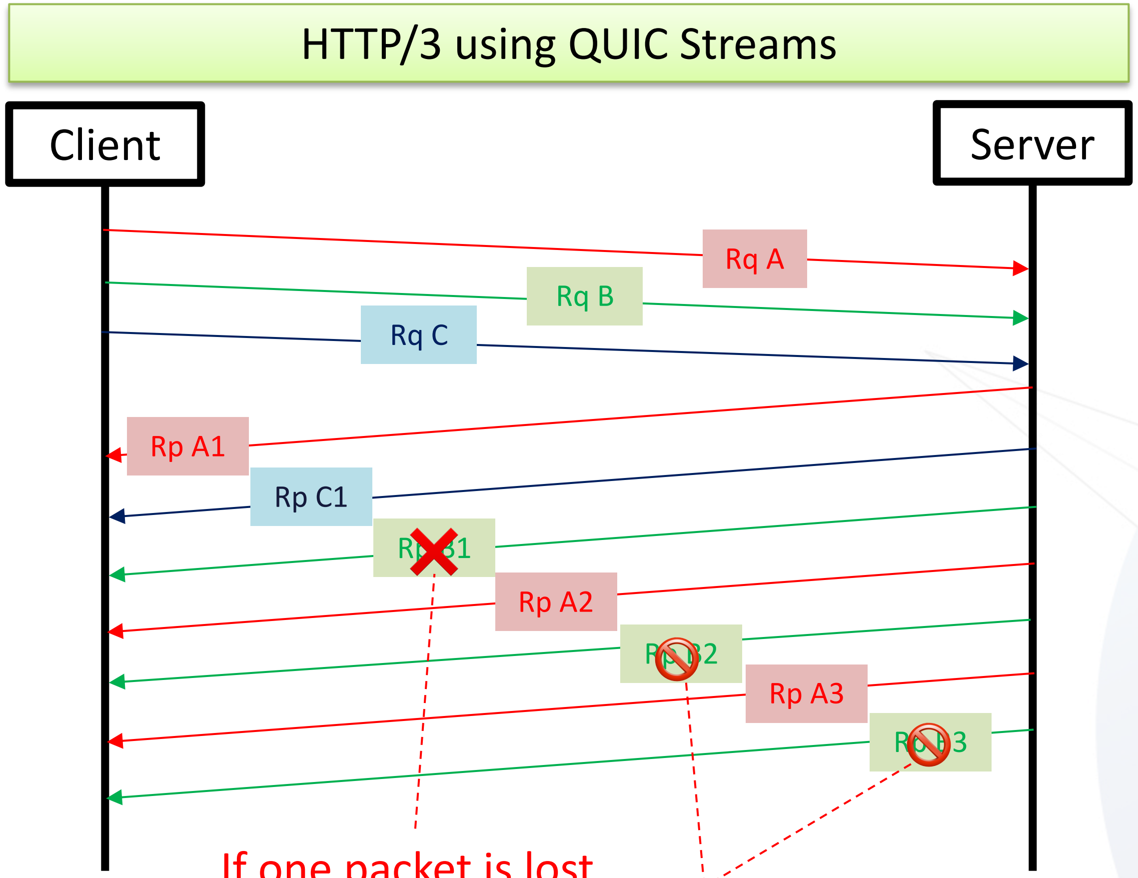
  → Avoid the TCP head-of-line blocking issue

- QUIC uses TLS and integrates TLS handshake
  – Sets up TLS the same time a QUIC connection is setup

  → One round trip (1-RTT) and you can start sending HTTP Messages

  → There is also a 0-RTT option

QUIC Streams Do Not Block One Another

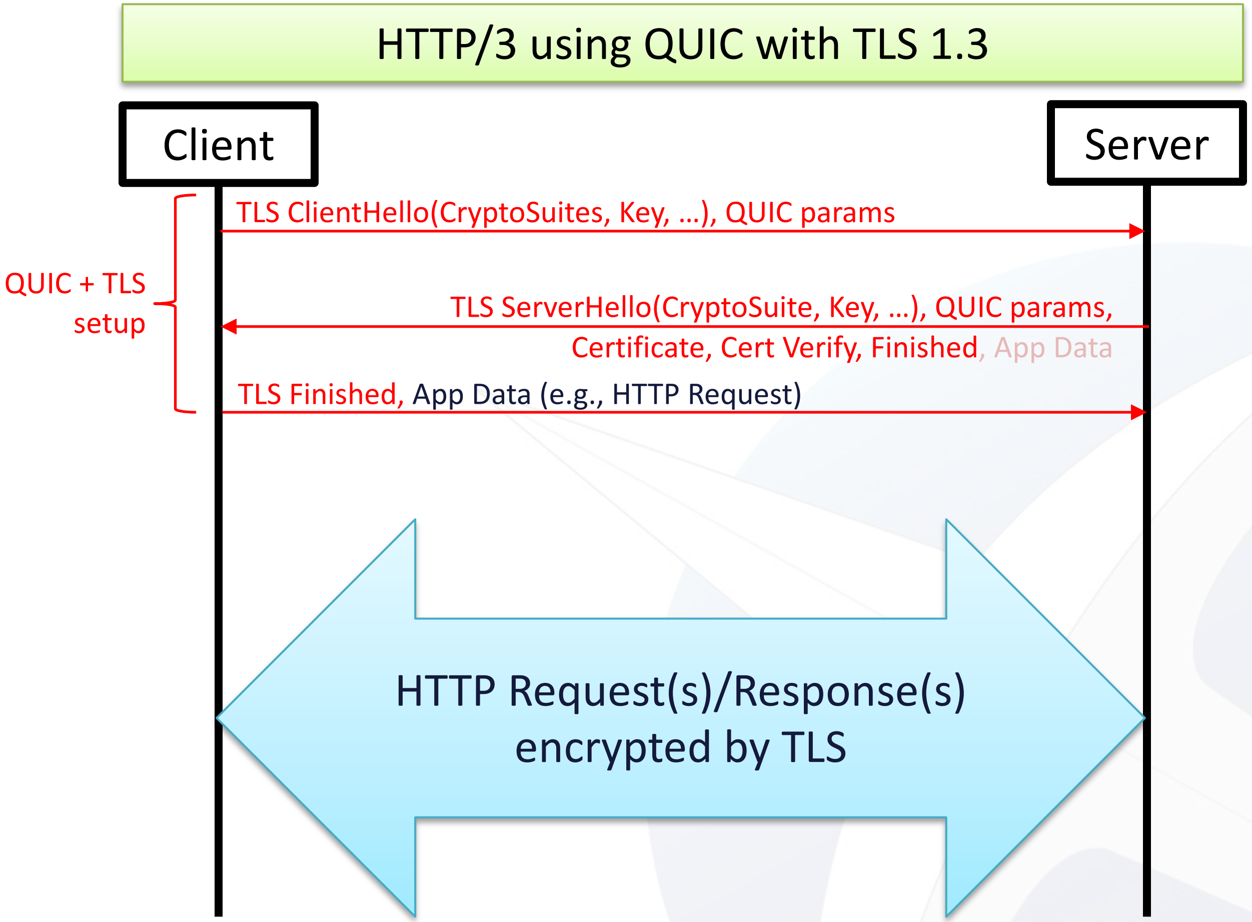


If one packet is lost,
QUIC will only hold off subsequent packets for the same stream.
Packets for other streams are not blocked.

QUIC Setup with TLS

- QUIC combines its own handshake with TLS handshake

➔ Client can send HTTP Request after 1-RTT



Sidebar

QUIC's 0-RTT Operation

- QUIC & TLS 1.3 allows client to send encrypted application data by *reusing negotiated parameters from a previous connection*
  - **Benefit**: even faster page load time when browser revisits a website to which it has previously connected
  - **Risk**: reusing previously used security parameters subject the 0-RTT packet to "replay attack"
  - **How to protect yourself from this risk?**
    - Play it safe: entertain 0-RTT only for idempotent methods

HTTP 1, 2, 3 … Summary

HTTP 0.9	• Humble beginning • "GET" only, HTML in ASCII text only
HTTP/1.0	• Multimedia, Multilingual, Authentication/Authorization • User data upload (query & "POST"), Status Codes
HTTP/1.1	• Encryption support enables e-commerce, Web API • Efficiency improvements (HTTP Pipelining)
HTTP/2	• Latency and efficiency improvements at the Application layer • HTTP Multiplexing, Server Push, Header Compression, Prioritization
HTTP/3	• More latency and efficiency improvements at the Transport layer, replace TCP with QUIC over UDP

HTTP 1, 2, 3 …

Thank you!

Questions and Answers, Discussion …