HTTP

ELEC-C7420 Basic Principles in Networking

Aalto-yliopisto
Sähkötekniikan
korkeakoulu

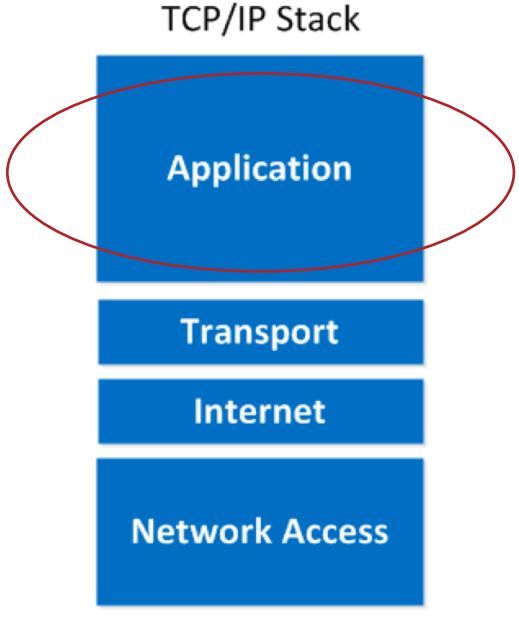
Yu Xiao

2022.02.08

Learning Outcomes

After the lecture, you should be able to

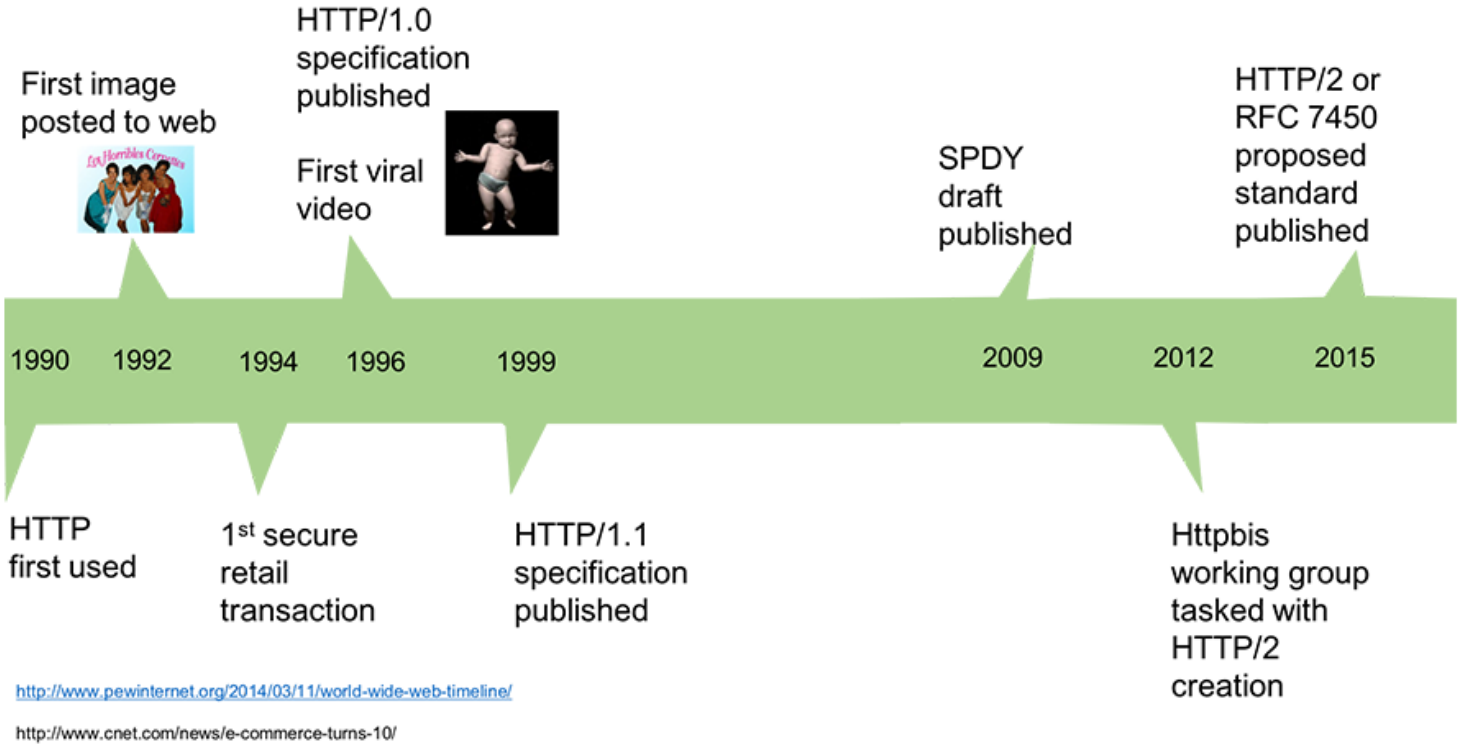
- **Describe the basic semantics of HTTP**
- **Understand the major differences between HTTP/1.1 and HTTP/2**



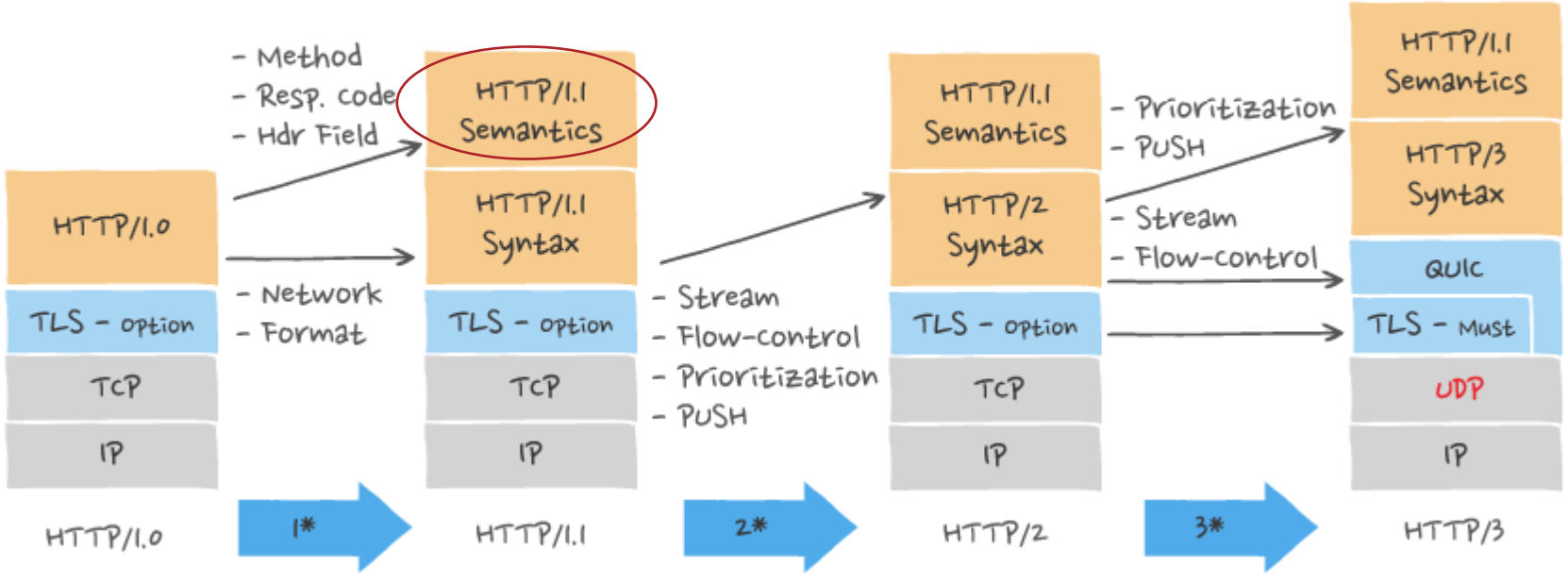
Hypertext Transfer Protocol (HTTP)

HTTP is an application layer protocol for distributed, collaborative, hypermedia information system.

Evolution of HTTP



HTTP protocol stack transition and comparison



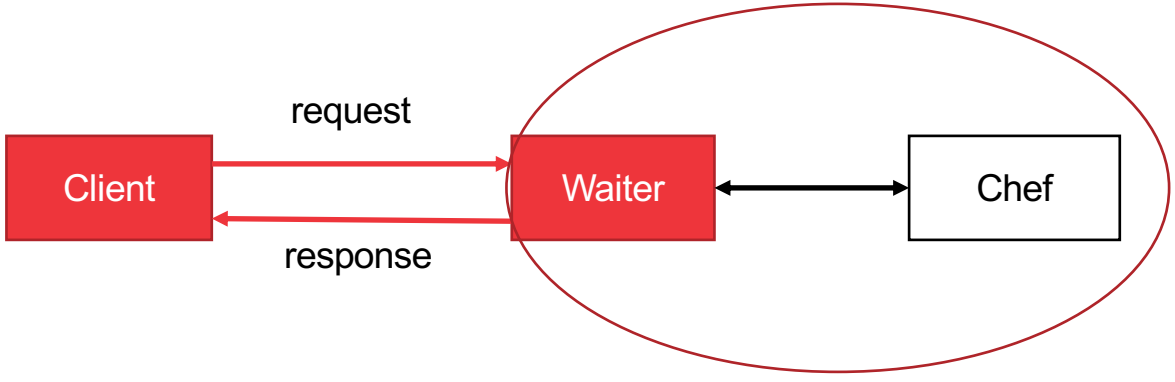
REST (Representational State Transfer)

- **REST is an architectural style**
- **Six Constraints:**
  - Client-server
  - Uniform interface
  - Stateless
  - Layered System
  - Cacheable
  - Code on Demand (optional)

Roy Fielding's doctoral dissertation (2000)

http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Client-Server Architecture



HTTP Semantics

- RFC 7231 https://httpwg.org/specs/rfc7231.html

“This document defines the semantics of HTTP/1.1 messages, as expressed by **request methods, request header fields, response status codes, and response header fields**, along with the payload of messages (metadata and body content) and mechanisms for content negotiation.”

Resources

"Any information that can be named can be a resource: a document or image, a temporal service (e.g. "today's weather in Los Angeles"), a collection of other resources, a non-virtual object (e.g. a person), and so on." – Roy Fielding

- **Resources are identified with Uniform Resource Identifiers (URIs)**
- A URI is a sequence of characters that identifies a logical or physical resource.
- **Multiple URIs may refer to same resource**
- **Separate from their representation(s)**

The state of resource at any particular timestamp is known as **resource representation**.

URI

Two types of URIs:

- Uniform Resource Locators (URLs) e.g., http://www.google.com
- Uniform Resource Names (URNs)
  - URN does not state which protocol should be used to locate and access the resource. It labels the resource with a persistent, location-independent unique identifier.

The generic form of any URI is

scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]

REST Resource Naming Guide

- **A resource can be a singleton or a collection**

  /customers /customers/{customerId}
- **A resource may contain sub-collection resources**

  /customers/{customerId}/accounts/{accountId}

REST Resource Naming Best Practices:

https://restfulapi.net/resource-naming/

Representation

- A "*representation*" is information that is intended **to reflect a past, current, or desired state of a given resource**, in a format that can be readily communicated via the protocol, and that **consists of a set of representation metadata** and a potentially unbounded stream of representation data.
- **Example:**
  - Resource: Person
  - Service: contact information (GET)
  - Representation:
    - *Name, address, phone number*
    - *JSON or XML format*

Uniform Interface

- **Defines the interface between client and server**
- **Simplifies and decouples the architecture**
- **For us this means:**
  - HTTP verbs
  - URIs (resource name)
  - HTTP response

HTTP Request Methods

Method	Description
GET	Transfer a current representation of the target resource
HEAD	Same as GET, but only transfer the status line and header section
POST	Perform resource-specific processing on the request payload
PUT	Replace all current representations of the target resource with the request payload
DELETE	Remove all current representations of the target resource
CONNECT	Establish a tunnel to the server identified by the target resource
OPTIONS	Describe the communication options for the target resource
TRACE	Perform a message loop-back test along the path to the target resource

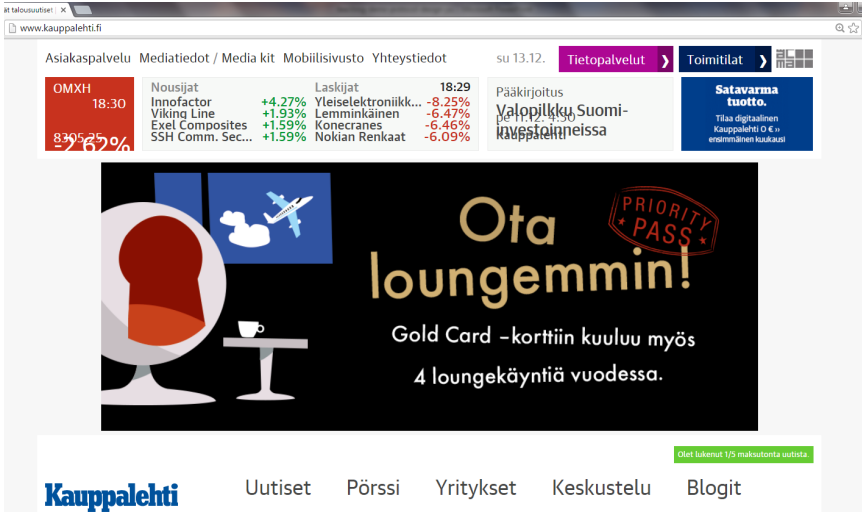
Request Header Fields

“A client sends request header fields to provide more information about the request context, make the request conditional based on the target resource state, suggest preferred formats for the response, supply authentication credentials, or modify the expected request processing.”
-- RFC7231

- **Controls (e.g. Cache-Control)**
- **Conditionals (e.g. if-match)**
- **Content Negotiation(e.g. Accept, Accept-Encoding)**
- **Authentication Credentials**
- **Request Context (e.g. from, user-agent)**

HTTP Verbs - Get

- **GET** me the webpage from Kauppalehti




```
GET / HTTP/1.1
HTTP/1.1 200 OK  (text/html)
GET /kauppalehti/Bootstrap.js HTTP/1.1
HTTP/1.1 304 Not Modified
GET /event/v3/pagestat?location=http%3A%2F%2Fwww.k
HTTP/1.1 200 OK
GET /api/news/frontpagelist/0/23 HTTP/1.1
HTTP/1.1 200 OK  (application/json)
GET /eas?target=_blank&EASformat=jsvars&EAScus=134
GET /api/news/popular-read/0/5?t=1450005378044 HTT
GET /api/news/group/new-york-times/0/1 HTTP/1.1
GET /api/news/group/financial-times/0/1 HTTP/1.1
GET /api/news/type/kommentti/0/1 HTTP/1.1
HTTP/1.1 200 OK  (application/json)
HTTP/1.1 200 OK  (application/x-javascript)
HTTP/1.1 200 OK  (application/json)
HTTP/1.1 200 OK  (application/json)
HTTP/1.1 200 OK  (application/json)
GET /alma/kauppalehti/serverComponent.php?r=62758.
OPTIONS /data HTTP/1.1
HTTP/1.1 200 OK
HTTP/1.1 200 OK  (text/javascript)
GET /api/news/online/0/30?profile=news&profile=fla
HTTP/1.1 200 OK  (application/json)
GET /blank.gif?url=http://www.kauppalehti.fi/ HTTP
HTTP/1.1 200 OK  (GIF89a)
GET /j0=,,,r=http%253A%252F%252Fwww.kauppalehti.fi
HTTP/1.1 302 FOUND  (text/plain)
GET /277x/http://flockler.com/thumbs/8685/vapo-kau
GET /277x/http://flockler.com/thumbs/11629/tieto_h
GET /adfserve/?bn=4450391;1x1inv=1;srctype=3;ord=1
HTTP/1.1 200 OK  (image/jpeg)
```

HTTP GET

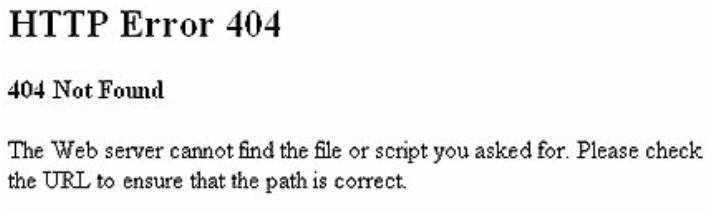
```
Hypertext Transfer Protocol
  GET / HTTP/1.1\r\n
  Host: www.kauppalehti.fi\r\n
  Connection: keep-alive\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n
  Upgrade-Insecure-Requests: 1\r\n
  User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.80 Safari/537.36\r\n
  Accept-Encoding: gzip, deflate, sdch\r\n
  Accept-Language: en-US,en;q=0.8,fi;q=0.6,zh-CN;q=0.4\r\n
```

```
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
  Date: Sun, 13 Dec 2015 11:16:16 GMT\r\n
  Server: Apache\r\n
  Vary: Cookie,Accept-Encoding\r\n
  Last-Modified: Fri, 11 Dec 2015 12:13:08 GMT\r\n
  ETag: "8739-5269e3f0c1d00-gzip"\r\n
  Accept-Ranges: bytes\r\n
  Content-Encoding: gzip\r\n
  Pragma: no-cache\r\n
  Cache-Control: no-cache, no-store, max-age=0, must-revalidate\r\n
  Expires: -1\r\n
  Content-Length: 9254\r\n
  Keep-Alive: timeout=600\r\n
  Connection: Keep-Alive\r\n
  Content-Type: text/html; charset=utf-8\r\n
```

HTTP/1.1 is a plain text protocol

HTTP Response

- **The first line of the response is called the *status* line and has a numeric status code and a text-based reason phrase**



- **HTTP status codes are primarily divided into five groups:**

Informational 1XX, Successful 2XX, Redirection 3XX, Client Error 4XX, and Server Error 5XX

Response Header Fields

These header fields give information about the server, about further access to the target resource, or about related resources.

- **Control Data (e.g. directs caching, or instructs the client where to go next)**
- **Validator Header Fields**
- **Authentication Challenges**
- **Response Context**

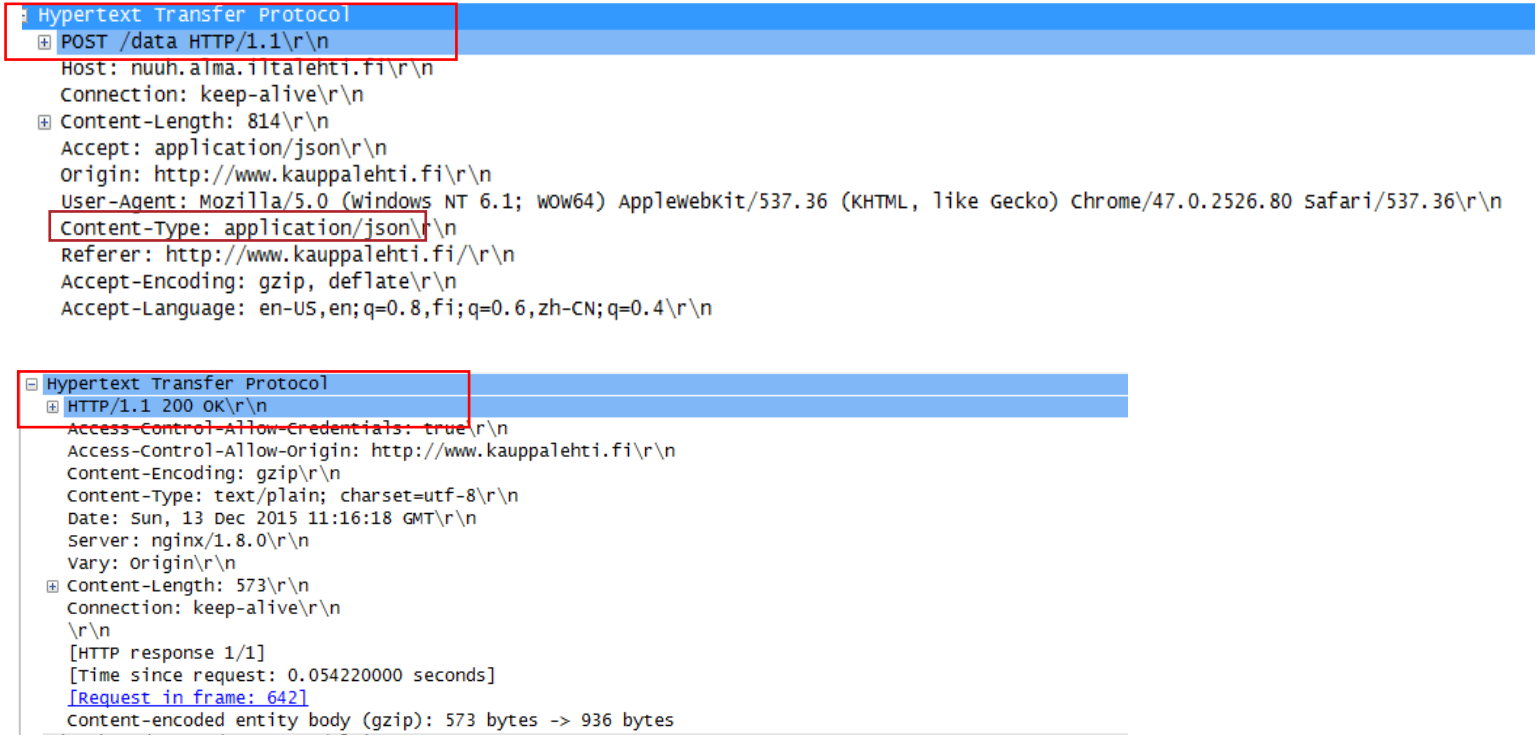
HTTP POST

- The POST request method requests that a web server accepts the entity enclosed in the body of the request message as a new subordinate of the web resource identified by the URI

- POST changes the state of the server

- Examples: submitting a web form

HTTP POST



Yu Xiao

HTTP PUT

- The PUT method requests that the enclosed entity be stored under the supplied URI
- If the URI refers to an already existing resource, it is modified; if the URI does not point to an existing resource, then the server can create the resource with that URI

POST vs PUT example

GET /device-management/devices : Get all devices

POST /device-management/devices : Create a new device

GET /device-management/devices/{id} : Get the device information identified by "id"

PUT /device-management/devices/{id} : Update the device information identified by "id"

DELETE /device-management/devices/{id} : Delete device by "id"

Reference: https://restfulapi.net/rest-put-vs-post/

HTTP is stateless

- **Server contains no client state**
- Each request contains all of the information necessary to understand the request, and all session state data should then be returned to the client at the end of each request
- Server will not store anything about latest HTTP request client made. It will treat each and every request as new.
- Statelessness ensures that each request can be treated independently

Code on demand

- **REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts**
- **Allow logic within clients (such as Web browsers) to be updated independently from server-side logic**
- **For example**
  - Jave applets
  - JavaScript
- **The only optional constraint**

Cacheable

- **Server responses (representations) are cacheable**
- **Services must be designed to produce accurate cache control metadata and return it in response messages.**
- Response messages are marked as cacheable or non-cacheable, either with explicit message metadata or as part of the contract definition.
- An optional consumer-side or intermediary cache repository enables the consumer to reuse cacheable response data for later request messages.
- Request messages must be comparable to determine whether or not they are equivalent.

Layered System

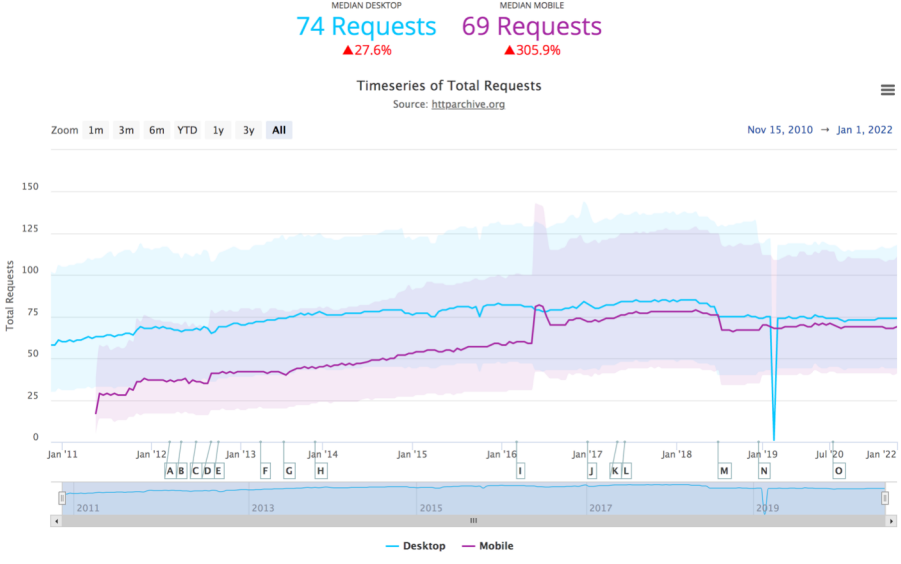
- Client can’t assume direct connection to server
- Allow software or hardware intermediaries (e.g. proxies, gateways, and firewalls) between client and server
- Improves scalability

HTTP/1.1 vs. HTTP/2

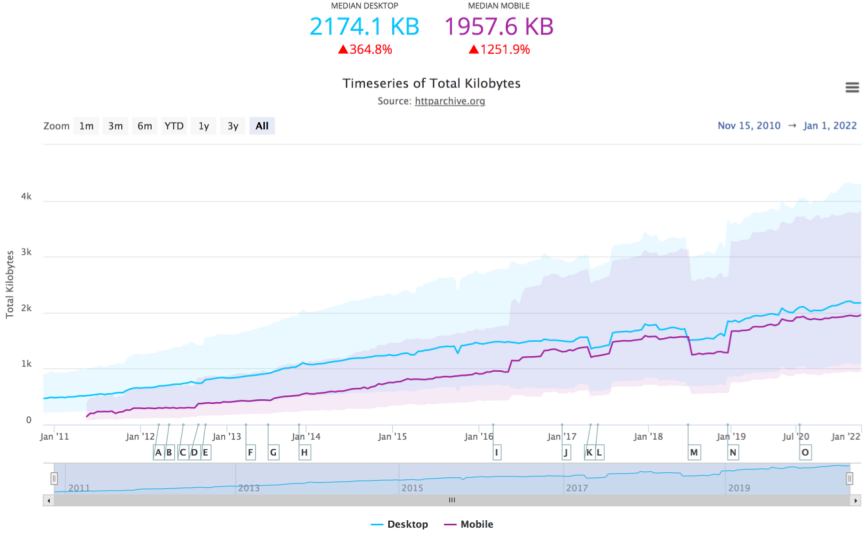
- Both HTTP/1.1 and HTTP/2 run on top of TCP
- HTTP/2 maintains high compatibility for methods, status code, URI's and header fields with HTTP/1.1
- HTTP/2 supports all the core features of HTTP/1.1, but aims to be more efficient in several ways
- **HTTP/2 is a binary protocol, instead of a plain text protocol**

#Requests and #KB increased

The number of resources requested by the page

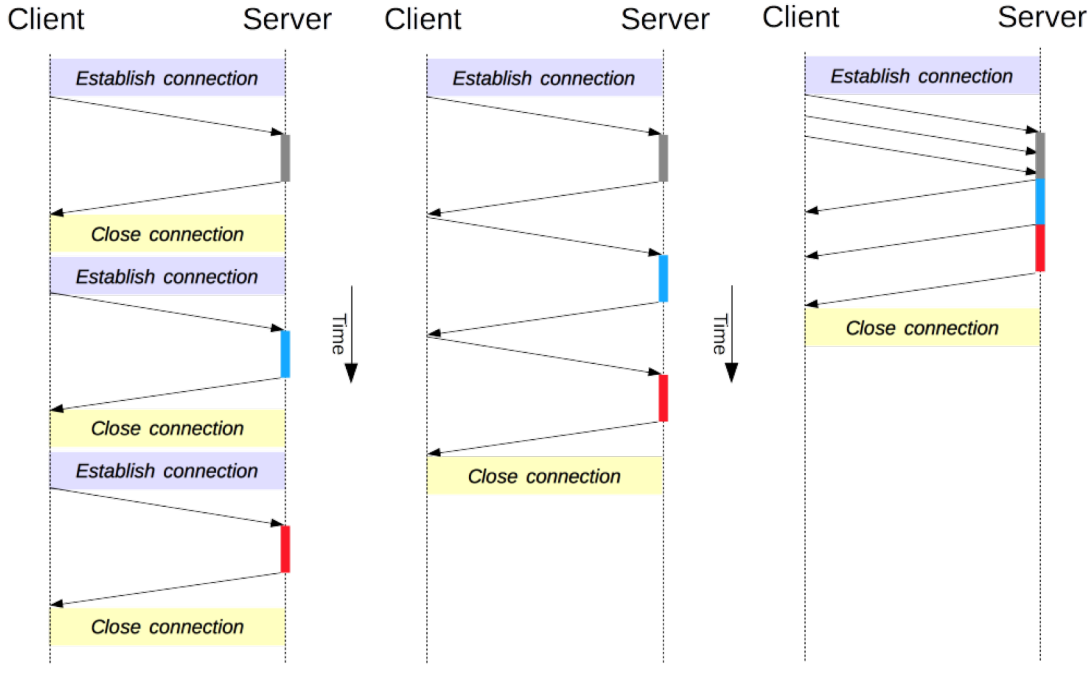


The sum of transfer size KB of all resources requested by the page



https://httparchive.org/reports/state-of-the-web

HTTP Pipelining



HTTP/1.0

1 request per TCP connection

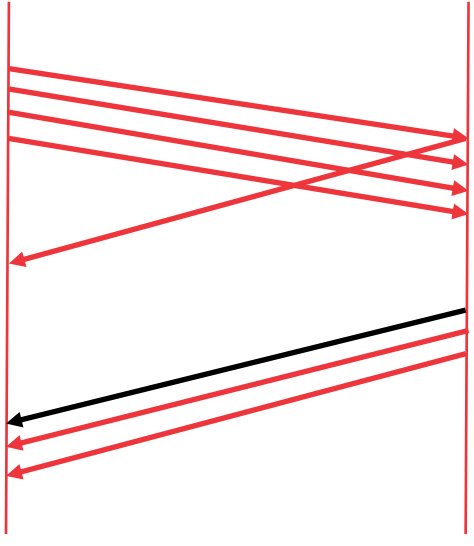
HTTP/1.1

Multiple requests through one TCP connection

Pipelining is the process to send successive requests, over the same persistent connection, without waiting for the answer.

Head of Line Blocking

- **First-in-first-out:** The server must send its responses in the same order that the requests were received.



A large or slow response can block others behind it

Primary Goals of HTTP/2

- **Reduce latency** by enabling full request and response multiplexing
- **Minimize protocol overhead** via efficient compression of HTTP header fields
- **Add support for request prioritization and server push**

Self-test

- HTTP request methods: GET, PUT, POST
- HTTP response status line
- Is HTTP stateless or not?
- What is head of line blocking?