

## 6 (Hyper Text Transfer Protocol - HTTP 1.0, 1.1, 2, 3)

It is a client/server communication protocol

Most of the times, client is our browser, and web server is our server. It is a (Request/Response) Architecture

HTTP Request {

URL

Method Type

Headers

Body

HTTP Response {

Status Code

Headers

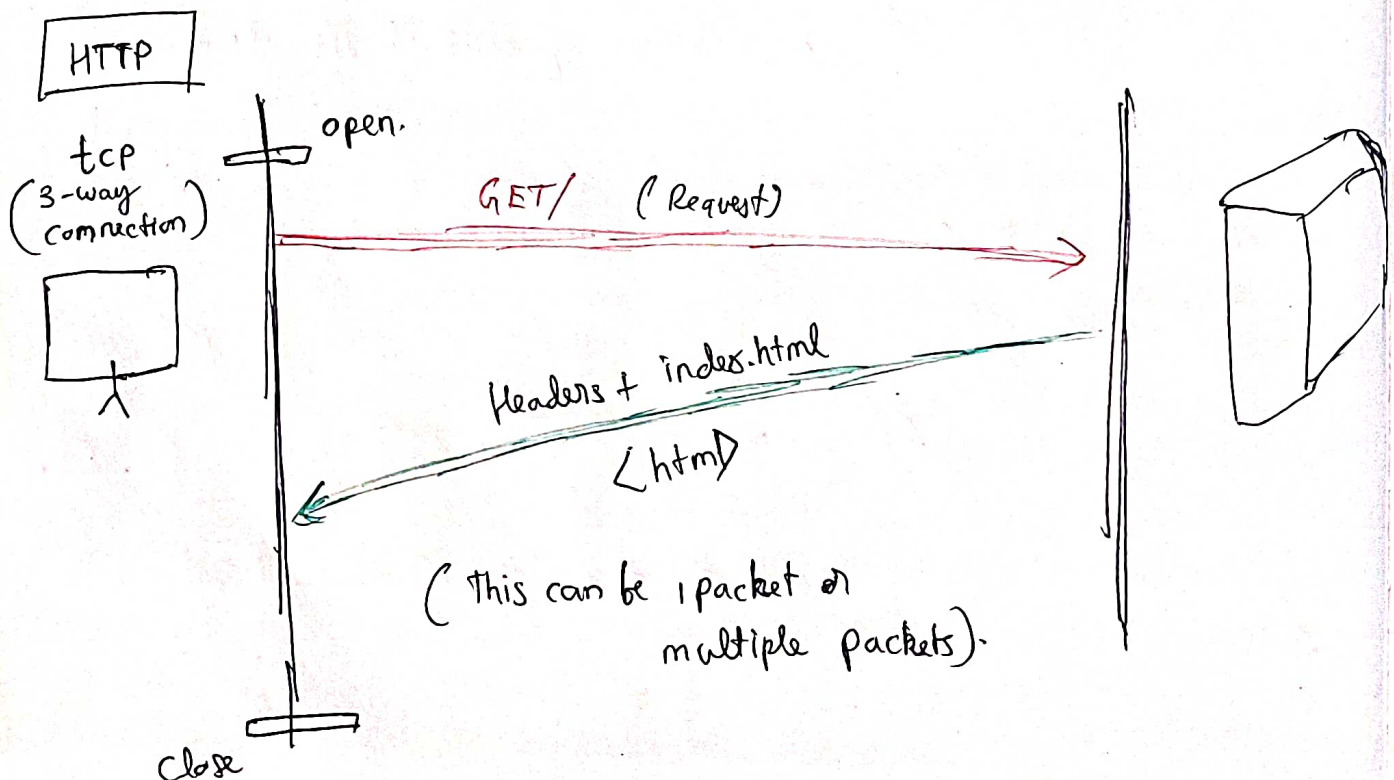
Body

}

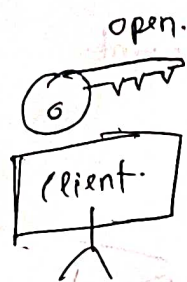
}

TCP is the vehicle that transports the blob of data packets.

to the across

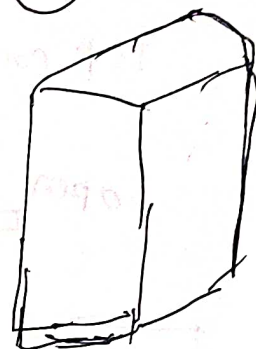


# HTTPS



Exactly like HTTP, but  
 Some thing happens at the  
 beginning. TLS handshake  
 happens. (Handshake)  
 Reason [ Have the Same Key ]

(For encryption)  
 (Same Key)



GET/

(Headers + index.html)   
 <html>

close

# HTTP 1.0

(Deprecated).

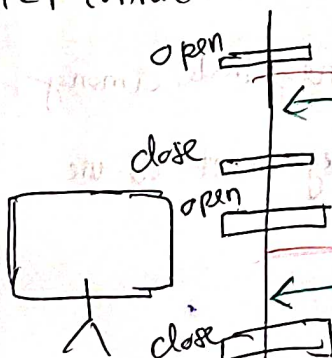
[ Buffering ]

will be explained.

Since TCP is a stateful protocol, Hence each TCP request  
 needs to maintain and reserve some memory in the RAM,

~~Hence~~ Earlier RAM was very less,

So for each Request HTTP 1.0, would open a TCP connection and close a  
 TCP connection.



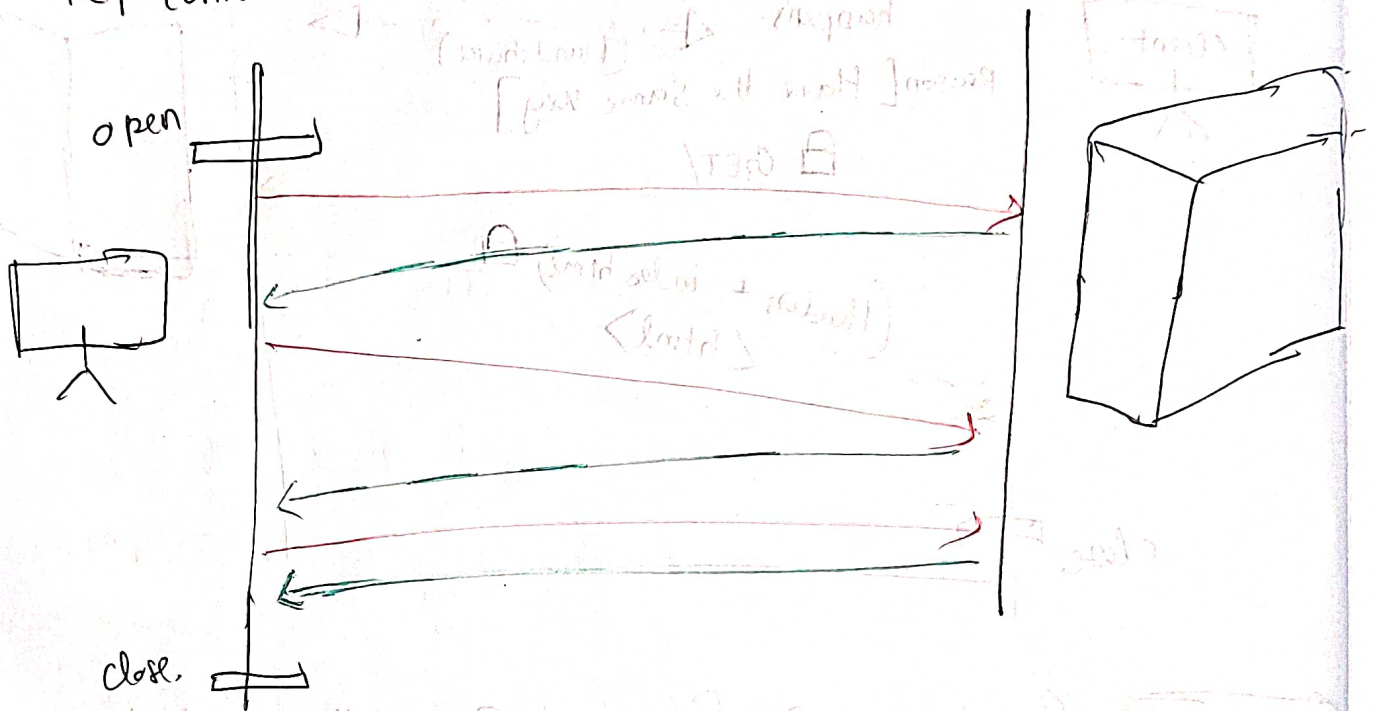
(Since tcp is super slow, So  
 HTTP 1.0 was super slow.)



## (HTTP 1.1) [Streaming with Chunked Transfer]

Keep Alive header: Passed with the request.

TCP connection is kept alive. till the client/server want



\* persisted tcp connection

\* Low Latency

## (HTTP/2)

(will be discussed) → (will be explained)

Introduced Multiplexing, Compression, Protocol Negotiation during TLS.

Protocol Negotiation: Basically client and server negotiate among themselves ~~whether they want~~ which protocol they want to use for security → TLS → (Transport Layer Security).

Also deciding on the transfer protocol

## (HTTP/3)

- \* Replaces TCP with QUIC [UDP with Congestion Control]
- \* All HTTP/2 features.

(~~\*)~~ Difference between Buffering  $\rightarrow$  [HTTP 1.1] & Streaming with Chunked Data  $\rightarrow$  [HTTP 2.0]

Example.

### Buffering:

Buffering is the process where data is accumulated into a buffer until a certain threshold is reached, after which it is sent in a single transmission.

Example: Buffering a File Download.

### (1 Application Layer)

- \* Client requests a large file from a server.
- \* Server reads the file into a buffer until it reaches a predetermined size (eg. 1MB)

Then fixed size buffer after going through (L4, L3, L2) is sent.



This buffering you will also see while watching a Youtube video.

By Buffering it fills some part of the bar, then loads again.

## Streaming with Chunked Transfer Encoding

Chunked Transfer Encoding is used to send data in a series of chunks, particularly useful for streaming data or when total length of content is unknown.

1. Application Layer: [Streaming Video]

- \* Client requests a video stream from a server.
- \* Server starts reading and encoding the video into small chunks (4KB).

| <u>Buffering</u>   | <u>Chunked Transfer</u>   |
|--|---|
| Transmission: Data accumulated and only sent once message is ready or buffer is full | Data is divided into chunks and sent as soon as chunk is ready. |
| Latency: Higher latency [Slower]   | Lower Latency [Faster]  |
| Use Case: Ensure data integrity:<br>File Transfer, Youtube                           | Video Streaming   |

Lets Explain to you how HTTP/2 uses multiplexing

and wins over HTTP/1.1.

HTTP/1.1 with Keep Alive Header:

- \* Allows the same tcp connection to be reused for multiple HTTP request and responses
- \* It reduces the overhead of establishing a new tcp request for each Request.

Pipelining :

- \* HTTP/1.1 introduced this concept, which allows multiple requests to be sent out waiting for corresponding responses.  
(Multiple requests can be sent)

- \* However, the responses must be received in the order the requests were sent - [Head-of-line blocking]

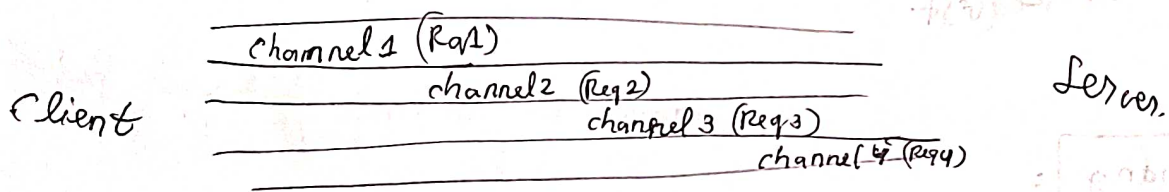
→ { \* If earlier request takes long time to process, all request following that will wait.

\* this lack of concurrency slows down  
}



## HTTP/2 Multiplexing

- \* HTTP/2 allows multiple streams within a single TCP connection, enabling multiple requests and responses to be sent and received independently and concurrently.
- \* Each stream/channel is identified by its own stream/channel id.



\*\*\*

### No Head-of-Line Blocking

- \* Since streams are independent, a delay in one stream (request) does not block others.
- \* Better Performance.

## Multiplexing Concept Innovation

HTTP/2 was designed to address the limitation of HTTP/1.1 particularly to handle the concurrency issue.

## (Key Concepts)

### (1) Streams

A stream is an independent, bidirectional sequence of frames exchanged between the client and server.

### (2) Messages and Frames

→ It corresponds to logical HTTP request and response.

Frames → Are packets. (Smallest Unit).

\*\*\*

Each HTTP request/response pair is associated with a new stream.

\*\*\*

Streams/Channels can be assigned higher priorities, for more important messages to be delivered faster.

## (Historical Context & Evolution)

Before HTTP/2, SPDY (a protocol developed by Google) was an experimental protocol that introduced many of the concepts including multiplexing.

SPDY introduced concept of streams and frame-based comms.  
Demonstrated the performance benefits of multiplexing

Later adopted by (HTTP/2)