HTTP Protocol

IF3110 – Web-based Application Development
School of Electrical Engineering and Informatics
Institut Teknologi Bandung

Reference

- Leon Shklar and Richard Rosen, **Web application architecture: principles, protocols, and practices**, John Wiley & Sons Ltd (2003)
- https://httpwg.org/specs/

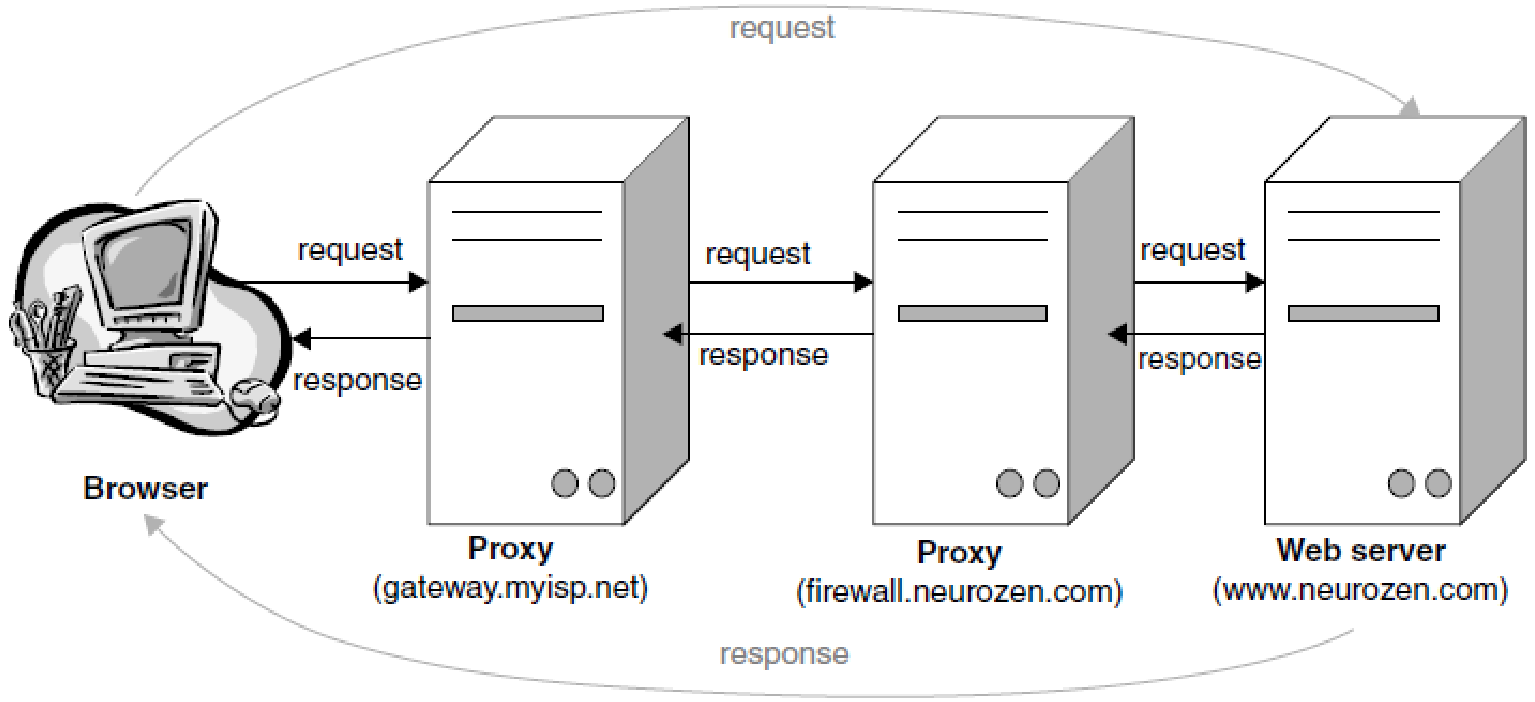
About HTTP

- Basic Ingredient Protocol for World Wide Web
- Simple – strength and weakness
- Doesn’t manage the state with limited functionality
- Application Layer Protocol founded by TCP

HTTP Protocol and Structure

- *request* and *response* paradigm
- *request* and *response* has similar structure
  - couple lines of *header*
  - empty line, followed *message body*
- *Stateless protocol* – a transaction is composed of a single *request* from *client* and a *response* from *server*

Request Response Virtual Circuit



Anatomy of HTTP 1.0

Web Client

Connect: Request

Web Server

Request Line	`GET / HTTP/1.0`
Request Header	`Host: www.yahoo.com`
Request Delimiter	`CRLF`

Response: Close

Response Status	`HTTP/1.0 200 OK`
Response Header	`Date: Tue, 16 Feb 2010 19:21:24 GMT`
	`Content-Type: text/html;`
Response Delimiter	`CRLF`
Response Body	`<html><head><title>Yahoo!</title>`

HTTP Request Structure

```
METHOD /path-to-resource  HTTP/version-number
Header-Name-1: value
Header-Name-2: value

[  optional request body  ]
```

```
GET /q?s=YHOO HTTP/1.1
Host: finance.yahoo.com
User-Agent: Mozilla/4.75 [en] (WinNT; U)
```

```
HEAD http://www.cs.rutgers.edu/~shklar/ HTTP/1.1
Host: www.cs.rutgers.edu
User-Agent: Mozilla/4.75 [en] (WinNT; U)
```

HTTP Request Structure

- *Request Line:*
  - *Request Methods*: **GET, POST, HEAD**, PUT, DELETE, TRACE, OPTION, CONNECT, PATCH
  - access URL
  - Version HTTP: 1.0 or 1.1
- Header – variable: value pairs
  - Host
  - Content-Type
  - Content-Length
  - User-Agent
  - Cookie, dll
- *Request body*

HTTP Response Structure

```
HTTP/version-number   status-code   message
Header-Name-1: value
Header-Name-2: value

[ response body ]
```

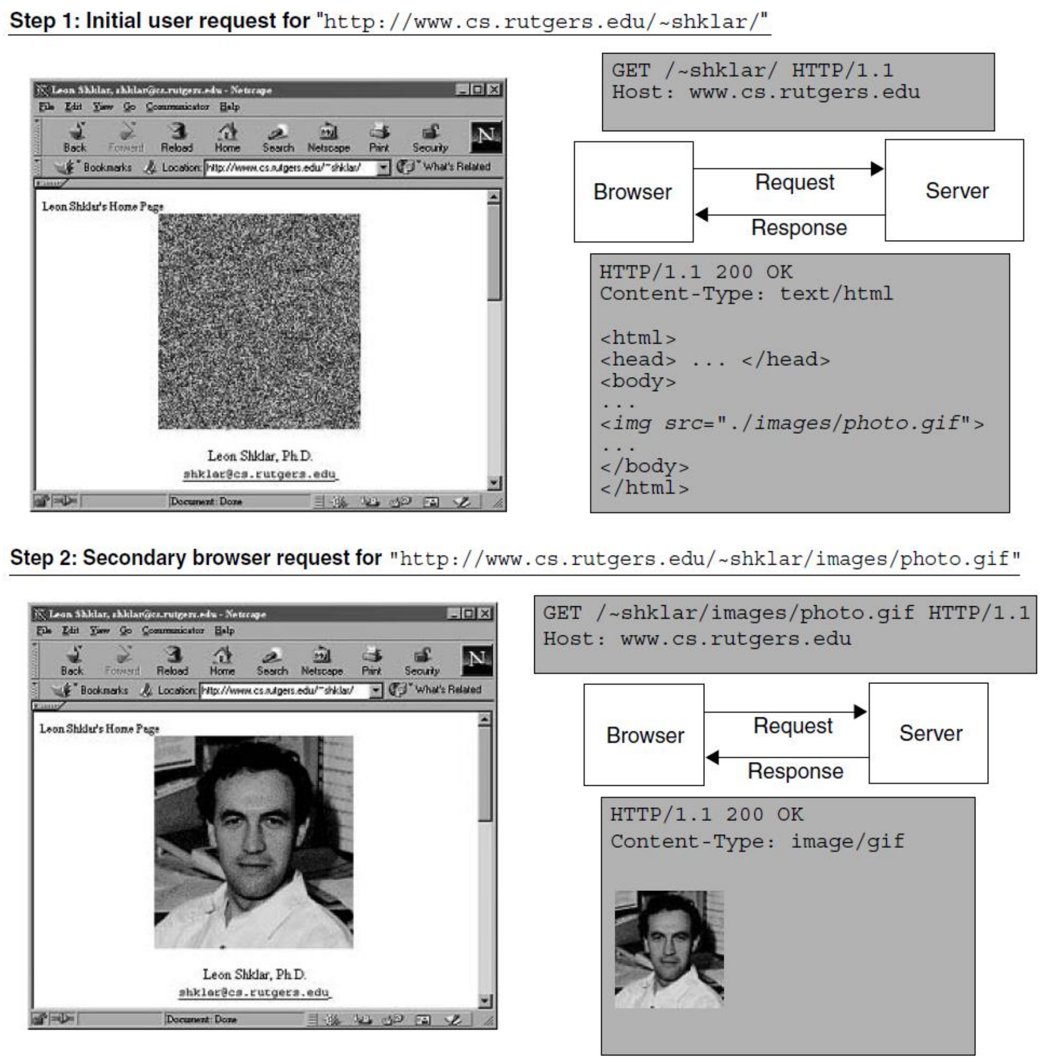
```
HTTP/1.0 200 OK
Date: Sat, 03 Feb 2001 22:48:35 GMT
Connection: close
Content-Type: text/html
Set-Cookie: B=9ql5kgct7p2m3&b=2;expires=Thu,15 Apr 2010 20:00:00 GMT;
path=/; domain=.yahoo.com

<HTML>
<HEAD><TITLE>Yahoo! Finance - YHOO</TITLE></HEAD>
<BODY>
  ...
</BODY>
</HTML>
```

HTTP Response Structure

- *Status line:*
  - Version HTTP: 1.0 or 1.1
  - *Status Code* and some description
- Header – variable: value pairs
  - Content-Type
  - Content-Length
  - Set-Cookie
  - Date, dll
- *Response body*

- Once receive the document; browser parses the doc to define additional resources to be retrived


Step 1: Initial user request for "http://www.cs.rutgers.edu/~shklar/"

```
GET /~shklar/ HTTP/1.1
Host: www.cs.rutgers.edu
```

```
HTTP/1.1 200 OK
Content-Type: text/html

<html>
<head> ... </head>
<body>
...
<img src="./images/photo.gif">
...
</body>
</html>
```

Step 2: Secondary browser request for "http://www.cs.rutgers.edu/~shklar/images/photo.gif"

```
GET /~shklar/images/photo.gif HTTP/1.1
Host: www.cs.rutgers.edu
```

```
HTTP/1.1 200 OK
Content-Type: image/gif
```

Figure 3.2 Sequence of browser requests for loading a sample page

Request Methods

- GET
  - most simple
  - doesn’t contain *request body*
  - *request parameters* will be added in the URL *query string* (after“?”)
  - Only retrieve the resource/data, without any other effects (inc. modification/deletion)
- POST
  - *request body* contains *request* parameters
  - Submits data to be processed to a specified resource
  - URL doesn’t contain any data (suitable for submit FORM)

Form Processing

- Form w/ POST method, will use HTTP POST to send data, with

```
content-type: application/x-www-form-urlencoded
```

- Query parameter will be provided as pairs of <u>type: value</u>

- File upload uses

```
content-type: multipart/form-data
```

Form Processing

```
POST /enlighten/calais.asmx/Enlighten HTTP/1.1
Host: api.opencalais.com
Content-Type: application/x-www-form-urlencoded
Content-Length: length

licenseID=string&content=string&paramsXML=string

--------------------------------
```

GET vs POST

GET

- can be cached
- remain in the browser history
- can be bookmarked
- Data is visible to everyone in the URL
- have length restrictions
- should be used only to retrieve data
- Only ASCII characters allowed

POST

- never be cached
- do not remain in the browser history
- cannot be bookmarked
- Data is not displayed in the URL
- have no restrictions on data length
- No restrictions. Binary data is also allowed

Request Methods

- HEAD
  - Similar to GET, but server MUST NOT return a message body in the response
  - Server only returns *header*
  - To support *cache* with *content modification information* (*Last-Modified*)

Request Methods

- PUT
  - to store a resource on a particular URI
  - if the URI refers to existing resource then the resource is being updated
- DELETE
  - to delete a resource.
- TRACE
  - send back the request received by the server
  - client can identify what the additional info added in a HTTP request (e.g., by http proxy)

Request Methods

- OPTIONS
  - return HTTP methods supported by the server on a particular URL
- CONNECT
  - convert request into a transparent TCP/IP tunnel,
  - used in SSL-encrypted used in HTTPS
- PATCH
  - used in modification on a part of the resource

Status Code

- Inform browser or proxy whether *response* is as expected
  - 1xx information
  - 2xx success
  - 3xx redirection
  - 4xx client request error
  - 5xx server error

HTTP Header

- General Header
  - `Date: Sun, 11 Feb 2001 22:28:31 GMT`
    Date time created *message*
  - `Connection: Close`
    Client or Server define whether the connection is maintained/not
- Request Header
  - `User-Agent: Mozilla/4.75 [en] (WinNT; U)`
    browser user agent
  - `Host: www.neurozen.com`
    to support virtual host
  - `Referer: http://www.cs.rutgers.edu/index.html`
    URL of the referral

HTTP Header

- Response Header
  - `Location: http://www.mywebsite.com/Page.html`
    Page intended to visit (*redirect*)
  - `Server: Apache/1.2.5`
    Server ID
- EntityHeader
  - `Content-Type: mime-type/mime-subtype`
    type of message body
  - `Content-Length: xxx`
    length of message body
  - `Last-Modified: Sun, 11 Feb 2001 22:28:31 GMT`
    modification date of the content

Virtual Hosting

```
GET http://finance.yahoo.com/q?s=YHOO HTTP/1.1
Host: finance.yahoo.com
```

```
GET /q?s=YHOO HTTP/1.1
Host: finance.yahoo.com
```

Authentication

```
HTTP/1.1 401 Authenticate
Date: Mon, 05 Feb 2001 03:41:23 GMT
Server: Apache/1.2.5
WWW-Authenticate: Basic realm="Chapter3"
```

```
GET /book/chapter3/index.html HTTP/1.1
Date: Mon, 05 Feb 2001 03:41:24 GMT
Host: www.neurozen.com
Authorization: Basic eNCoDEd-uSErId:pASswORd
```

Session Management

```
GET /movies/register HTTP/1.1
Host: www.sample-movie-rental.com
Authorization:...
```

```
HTTP/1.1 200 OK
Set-Cookie: CLIENT=Rich; path=/movies
...
```

```
GET /movies/rent-recommended HTTP/1.1
Host: www.sample-movie-rental.com
Cookie: CLIENT=Rich
```

Caching control

```
GET /~shklar/ HTTP/1.1
Host: www.cs.rutgers.edu
If-Modified-Since: Fri, 11 Feb 2001 22:28:00 GMT
```

Persistent Connection

- HTTP 1.0 uses TCP separately for each request
  - not efficient
  - slow (i.e., high-latency)
- HTTP 1.1 uses a persistent connection, can be used by several requests

```
Connection: Close
Connection: Keep-Alive
```

HTTP 1.1 vs 1.0

- Additional Methods (PUT, DELETE, TRACE, CONNECT + GET, HEAD, POST)
- Additional Headers
- Transfer Coding (chunk encoding)
- Persistent Connections (content-length matters)
- Request Pipelining

HTTP Sometime Back

- Using HTTP 1.1 since 1997 / 1999
  - Connection: keep-alive
  - Head of Line Blocking
- But we still use N TCP Connections per origin
- No Header Compression

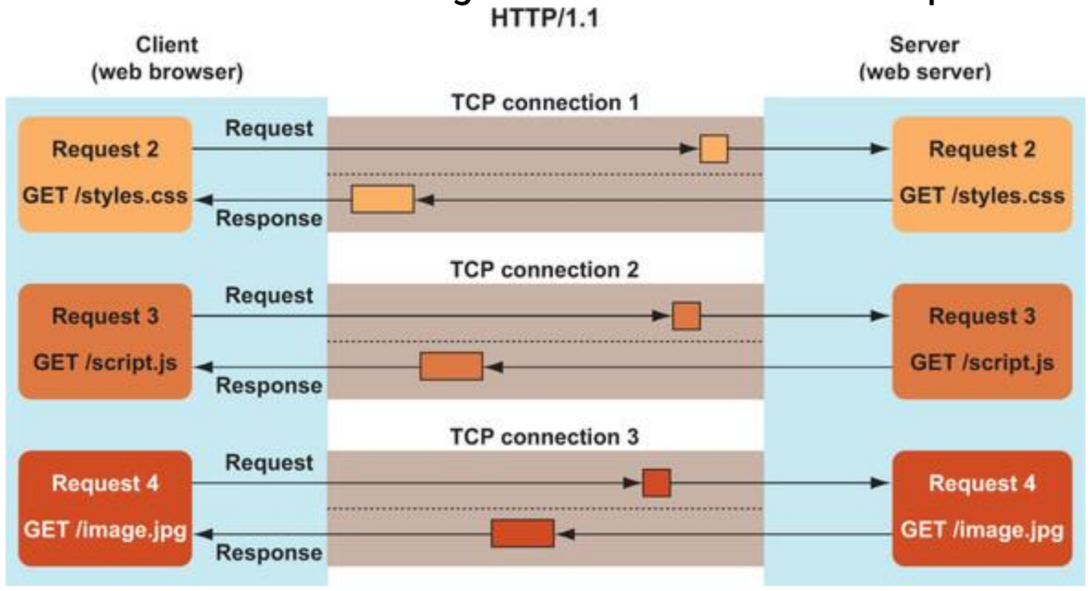
- And Many Hacks because requests are evil
  - Spriting of Images
  - Resource Inlining
  - Concatenation of files
  - Domain Sharding
  - CDNs

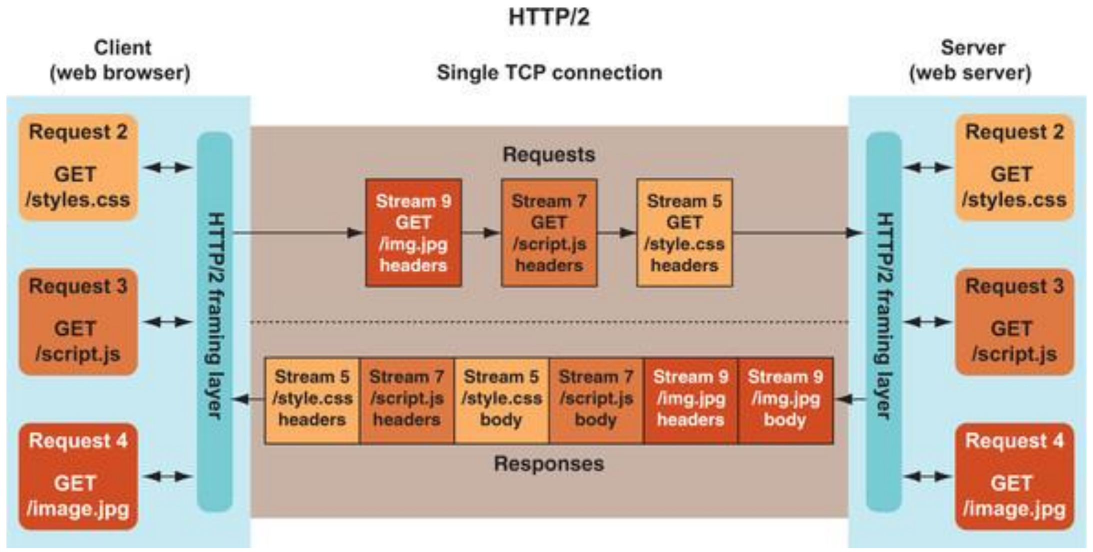
HTTP/2

- Addressing HTTP 1.1 Performance Issues:
    - Binary rather than textual protocol
    - Multiplexed rather than synchronous
    - Flow control
    - Stream prioritization
    - Header compression
    - Server push

Multiple TCP connection vs Single Connection

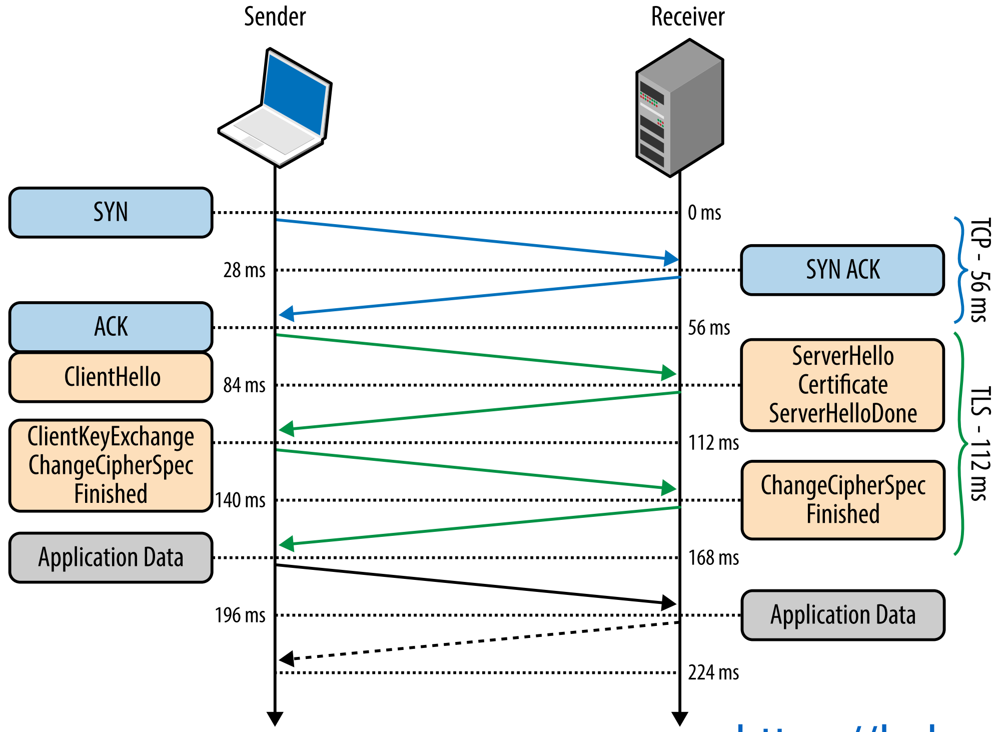
- Multiple HTTP/1 requests in parallel require multiple TCP connections.
- Most browsers open up to six connections per domain in parallel for this reason.
- But when the max. connection reached => blocking occurs. Also, TCP connections are expensive.
- A technique called domain sharding splits resources into many domains to address this limitation.
- HTTP/1 may allow "persistent connection": using connection:keep-alive header to use one connection for several requests. However this introduce *head-of-line blocking*: the current request must complete before the next one can be sent.
- HTTP/2 allows single TCP connection multiplex streams of resources over frames which may be interleaved.





HTTPS

- Is HTTP over TLS over TCP
- Hence, TCP 3 way handshake and certificate exchange is performed before actual HTTP protocol application data.



https://hpbn.co/