Lec04_Web

Glossary

1. *HTTP*: Hyper Text Transport Protocol
2. *URL*: Uniform Record Locator
3. *RTT*: Round-trip Time

Web

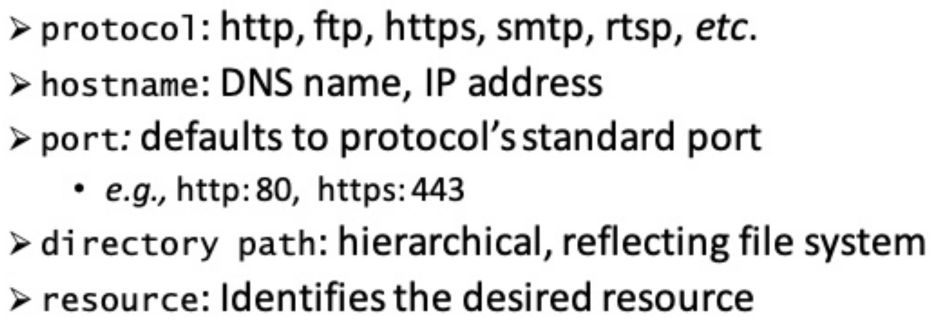
World Wide Web: a distributed database of "pages" linked through HTTP.

- Infrastructure:
    a. Clients
    b. Servers (DNS, CDN, datacenters)
- Content:
    a. URL: naming content
    b. HTML: formatting content
- Protocol for exchanging information: HTTP

URL: Uniform Record Locator

Format of URL:

protocol://host-name[:port]/directory-path/resource



- `protocol`: http, ftp, https, smtp, rtsp, *etc.*
- `hostname`: DNS name, IP address
- `port`*:* defaults to protocol's standard port
    - *e.g.,* http: 80, https: 443
- `directory path`: hierarchical, reflecting file system
- `resource`: Identifies the desired resource

Why we use URL:

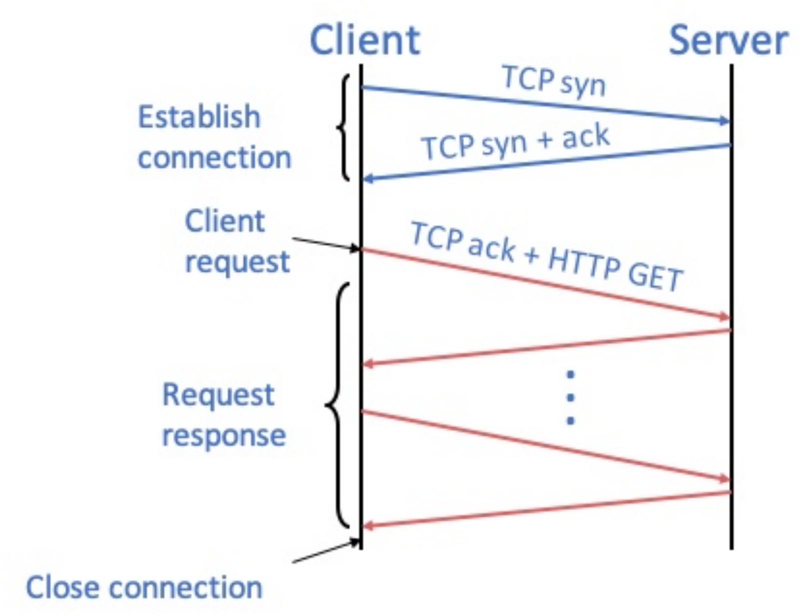
1. Extend the idea of hierarchical hostnames to include anything in a file system
2. Extend to program executions as well. Server side processing can be included as well in the name.

HTTP: Hyper Text Transport Protocol

- Client-server architecture.
    a. Server is "always on" and "well known"
    b. Clients initiate contact to server.

- Synchronous request/reply protocol. (Runs over TCP, port 80)
- Stateless.
- ASCII format.

Steps in HTTP request/response:



Method types (HTTP 1.1)

1. Get, head; 2. Post; 3. Put; 4.Delete.

Pattern of message

1. Client-to-server communication (HTTP request message)

- Request line
- Request headers
- Body (the last line is a blank line which indicates the end of message)

2. Server-to-client communication (HTTP response message)

- Status line
- Response headers
- Body

HTTP is stateless

Each request-response treated independently. Servers are not required to retain states.

- Pro: improves scalability on the server-side.
    a. Failure handling is easier
    b. Can handle higher rate of requests
    c. Order of requests doesn't matter
- Cons: some applications need persistent state.
    a. Need to uniquely identify user or store temporary info

Cookies

Used to keep state in HTTP in client-side. Client stores small state on behalf of server and sends state in future requests to the server. Can provide authentication.

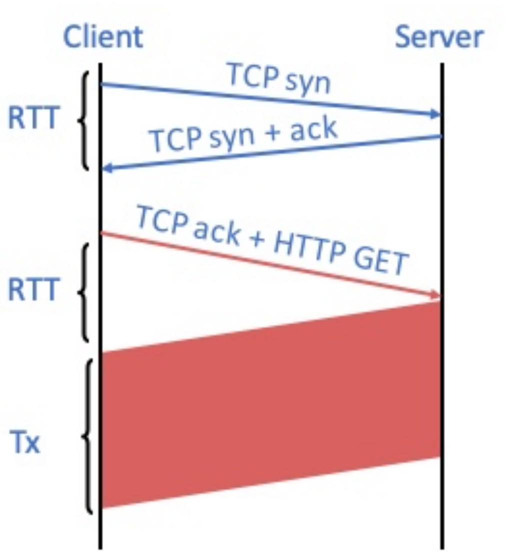
HTTP performance

in HTTP 1.0 (non-persistent connection)

Most web pages have multiple **objects** like HTML file and bunch of embedded images. In HTTP 1.0 it is a non-persistent connection so we retrieve those objects one item at a time and even set up new TCP connection per (small) object.

RTT (round-trip time) is the time for a small packet to travel from client to server and back.

Response time: 1 RTT for TCP setip; 1 RTT for HTTP request and first few bytes; transmission time. The total = 2RTT + transmission time.



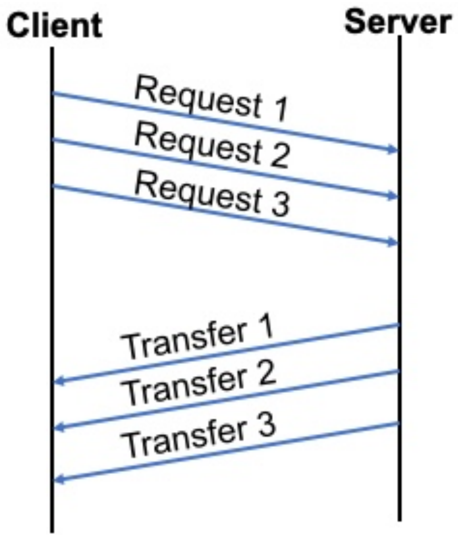
Thus in HTTP/1.0 we need $2RTT + \Delta$ for each object in the HTML file and one more $2RTT + \Delta$ for the HTML file itself. Doing the same thing over and over again is inefficient.

in HTTP 1.1 (persistent connection)

- Use multiple connection in parallel
- Does not necessarily maintain order of responses.
- Good to Client and server but increase the burden of network.

In HTTP/1.1, we maintain TCP connection across multiple requests including transfers subsequent to current page. Clients or servers can tear down connections. The **advantage** of doing so is to avoid overhaead of connection set-up and tear-down, and allow underlying layers (e.g., TCP) to learn about RTT and bandwidth characteristics.

Pipelined requests & responses: Batch requests and responses to reduce the number of packets and multiple requests can be contained in one TCP segment. This can result in a great improvement in page loading time especially over high latency connections.


Client
Server
Request 1
Request 2
Request 3
Transfer 1
Transfer 2
Transfer 3

Caching