HTTP, Web and Web services

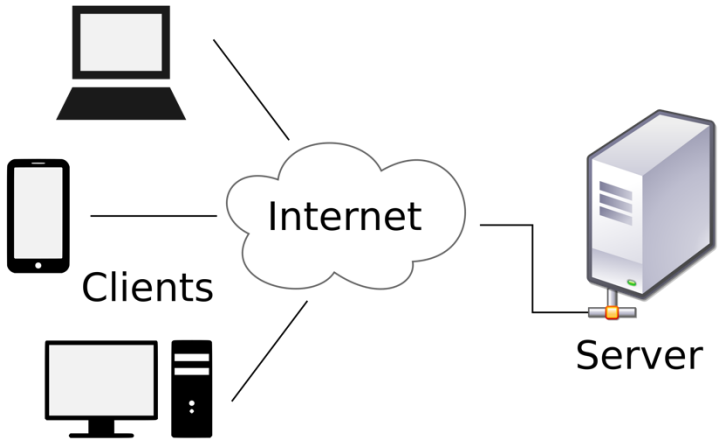
Dario Salvi

Materials

- Internetworking with TCP/IP Vol 1 (ed. 6)
  - Chapter 25
- Online resources
  - https://en.wikipedia.org/wiki/Client%E2%80%93server_model
  - https://student.cs.uwaterloo.ca/~cs446/1171/Arch_Design_Activity/Peer2Peer.pdf
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
  - https://en.wikipedia.org/wiki/HTTPS
  - https://www.ibm.com/docs/en/cics-ts/5.1?topic=services-what-is-web-service
  - https://www.w3schools.com/html/html_xhtml.asp
  - https://www.restapitutorial.com/
  - https://restfulapi.net/

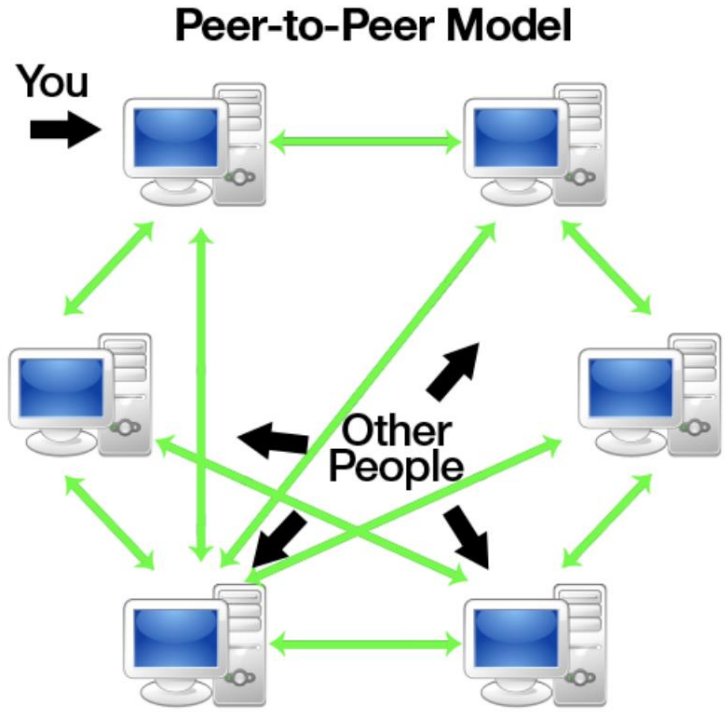
Client-server architecture

- “The client-server architecture refers to a system that hosts, delivers, and manages most of the resources and services that the client requests. In this model, all requests and services are delivered over a network, and it is also referred to as the networking computing model or client server network.”
  - Multiple users’ computers, phones, devices connected to a central server via an Internet connection.
  - The client sends a **request** for data, and the server accepts and accommodates the request, sending a **response** back to the user who needs them.



Peer to peer architecture

- A peer-to-peer (P2P) architecture consists of a decentralized network of **peers nodes** that are **both clients and servers**.
- P2P networks distribute the workload between peers, and all peers contribute and consume resources within the network without the need for a centralized server.
  - Some nodes can be “super nodes”, offering more services than others.
- P2P architecture can be completely decentralized.
  - However, sometimes there is a central tracking server to help peers find each other and manage the network.



History of the web

- In the early days of Internet FTP was used mostly (1/3 of traffic).
- Gopher was a competitor, but lost the battle.
  - Was more rigid and resembled a file system more than a loosely linked network of documents.
  - https://en.wikipedia.org/wiki/Gopher_(protocol)
- Tim Berners-Lee, a British scientist, invented the World Wide Web (WWW) in 1989, while working at CERN.
- The Web was originally conceived and developed to meet the demand for automated information-sharing between scientists in universities and institutes around the world.
- In 1995 HTTP took over FTP as highest traffic percentage.
- Nowadays dominates all Internet traffic.

Architecture

- The web is a network of documents (web pages), accessible on the Internet and linked by *hyperlinks*.
- Documents can be many things:
  - HTML pages.
  - Images.
  - Video / audio.
  - Files.
- A **web server** hosts a bunch of pages and a **web client (browser)** downloads them, displays them and allows following the links.
- HTML (HyperText markup language) documents include content and “commands” or *tags* that style the content or add meta-information (like links or images).

Uniform resource locators

- Each document is identified by a URL.
- Format: protocol :// hostname : port / path ? query
  - Protocol: usually http or https.
  - Hostname: the IP address or DNS name.
  - TCP port number (standard: 80 or 423).
  - Path: path inside the host.
  - Query: additional parameters to be passed.
- Within documents, it is possible to use *relative* URLS, where only the path and query are specified (the rest is taken from the current document).

HTML

<html>

Please consult the

University webpage.

</html>

- The browser understands that the “University webpage” is a link and makes it clickable.

- Learn HTML: https://www.w3schools.com/html/default.asp

Hypertext transfer protocol

- HTTP is an **application** layer protocol on top of TCP.
- **Client/server** and **request/response**. Client establishes a connection and makes a request, server responds.
- **Stateless**: each request is self contained and does not affect others.
- **Bidirectional:** clients can also send documents.
- Capabilities (such as compression, character set) can be **negotiated** between client and server.
- Browsers can **cache** content they have already seen recently.
- **Proxy servers** can be allowed as additional caches.

GET

- Commands are in plain text.
- A client wants a document (e.g. http://mau.se/info)
  - Get IP address from mau.se
  - Establish a TCP connection on destination port 80 (source port is dynamic).
  - Send **GET** command over TCP:
    - GET /info HTTP/1.1
  - Server replies by sending the page.
  - Browser parses the page and shows the content.
  - If the page includes additional materials (e.g. images), further GETs are issued, one per image.
- If an errors occurs, the server can reply back with a "**status code**" plus a page for the user to read. See: https://www.restapitutorial.com/httpstatuscodes.html

Persistent connections

- Initially HTTP had one TCP connection per document / image.
  - Establishing a connection implies overhead -> latency.

- HTTP 1.1 (1997) introduced the possibility to issue more commands on one TCP connection.
  - The client needs to identify where each file starts and ends.
  - -> on each reply, HTTP specifies the **length** of the coming file.
  - But length is not always known! (pages can be generated on the fly).
    - In this case, the server warns the client and must close the connection at the end of the file.

Header

- HTTP can send meta and signalling information in a “**header**”.
- Headers are sent before the document.
- Headers are also **text based** and readable.
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers
- Some items are borrowed from emails:
  - **Content-Length**: size of the document.
  - **Connection**: **close**. Used when length is not known.
  - **Content-Type**: MIME type of the content. https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types
  - **Content-Encoding**: character encoding.
  - **Content-Language**: language(s) used in the document.

Header: content negotiation

- Permits to negotiate capabilities.
- Server driven:
  - Client sends a list of preferences (e.g. languages).
    - **Accept**: text/html, text/plain; q=0.5 (q means willingness to accept).
    - **Accept-Language**: en, it
  - Server selects the ones it can satisfy.
- Client driven:
  - Client asks for available preferences.
  - Server provides possibilities.
  - Browser selects one and sends further request for document.

Header: conditional request

- Browser can specify condition under which server can send document.
- Example: **If-Modified-Since**
  - If-Modified-Since: Mon, 01 Apr 2013

Proxies

- Proxy servers are used to optimise traffic and caching.
- Non-transparent:
  - The user configures the proxy in the browser.
- Transparent:
  - The user does not configure it, it’s in the middle and catches all connection to port 80.
- HTTP has explicit support for proxies, for example for authentication or for controlling how many proxies are allowed between the server and the browser.

Caching

- Caches improve efficiency by eliminating unneeded transfers.
- When a document is accessed it’s downloaded and stored locally.
  - A further download can be avoided!
- How long should a page be kept in cache?
  - Server can specify documents’ expiry.
  - Clients can specify maximum age of cached copies willing to receive.

Cookies

- Cookies are small pieces of data that are sent by the server and re-sent back at each subsequent request.
  - **They allow stateful interaction**!
- Typically used for:
  - Session management
    - Logins, shopping carts, game scores, or anything else the server should remember
  - Personalization
    - User preferences, themes, and other settings
  - Tracking
    - Recording and analysing user behaviour
- Cookies are sent with every request, so they can worsen performance.
  - Usually kept limited to a session identifier.

HTTPS

- HTTP secure is HTTP over **Transport Layer Security** (TLS).
- Uses encryption for secure communication.
  - Authentication of the accessed website. (the website is what it claims to be)
  - Protection of the privacy and integrity of the exchanged data while it is in transit. (data is encrypted)
    - But underlying IP and TCP are unecrypted.
- Requires a trusted third party to sign server-side digital certificates.
  - Web browsers know how to trust HTTPS websites based on **certificate authorities** that come pre-installed in their software.
  - Certificate authorities are trusted by web browser creators to provide valid certificates.
  - One can install additional certificates at own risk.

HTTP2

- Mostly developed by Google
- Standardised in 2015 (RFC 7540)
- Currently supported by 40% of the top 10 M websites
- Novelty only related to how data is framed and transported
- Allows the server to "push" content it knows a web browser will need to render a web page, without waiting for the browser to examine the first response, and without the overhead of an additional request cycle.
- In HTTP 1, when the number of allowed parallel requests in the browser is used up, subsequent requests need to wait for the former ones to complete. HTTP2 solves this “head-of-line blocking ” with multiplexing of requests and responses.

Demo

- Open Google Chome and select “developer tools”
- Open the the Network tab
- Open a web page and inspect the information on the tab.

Extensible Markup Language (XML)

- Simple text-based format for representing structured information
  - Examples: documents, data, configuration, books, transactions, invoices,..

- Similar to HTML, but tags are custom and has slightly stricter rules
  - Actually there was a version of HTML based on HTML: XHTML

```
<book>
   <id>32344</id>
   <author>
      <name>Tanenbaum</name>
   </author>
   <title>Computer Networks</title>
   <publisher>Pearson Education</publisher>
   <year>2013</year>
</book>
```

JavaScript Object Notation (JSON)

- Lightweight data-interchange format. Based on JavaScript (ECMA script)
- Easy for humans to read and write. Easy for machines to parse and generate.

```
{
   "book": {
    "id": 32344,
    "author": {
     "name": "Tanenbaum"
    },
    "title": "Computer Networks",
    "publisher": "Pearson Education",
    "year": 2013
   }
}
```

JSON

XML

Web services

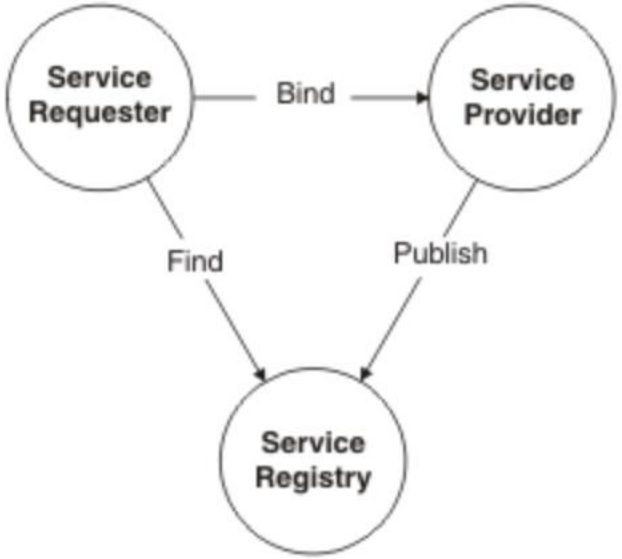
- HTTP can be used to deliver any type of data, not only webpages.
  - One can use it as a general purpose client-server protocol.
  - But some conventions are needed in order to create an application!
    - What type of data? Binary? Text? What format?
    - What actions / verbs are allowed? Read? Write? Update? Delete?
    - How can one locate a certain piece of data? Using URLs?
    - How can the data be described?
    - Is the format of the data mandatory or is there some flexibility?

Web services

- A web service is described using a standard, formal XML notion: Web Service Definition Language (WSDL)
  - provides all of the details necessary to interact with the service, including message formats (that detail the operations), transport protocols, and location
- Messages are text-based
  - Inefficient format but:
  - Supported by all hardware, programming languages, operating systems etc.
  - Applications are loosely coupled, component oriented, multi-platform

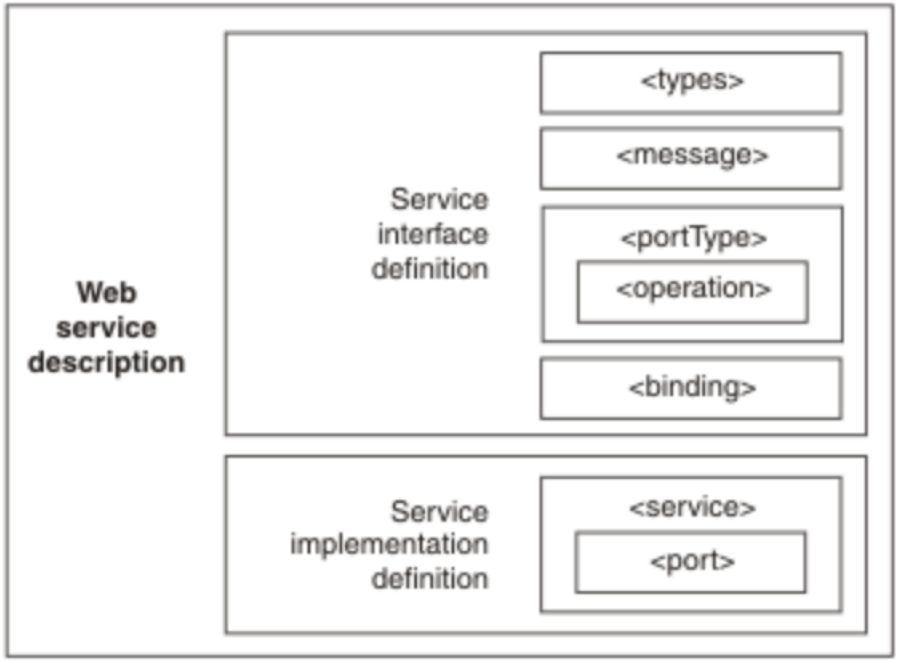
Web services architecture

- Provider:
  - Provides the service(s)
- Requester:
  - Requests the service(s)
- Registry:
  - Central location where service providers can publish their service descriptions and where service requesters can find those service descriptions (WSDLs).



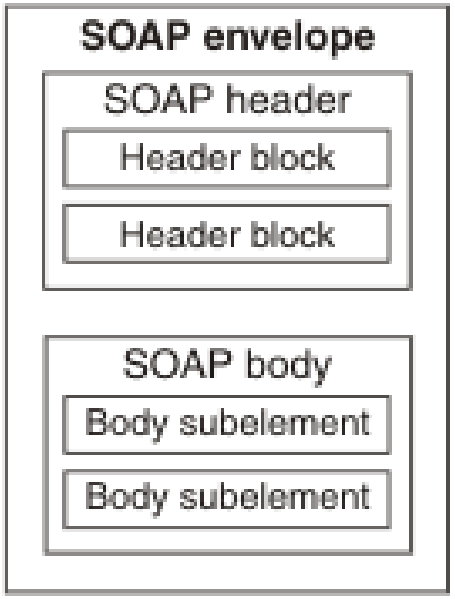
WSDL

- Service Interface Definition:
  - **Types**: data type definitions using XML Schema.
    - Example: Book (id, author, title), Loan (bookId, start, end)
  - **Message**: data types used to define the input and output parameters of an operation.
    - Example: a borrow requires user and book
  - **PortType**: collection of operations.
    - Example: a library API
  - **Operation**: function executed using messages.
    - Example: find (title), borrow (userId, bookId, start, end)
  - **Binding**: protocol, data format, security, etc.
    - Example: borrow (http, borrow message, authenticated).
  - **Service**: actual implementation on a server
    - Example: server for biblio.net
  - **Port**: URL of an operation
    - Example: https://biblio.net/api/borrow



Simple Object Access Protocol (SOAP)

- W3C standard for XML messages
- Envelope: root element of the message
- Header: used to pass application-related information to be processed along the path
- Body: information intended for the ultimate recipient of the message



RESTful APIs

- REpresentational State Transfer
- Invented by Roy Fielding in his dissertation in 2000
- Simpler than SOAP and WSDL
- Based on HTTP standard commands (verbs):
  - **HEAD**: requests a response without document (only header).
  - **POST**: sends new document to the server.
  - **PUT**: replaces existing document.
  - **PATCH**: applies partial modifications to a document.
  - **DELETE**: removes document.

Principles

- Client-server:
  - Clear roles distinction.
- Stateless:
  - Each request from the client to the server must contain all the information necessary to understand and complete the request.
  - The server cannot take advantage of any previously stored context information on the server.
- Cacheable:
  - Answers can be cached by the client if server states it is possible.
- Resources:
  - Each document/ image / piece of data contains the data, its metadata (type), and the links to related resources.

Principles

- Identification of resources:
  - Each “resource” (data) must be uniquely identified, typically by an URL.
- Manipulation of resources through representations:
  - Resources should have uniform representations (typically JSON) in the server response. API consumers should use these representations to modify the resources state in the server.
- Self-descriptive messages:
  - Resource representation should carry enough information to describe how to process the message.
- Hypermedia as the engine of application state:
  - From an initial URL, clients can dynamically drive all other resources and interactions with the use of hyperlinks.

HTTP and REST

- HTTP is not mandated in REST but it's the norm.
- Uses all aspects of HTTP: verbs, headers, body contents, query-string parameters, response codes.
- **GET**: used to retrieve a resource, without modifying it.
  - Example: GET http://biblio.net/books?limit=20&page=5
  - Example: GET http://biblio.net/books/32323
- **POST**: used to create a new resource
  - Example: POST http://biblio.net/books (no ID, not known yet!)
- **PUT**: used to update a resource (completely)
  - Example: POST http://biblio.net/books/32323 (ID must be specified!)
- **PATCH**: used to update a resource (partially)
- **DELETE**: used to delete a resource

HTTP codes

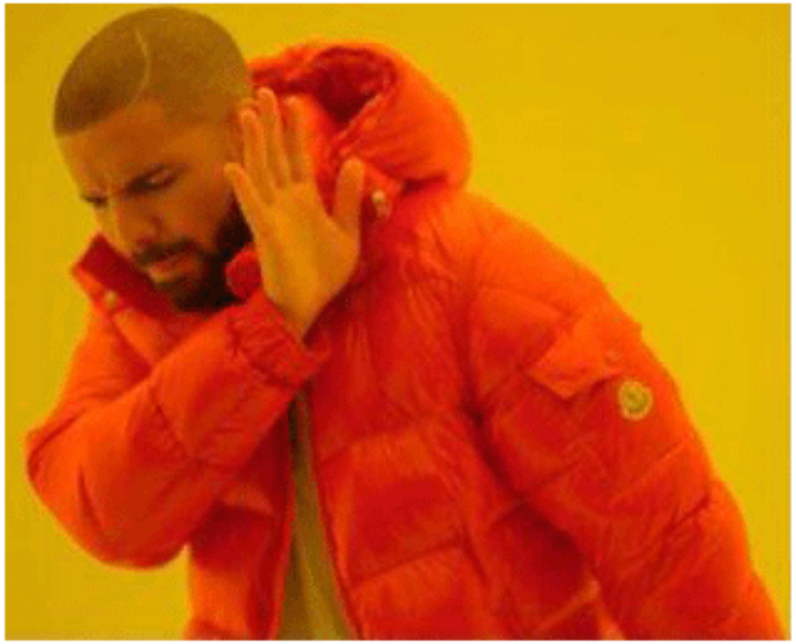
- HTTP response codes can be used to reply if an action was performed or if an error occurred.
- Examples:
  - 200: OK
  - 201: created (after a POST)
  - 301: moved permanently (use another URL)
  - 400: bad request (error in the request params)
  - 401: unauthorized (requires login)
  - 403: forbidden (does not have permission)
  - 404: resource does not exist
  - 500: internal server error (server crashed)

REST actions

HTTP Method	CRUD	Collection Resource (e.g. /users)	Single Resouce (e.g. /users/123)
POST	Create	201 (Created), 'Location' header with link to /users/{id} containing new ID	Avoid using POST on a single resource
GET	Read	200 (OK), list of users. Use pagination, sorting, and filtering to navigate big lists	200 (OK), single user. 404 (Not Found), if ID not found or invalid
PUT	Update/Replace	405 (Method not allowed), unless you want to update every resource in the entire collection of resource	200 (OK) or 204 (No Content). Use 404 (Not Found), if ID is not found or invalid
PATCH	Partial Update/Modify	405 (Method not allowed), unless you want to modify the collection itself	200 (OK) or 204 (No Content). Use 404 (Not Found), if ID is not found or invalid
DELETE	Delete	405 (Method not allowed), unless you want to delete the whole collection — use with caution	200 (OK). 404 (Not Found), if ID not found or invalid

SOAP vs REST

- Protocol:
  - SOAP uses HTTP
  - REST is independent, but everybody uses HTTP
- Format:
  - SOAP uses XML
  - REST can use any format, but JSON is more common
- State:
  - Both stateless
- HTTP verbs:
  - SOAP uses only POST, which is not cached by default
  - REST uses all verbs, including GET, which can be cached
- Schema:
  - SOAP uses WSDL
  - REST should be self descriptive
- Complexity:
  - SOAP can be very complex. XML has ambiguity and can become easily unreadable.
  - REST is simple and elegant. JSON is easier to read but has no standard schema.



```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
    soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
  <soap:Body xmlns:m="https://www.superglobamegacorp.example/stock">
    <m:GetStockPriceResponse>
      <m:StockSymbol>AAPL</m:Price>
      <m:Price>170.41</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>
</soap:Envelope>
```

```
{
  "stockSymbol": "AAPL",
  "price": 170.41
}
```