Network Layer Models and Application Layer

Unit 04 - Hands-On Networking - 2018

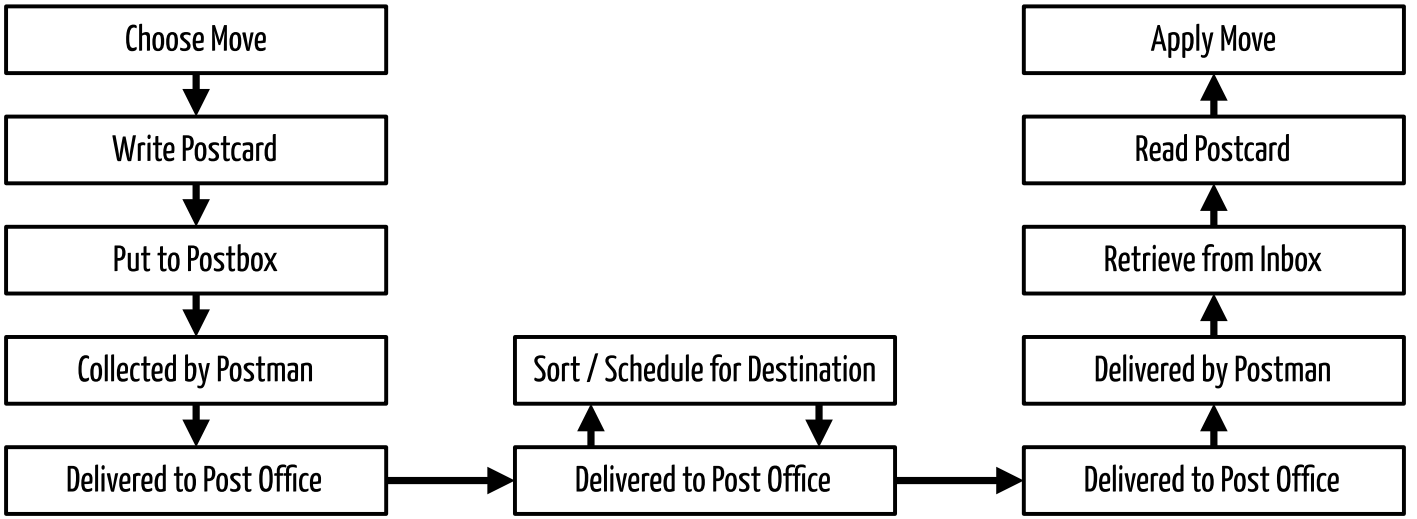
Prof. Dr.-Ing. Thorsten Herfet, Andreas Schmidt, Pablo Gil Pereira

Telecommunications Lab, Saarland Informatics Campus, 20th Feb. 2018

Recap

- **Information** is transmitted with **symbols** from an **alphabet**.
- **Services** are provided by **networks** using **communication systems**.
- **Train stations** are a good example for communication and have protocols.

Going Postal... with Chess



Derivations

- Post Delivery := Series of steps.
- Symmetry between start and end steps (not perfect here, but see later).
- Steps decoupled (different parties can be responsible for them).
- Traversing layers is well-defined.

What's in a Layer?

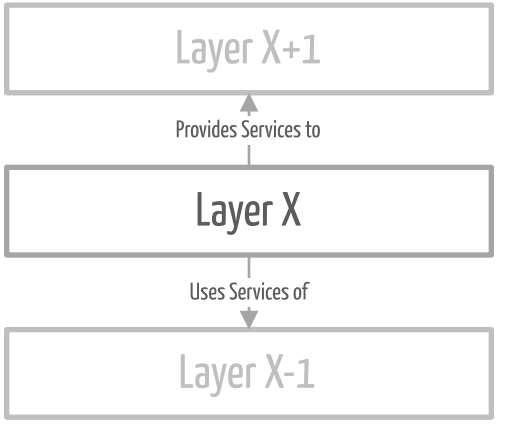
Definition: A component of a system, encapsulating functions with interfaces towards upper (north) and lower (south) layers.

↑ Northbound

- Layer implements a service it provides to the layer above.
- e.g. reliable transfer, point-to-point connectivity.

↓ Southbound

- Layer relies on services provided by the layer below.
- e.g. uses sessions established by lower layer protocols.



Layers are everywhere

- Application Software
  Database, Data Access, Domain Logic, User Interface
- Systems
  Hardware, Firmware / Device Drivers, Operating Systems, Middleware, Application Software
- Memory
  Registers, L1-3 Caches, Disk
- Cake…

Layer Pattern

Context: When dealing with complex, multi-faceted systems.

Implementation: Define layers by stating the services they *provide* and services they *use*.

Benefits:

- Layers can be developed independently.
- Layers can be replaced, as long as the API stays the same.
- Communication about system architecture gets easier.

Drawbacks:

- Can increase complexity and size of system.
- Might induce performance penalties due to layer-crossing methods.

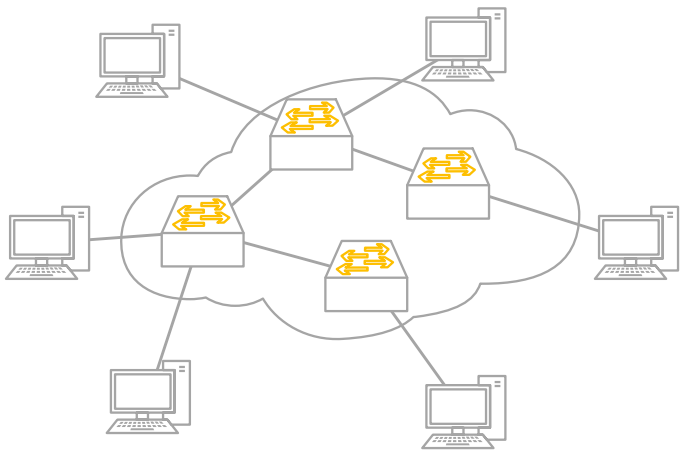
Layered Network Systems

Getting connected - with hierarchy!

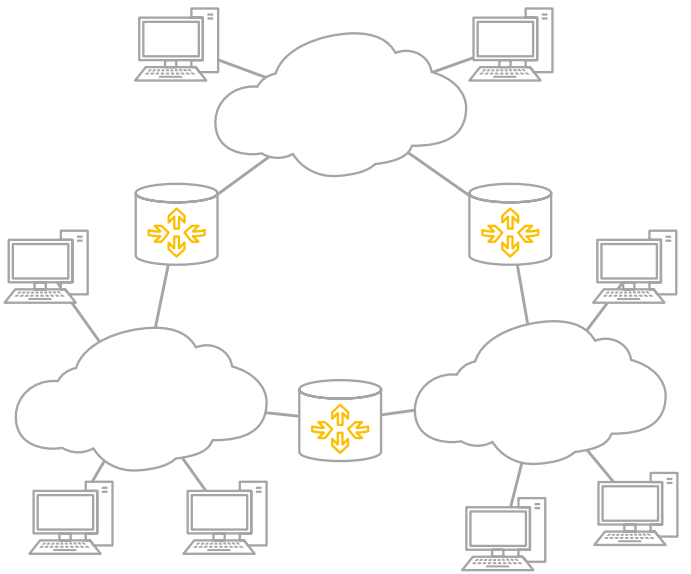
Direct Connection

Multiple Access

Switched Networks



Interconnected Networks



Open System Interconnection Model

Also known as the ISO/OSI model (ISO/IEC 7498-1, ITU-T X.200).

The 7 Layers:

- **1 - Physical**: Bits "on the wire", transmission schemes, media.
- **2 - Link**: Device-to-Device communication, shared media.
- **3 - Network**: Forwarding and routing of data.
- **4 - Transport**: Process-to-Process data transfer.
- **5 - Session**: Connection Establishment, State, Synchronization, Recovery.
- **6 - Presentation**: Data Composition, Encoding, Encryption, Compression.
- **7 - Application**: Services for users/machines.
- **8 - User** (sometimes refered to when talking about human errors, bonus layer)

TCP/IP Model

Similar to ISO/OSI model, but:

- More pragmatic and realistic.
- Combining layers.

Motivation

- Presentation and session were never created explicitly but are always embedded in application or transport layer protocols.

Naming

- Named after the two major protocols used, but makes not too much sense.
- Better: **Internet Protocol Suite**

Model Layers

- **1 - Network Access [1+2]**
- **2 - Internet [3]**
- **3 - Transport [4]**
- **4 - Application [5+6+7]**

Original OSI layers in [ ].

Topologies I

Bus

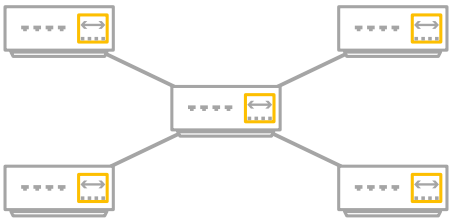
- Old system (still used in industrial networks and e.g. cars).
- Required coordination to avoid collisions.
- Disconnected wires can disrupt complete communication.

Ring

- Other traditional topology.
- Little number of connections.
- Reliable (one link can break down).
- Long distances to cover.

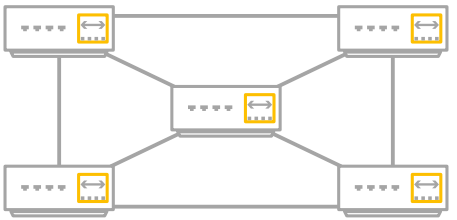
Topologies II

Star



- Modern topology for wired Ethernet networks.
- No need to handle collisions.
- Disconnects only affect small areas.
- Distances are short.

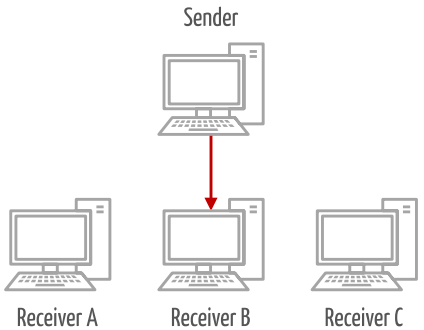
Mesh



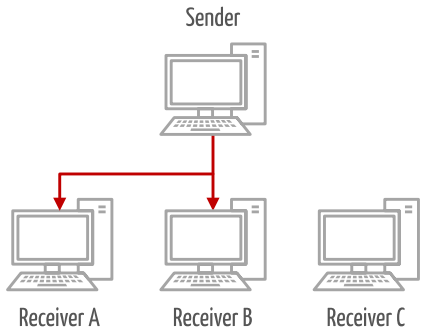
- Applied in data centers.
- Resilience is high.
- Distances are minimal.
- Wiring effort is high.

*-Cast

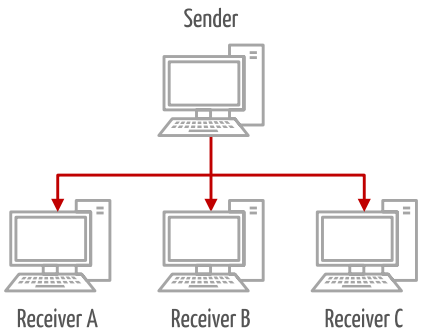
Unicast (*one receiver*)



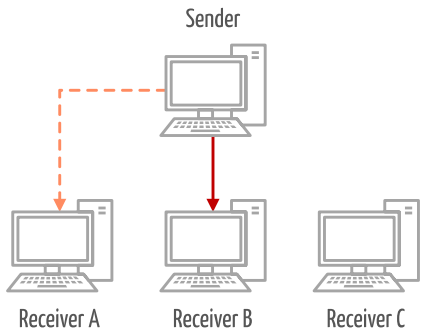
Multicast (*interested receivers*)



Broadcast (*all receivers*)



Anycast (*"closest" receiver*)



*AN (Some-Area-Network)

LAN (Local)

- Instances: Your Home Network.
- Components: SmartPhone, Desktop PC, Access Point, Router, Modem, Cordless Phone, ...
- Technologies: Ethernet, WLAN, DSL, PPP, DECT, ...

MAN (Metropolitan)

- Instances: Your University Network.
- Technologies: same as LAN.

CAN (Controller)

- Instances: Your car.
- Technologies: CANBUS, ...

WAN (Wide)

- Instances: Internet.
- Components: Autonomous Systems.
- Technologies: ATM, MPLS, ...

PAN (Personal)

- Components: SmartPhones, Fitness Tracker, Earbuds, ...
- Technologies: BlueTooth, ANT+, ...

BAN (Body)

- Components: Insulin Pump, Prostheses, Pacemaker, ...
- Technologies: Yet to be created and standardized (IEEE 802.15.6).

Getting Rid of Layers Again

☰ Layered-Design

- Good for design.
- Keeps systems maintainable.
- But: Might have performance drawbacks.

Cross-Layer Concepts

- Leave layers for design.
- Remove layers in implementation.
- Examples:
  - Deep Packet Inspection.
  - TCP stacks implemented in chips (TCP-in-silicon).
  - Web Browser taking into account whether WiFi is stable or not.

⚠ Be aware that...

> *"Premature optimization is the root of all evil."* - Tony Hoare

Standardization

Request for Comments (RFC) are published by the *Internet Engineering Task Force* (IETF) and specify *"the Internet"*.

RFC Index

Num Information

0001 **Host Software** S. Crocker [April 1969] (TXT = 21088) (Status: UNKNOWN) (Stream: Legacy) (DOI: 10.17487/RFC0001)
0002 **Host software** B. Duvall [April 1969] (TXT = 17145) (Status: UNKNOWN) (Stream: Legacy) (DOI: 10.17487/RFC0002)
0003 **Documentation conventions** S.D. Crocker [April 1969] (TXT = 2323) (Obsoleted-By RFC0010) (Status: UNKNOWN) (Stream: Legacy) (DOI: 10.17487/RFC0003)
0004 **Network timetable** E.B. Shapiro [March 1969] (TXT = 5933) (Status: UNKNOWN) (Stream: Legacy) (DOI: 10.17487/RFC0004)
0005 **Decode Encode Language (DEL)** J. Rulifson [June 1969] (TXT = 26408) (Status: UNKNOWN) (Stream: Legacy) (DOI: 10.17487/RFC0005)
0006 **Conversation with Bob Kahn** S.D. Crocker [April 1969] (TXT = 1568) (Status: UNKNOWN) (Stream: Legacy) (DOI: 10.17487/RFC0006)
0007 **Host-IMP interface** G. Deloche [May 1969] (TXT = 13408) (Status: UNKNOWN) (Stream: Legacy) (DOI: 10.17487/RFC0007)
0008 **ARPA Network Functional Specifications** G. Deloche [May 1969] (PDF = 750612) (Status: UNKNOWN) (Stream: Legacy) (DOI: 10.17487/RFC0008)
0009 **Host Software** G. Deloche [May 1969] (PDF = 722638) (Status: UNKNOWN) (Stream: Legacy) (DOI: 10.17487/RFC0009)
0010 **Documentation conventions** S.D. Crocker [July 1969] (TXT = 3348) (Obsoletes RFC0003) (Obsoleted-By RFC0016) (Updated-By RFC0024, RFC0027, RFC0030) (Status: UNKNOWN) (Stream: Legacy) (DOI: 10.17487/RFC0010)
0011 **Implementation of the Host - Host Software Procedures in GORDO** G. Deloche [August 1969] (TXT = 46971, PDF = 2186431) (Obsoleted-By RFC0033) (Status: UNKNOWN) (Stream: Legacy) (DOI: 10.17487/RFC0011)
0012 **IMP-Host interface flow diagrams** M. Wingfield [August 1969] (TXT = 177, PS = 1489750, PDF = 1163721) (Status: UNKNOWN) (Stream: Legacy) (DOI: 10.17487/RFC0012)
0013 **Zero Text Length EOF Message** V. Cerf [August 1969] (TXT = 1070) (Status: UNKNOWN) (Stream: Legacy) (DOI: 10.17487/RFC0013)
0014 Not Issued
0015 **Network subsystem for time sharing hosts** C.S. Carr [September 1969] (TXT = 10695) (Status: UNKNOWN) (Stream: Legacy) (DOI: 10.17487/RFC0015)
0016 **M.I.T** S. Crocker [August 1969] (TXT = 682) (Obsoletes RFC0010) (Obsoleted-By RFC0024) (Updated-By RFC0024, RFC0027, RFC0030) (Status: UNKNOWN) (Stream: Legacy) (DOI: 10.17487/RFC0016)
0017 **Some questions re: Host-IMP Protocol** J.E. Kreznar [August 1969] (TXT = 6065) (Status: UNKNOWN) (Stream: Legacy) (DOI: 10.17487/RFC0017)
0018 **IMP-IMP and HOST-HOST Control Links** V. Cerf [September 1969] (TXT = 634) (Status: UNKNOWN) (Stream: Legacy) (DOI: 10.17487/RFC0018)
0019 **Two protocol suggestions to reduce congestion at swap bound nodes** J.E. Kreznar [October 1969] (TXT = 3392) (Status: UNKNOWN) (Stream: Legacy) (DOI: 10.17487/RFC0019)
0020 **ASCII format for network interchange** V.G. Cerf [October 1969] (TXT = 18504, PDF = 197096) (Also STD0080) (Status: INTERNET STANDARD) (Stream: Legacy) (DOI: 10.17487/RFC0020)
0021 **Network meeting** V.G. Cerf [October 1969] (TXT = 2143) (Status: UNKNOWN) (Stream: Legacy) (DOI: 10.17487/RFC0021)
0022 **Host-host control message formats** V.G. Cerf [October 1969] (TXT = 4606) (Status: UNKNOWN) (Stream: Legacy) (DOI: 10.17487/RFC0022)
0023 **Transmission of Multiple Control Messages** G. Gregg [October 1969] (TXT = 690) (Status: UNKNOWN) (Stream: Legacy) (DOI: 10.17487/RFC0023)
0024 **Documentation Conventions** S.D. Crocker [November 1969] (TXT = 3460) (Obsoletes RFC0016) (Updates RFC0010, RFC0016) (Updated-By RFC0027, RFC0030) (Status: UNKNOWN) (Stream: Legacy) (DOI: 10.17487/RFC0024)
0025 **No High Link Numbers** S.D. Crocker [October 1969] (TXT = 479) (Status: UNKNOWN) (Stream: Legacy) (DOI: 10.17487/RFC0025)

10.17487/RFC7860)
7804 **Proxy Mobile IPv6 Extensions to Support Flow Mobility** CJ. Bernardos [May 2016] (TXT = 44225) (Updates RFC5213) (Status: PROPOSED STANDARD) (Stream: IETF, Area: int, WG: netext) (DOI: 10.17487/RFC7804)
7865 **Session Initiation Protocol (SIP) Recording Metadata** R. Ravindranath, P. Ravindran, P. Kyzivat [May 2016] (TXT = 67100) (Status: PROPOSED STANDARD) (Stream: IETF, Area: art, WG: siprec) (DOI: 10.17487/RFC7865)
7866 **Session Recording Protocol** L. Portman, H. Lum, C. Eckel, A. Johnston, A. Hutton [May 2016] (TXT = 104535) (Status: PROPOSED STANDARD) (Stream: IETF, Area: art, WG: siprec) (DOI: 10.17487/RFC7866)
7868 **Cisco's Enhanced Interior Gateway Routing Protocol (EIGRP)** D. Savage, J. Ng, S. Moore, D. Slice, P. Paluch, R. White [May 2016] (TXT = 179677) (Status: INFORMATIONAL) (Stream: INDEPENDENT) (DOI: 10.17487/RFC7868)
7869 **The "vnc" URI Scheme** D. Warden, I. Iordanov [May 2016] (TXT = 53002) (Status: INFORMATIONAL) (Stream: INDEPENDENT) (DOI: 10.17487/RFC7869)
7870 **Dual-Stack Lite (DS-Lite) Management Information Base (MIB) for Address Family Transition Routers (AFTRs)** Y. Fu, S. Jiang, J. Dong, Y. Chen [June 2016] (TXT = 53247) (Status: PROPOSED STANDARD) (Stream: IETF, Area: int, WG: softwire) (DOI: 10.17487/RFC7870)
7871 **Client Subnet in DNS Queries** C. Contavalli, W. van der Gaast, D. Lawrence, W. Kumari [May 2016] (TXT = 67651) (Status: INFORMATIONAL) (Stream: IETF, Area: ops, WG: dnsop) (DOI: 10.17487/RFC7871)
7873 **Domain Name System (DNS) Cookies** D. Eastlake 3rd, M. Andrews [May 2016] (TXT = 55356) (Status: PROPOSED STANDARD) (Stream: IETF, Area: ops, WG: dnsop) (DOI: 10.17487/RFC7873)
7874 **WebRTC Audio Codec and Processing Requirements** JM. Valin, C. Bran [May 2016] (TXT = 16160) (Status: PROPOSED STANDARD) (Stream: IETF, Area: art, WG: rtcweb) (DOI: 10.17487/RFC7874)
7875 **Additional WebRTC Audio Codecs for Interoperability** S. Proust [May 2016] (TXT = 26626) (Status: INFORMATIONAL) (Stream: IETF, Area: art, WG: rtcweb) (DOI: 10.17487/RFC7875)
7879 **DTLS-SRTP Handling in SIP Back-to-Back User Agents** R. Ravindranath, T. Reddy, G. Salgueiro, V. Pascual, P. Ravindran [May 2016] (TXT = 29648) (Status: PROPOSED STANDARD) (Stream: IETF, Area: art, WG: straw) (DOI: 10.17487/RFC7879)
7887 **Hierarchical Join/Prune Attributes** S. Venaas, J. Arango, I. Kouvelas [June 2016] (TXT = 17207) (Updates RFC5384) (Status: PROPOSED STANDARD) (Stream: IETF, Area: rtg, WG: pim) (DOI: 10.17487/RFC7887)
7888 **IMAP4 Non-synchronizing Literals** A. Melnikov [May 2016] (TXT = 17375) (Obsoletes RFC2088) (Status: PROPOSED STANDARD) (Stream: IETF, Area: art, WG: imapapnd) (DOI: 10.17487/RFC7888)
7889 **The IMAP APPENDLIMIT Extension** J. SrimushnamBoovaraghamoorthy, N. Bisht [May 2016] (TXT = 13801) (Status: PROPOSED STANDARD) (Stream: IETF, Area: art, WG: imapapnd) (DOI: 10.17487/RFC7889)
7891 **Explicit Reverse Path Forwarding (RPF) Vector** J. Asghar, IJ. Wijnands, S. Krishnaswamy, A. Karan, V. Arya [June 2016] (TXT = 19067) (Status: PROPOSED STANDARD) (Stream: IETF, Area: rtg, WG: pim) (DOI: 10.17487/RFC7891)
7892 **IANA Allocation Procedures for the GMPLS OTN Signal Type Registry** Z. Ali, A. Bonfanti, M. Hartley, F. Zhang [May 2016] (TXT = 7019) (Updates RFC7139) (Status: PROPOSED STANDARD) (Stream: IETF, Area: rtg, WG: ccamp) (DOI: 10.17487/RFC7892)
7894 **Alternative Challenge Password Attributes for Enrollment over Secure Transport** M. Pritikin, C. Wallace [June 2016] (TXT = 19712) (Status: PROPOSED STANDARD) (Stream: IETF, WG: NON WORKING GROUP) (DOI: 10.17487/RFC7894)
7896 **Update to the Include Route Object (IRO) Specification in the Path Computation Element Communication Protocol (PCEP)** D. Dhody [June 2016] (TXT = 9960) (Updates RFC5440) (Status: PROPOSED STANDARD) (Stream: IETF, Area: rtg, WG: pce) (DOI: 10.17487/RFC7896)
7897 **Domain Subobjects for the Path Computation Element Communication Protocol (PCEP)** D. Dhody, U. Palle, R. Casellas [June 2016] (TXT = 71770) (Status: EXPERIMENTAL) (Stream: IETF, Area: rtg, WG: pce) (DOI: 10.17487/RFC7897)
7898 **Domain Subobjects for Resource Reservation Protocol - Traffic Engineering (RSVP-TE)** D. Dhody, U. Palle, V. Kondreddy, R. Casellas [June 2016] (TXT = 34962) (Status: EXPERIMENTAL) (Stream: IETF, Area: rtg, WG: teas) (DOI: 10.17487/RFC7898)
7902 **Registry and Extensions for P-Multicast Service Interface Tunnel Attribute Flags** E. Rosen, T. Morin [June 2016] (TXT = 14332) (Updates RFC6514) (Status: PROPOSED STANDARD) (Stream: IETF, Area: rtg, WG: bess) (DOI: 10.17487/RFC7902)
7910 **Interoperability between the Virtual Router Redundancy Protocol and PIM** W. Zhou [June 2016] (TXT = 14255) (Status: INFORMATIONAL) (Stream: INDEPENDENT) (DOI: 10.17487/RFC7910)

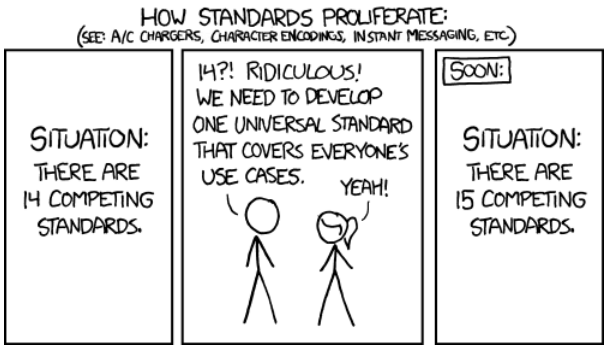
Interesting, but not particularly relevant ones: RFC1606, RFC1882, RFC2549 RFC5841, RFC6921, RFC7169, RFC7511, RFC8135

Standardization | Down-to-Earth

- While IETF specifies the high layers of the "Internet" communication, there is another very important standardization body:

IEEE The world's largest professional organization for the advancement of technology

- **Institute of Electrical and Electronics Engineers** (IEEE, pronounced "I triple E")
- Protocols involving Physical, Link and Medium Access technologies are usually standardized by IEEE.



The Application Layer

HON

Applications - Examples

- Electronic Mail (Email)
- Web (Wikipedia, News Sites, ...)
- Text Messaging (WhatsApp, Signal, Telegram, ...)
- Remote Login (Telnet, SSH, ...)
- P2P File Sharing (BitTorrent, ...)
- Multi-User Network Games (World of Warcraft, Pokémon Go, ...)
- Streaming Stored Video (YouTube, Netflix, Vimeo, ...)
- Search (Google, Bing, ...)
- Voice over IP (Skype, ...)
- Video Conferencing (Google Hangouts, Adobe Connect, ...)
- Social Networking (Facebook, Twitter, ...)
- Help Platforms (StackExchange, ...)
- Cloud File Store and Share (Dropbox, Google Drive, OneDrive, ...)
- ...

HON

Application Development - Motivation

◎ **Goal:** Create the next textual messaging service *chattr*.

❓ What steps are required to provide this to users (and make profit)?

- Invent the service (including business model) and its functionality (*think*).
- Create a software system that provides your service (*code*).
- Create a user interface so customers can consume your service (*design*).
- Create a network of hardware end-devices and relaying stations (*build*).

But: The last step building this network is a) expensive and b) time-consuming.

- Traditional services were single purpose and required this (e.g. telephone).
- Today: Internet provides an established network for communication services.
  - Many requirements of communication applications can be fulfilled using existing protocols and systems.
  - Deploying an application only requires a) hosting the server and b) providing the app to users.

Application Development

⚙ Details

- Applications run on end-systems (desktops, mobiles, TVs, ...).
- Communicate using the network and its transmission capabilities.

👍 Benefits

- Network-core devices are unchanged.
- Applications are not run on the core devices (they can stay "dumb").
- Application development fast, as only end-systems have to be changed when deploying.

chattr Example

- Service: Text messaging.
- Provision Process:
  - Create a server that stores and forwards messages.
  - Deploy server software e.g. in Amazon Cloud or similar.
  - Create an app for sending and receiving messages.
  - Distribute via App Store, Play Store, etc.

Architectures

Services can be implemented using two different approaches:

Client-Server (CS)

- Specific roles assigned to hosts.
- Always-on hosts provide services that are consumed by sporadically-on hosts.
- Examples:
  - Web (HTTP)
  - File Transfer (FTP)

Peer-To-Peer (P2P)

- All hosts are equal.
- Sporadically-on hosts communicate, sometimes using deferred relaying of messages and data.
- Examples:
  - Multimedia Communication (WebRTC)
  - File Sharing (BitTorrent)
  - Load Sharing (DropBox LAN Sync, Windows Update)

Client / Server Pattern

Context:

- Users consume a service *sporadically*.
- Information is stored in a *central location* that is *always available*.

Implementation:

- **Server**: Always-on host with permanent address.
- **Client**: Initiates contact with server and has a dynamic address. No direct client-to-client communication, even for interactive applications.

Benefits:

- Service always* accessible.
- Simple to build.
- Well-established (works with firewalls, NATs etc.)

(* Excluding crashes, maintenance, ...)

Drawbacks:

- Servers are single point of failure.
- Data might take detours.
  Call your neighbor in Saarbrücken via a server in Frankfurt.

Peer-To-Peer Pattern

Context:

- Users communicate with each other *directly* and hosts provide the service.
- Information is stored on end-hosts.

Implementation:

- *No servers*, as all hosts are client and server at the same time.
- *Arbitrary* end-systems communicate with each other directly.
- Peers *intermittently connect* and *change IP addresses*.
- Not all participants communicate with each other *at the same time*.

Benefits:

- Better performance (sometimes), because connections are shorter.
- More resilient, as peers can take over for others.

Drawbacks:

- NATs etc. may still require servers for NAT-traversal.
- Complex management of data flow (e.g. distributed hash tables etc.).
- Public considers this as bad.
- Legal issues (consume/provide at same time).

Communicating Processes

Process: Program running within a host (i.e. operating system environment).

- Within the same host, processes communicate using inter-process communication (e.g. UNIX sockets, Pipes, ...)
- Processes on different hosts communicate using messages.

Client Process initates communication.

Server Process waits to be contacted.

Hosts in P2P architectures have client and server processes simultaneously.

Application Requirements

Loss-Tolerancy

Options:

1. Require 100% reliability (e.g. transactions, file transfer).
2. Tolerate certain amount of loss (e.g. video).

Timing

Options:

1. Timely delivery irrelevant, eventual delivery required (file transfer).
2. Upper bound on latency to be effective (telephony, interactive games, video-stream).

Throughput

Options:

1. Minimum data rate required for effective operation (e.g. multimedia).
2. Use whatever data rate is available aka "elastic" (e.g. file upload).

Security

- Encryption, integritiy, authenticity, ...
- Covered later.

Quiz: Application Requirements

Application	Data Loss	Throughput	Time Sensitive
File Transfer	???	???	???
E-Mail	???	???	???
Web Pages	???	???	???
Real-Time Audio/Video	???	???	???
Stored Audio/Video	???	???	???
Interactive Games	???	???	???
Text Messaging	???	???	???

Quiz: Application Requirements

Application	Data Loss	Throughput	Time Sensitive
File Transfer	No Loss	Elastic	No
E-Mail	No Loss	Elastic	No
Web Pages	No Loss	Elastic	No
Real-Time Audio/Video	Loss-Tolerant	Audio: 5kbps - 1Mbps Video: 10kbps - 5 Mbps	Yes (100ms)
Stored Audio/Video	Loss-Tolerant	see above	Yes (few secs)
Interactive Games	Loss-Tolerant	Few kbps and more	Yes (100ms)
Text Messaging	No Loss	Elastic	Depends

HTTP and the Web

Web | Motivation

Goal: Share knowledge and documents between scientists.

Universal Libraries

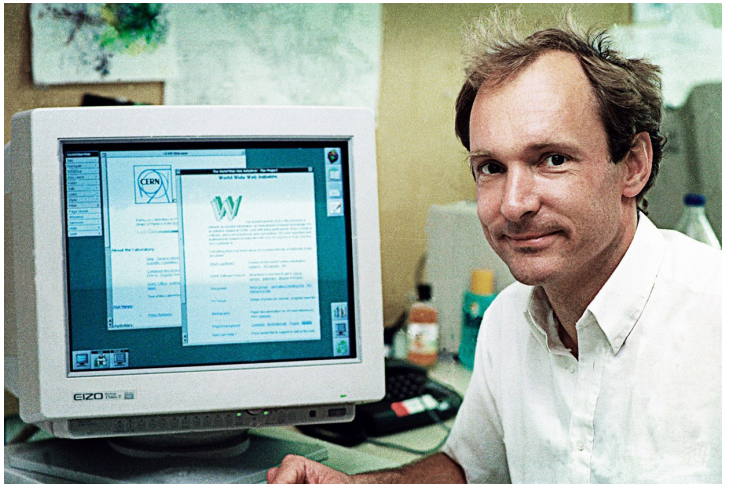
- Library of Alexandria as the first major place to host nearly all knowledge in the world (estab. BC).
- Mundaneum (estab. ~1900).

Problems

- Books have to be retrieved.
- People travel to see the books.
- Books are not really alterable.

Fast-Forward (Tim Berners-Lee, 1989)

Proposal of an **information management system** and *implementation* of the first **client**, **server** and protocol **HTTP**.



Web | Definition and Components

The **Web** is a service *(often confused with the Internet or used interchangingly)*.

- Internet is a *Network*.
- Web is a *Service*.

Web Pages consist of objects:

- Hyper-Text Markup Language (HTML): content / structure
- Cascading Style Sheets (CSS): style / layout
- JavaScript (JS): logic, interaction
- JPEG, PNG, SVG: graphics

Web Applications become more and more like desktop applications that run in your browser (consider Chromium OS).

Source

Home Page is the starting page of a company or organization on the web *(often mistakenly used for the complete web site, esp. in Germany)*.

+++ "Haben sich auf einer Blattform im Netz kennengelernt: Spinnenpärchen frisch verliebt", +++ der-postillion.com

Uniform Resource Locator (URL)

Syntax:

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

Example:

```
https://cms.nt.uni-saarland.de/system/theme/TcCms/img/logo.png
```

- scheme: `https`
- user/password: omitted
- host: `cms.nt.uni-saarland.de`
- port: default for scheme, hence 443
- path: `system/theme/TcCms/img/logo.png`
- query, fragment: omitted

Special form of *Uniform Resource Identifier (URI)* (RFC3986).

HON

Hypertext Transfer Protocol (HTTP)

The application layer protocol for the **Web** service.

Client: Web Browser

- Requests resources and renders them.
- Nowadays a platform of its own for applications as known from the desktop.

Server: Web Server

- Providing objects upon request.
- Uses other components such as database etc. so that requests have a persistent effect.

Communcation Process

- Request
  Client composes a request in form a URL, a verb and possibly additional data.
- Response
  Server interprets this and answers with status code and possibly additional data.

Instances of *Request / Response* and *Client / Server* patterns.

HTTP | Browsing the Web...

Open browser and navigate to `http://www.nt.uni-saarland.de`.

Request

```
GET / HTTP/1.1\r\n
Host: www.nt.uni-saarland.de\r\n
Connection: keep-alive\r\n
Cache-Control: max-age=0\r\n
Upgrade-Insecure-Requests: 1\r\n
User-Agent: Mozilla/5.0 (X11; Linux x86_64) \r\n
Accept: text/html,application/xhtml+xml,... \r\n
Accept-Encoding: gzip, deflate, sdch\r\n
Accept-Language: en-US,en;q=0.8\r\n
\r\n
```

Response

```
HTTP/1.1 200 OK\r\n
Date: Fri, 26 Aug 2016 08:17:09 GMT\r\n
Server: Apache\r\n
Set-Cookie: path=/\r\n
Expires: 0\r\n
Last-Modified: Fri, 26 Aug 2016 08:17:09 GMT\r\n
Cache-Control: no-cache, must-revalidate\r\n
Pragma: no-cache\r\n
Max-Age=51526307; path=/\r\n
Vary: Accept-Encoding\r\n
Content-Encoding: gzip\r\n
Content-Length: 2303\r\n
Keep-Alive: timeout=5, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=utf-8\r\n
\r\n
```

HTTP | Headers and Methods

■ Verbs / Methods

- `GET`: retrieve a resource
- `PUT`: request the enclosed data to be stored under the given URL
- `DELETE`: request the removal of the resource under the given URL
- `POST`: accept the data sent in the body (annotate existing resources, add a message to a thread)
- Others: `HEAD`, `CONNECT`, `TRACE`.

↱ Request Headers

- `Accept`: Content-type that is accepted (e.g. `text/plain`)
- `Content-Type`: MIME type of request body (e.g. `application/json`)
- `Host`: Server domain name of server and TCP port number (mandatory)
- …

↰ Response Headers

- `Content-Type`: MIME type of response body
- `Expires`: Date/time the response is considered stale
- `Last-Modified`: Date the resource got last modified
- …

HTTP | Response Status Codes

Servers reply with three digit codes (RFC7231). First digit (1-5) indicates a group:

- 1xx Informational: Request received, continuing process.
  100: Continue; 101 Switching Protocols.
- 2xx Success: Request was received, accepted, and processed successfully.
  200: OK; 201: Created
- 3xx Redirection: Requester must take additional actions to complete request.
  301: Moved Permanently; 304: Not Modified
- 4xx Client Error: Requester must have erred.
  400: Bad Request; 403: Forbidden; 404: Not Found; 418: I'm a teapot
- 5xx Server Error: Server failed to fulfill an apparently valid request.
  500: Internal Server Error; 503: Service Unavailable

Stateless Pattern

Context:

- Being stateful is complex (for protocols and others).
- Maintaining state is difficult when faced with crashes etc.

Implementation:

- Consider ACID principle as common in database design.
  - **A**tomic: Operations itself are considered as one atomic unit.
  - **C**onsistent: System is consistent before and after each transaction.
  - **I**solation: Concurrent transactions don't interfere.
  - **D**urability: Changes will remain (even after power loss, crash, etc.).

Benefits:

- Transactions are short-term.

Drawbacks:

- State has to be kept somewhere else.

HTTP | Keeping State

Without connection state, how can multiple req/resp pairs be related to each other?

Answer: Keep state, but at the client (and a bit of it on the server).

Implementation

1. Cookie header line in HTTP response message (`set-cookie`).
2. Cookie header line in following HTTP request messages.
3. Cookie contents kept on client, managed by the browser.
4. Back-end database at the Web site for lookup of cookie-related data.

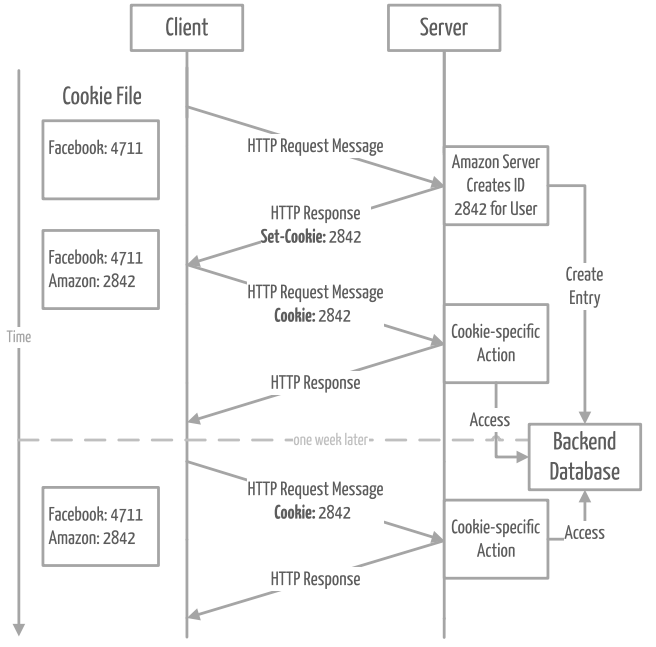
Source

Cookie Usages: Authorization, Shopping Carts, Recommendations and User session state.

HTTP | Cookies down to the Crumb

Example

- Bob always accesses Internet from same PC and identical browser.
- Visits `amazon.com` for the first time:
  - No cookie present in request.
  - Request related to this specific domain.
- Response contains the web page.
  - Cookie is set and contains a unique ID chosen by Amazon.
  - Amazon has created an entry in the backend database with this ID.
- Subsequent requests:
  - Carry the cookie.
  - Can be used by Amazon to track actions.



Cookies used for tracking users, their behaviour and preferences.

Sending what is known... again!

Problems:

- Static parts of web sites (styles, Javascript code) do not change frequently, but are sent with every request to the web page.
- Users inside a local network (e.g. university) might access the same websites (e.g. Facebook).

What to do about it?

Caching

Cache Pattern

Context:

- Accessing the same content again and again.
- Generating / retrieving the content is expensive.

Implementation:

- Provide a (usually) fixed size storage for cache items.
- After retrieving some content for the first time, store it in the cache.
- Invalidate the cache after some time or some event happened.
- Release / free cache space of "outdated" items, so that others can be cached.

Benefits:

- Saves computation time (for generated results).
- Saves transmission time (retrieve it from a close place).
- Saves bandwidth (no need to send it over links).

Drawbacks:

- Content might be outdated when retrieved from cache.
- Cache dimensioning and invalidation are problems on their own.

Web | Caching Content

Browser Cache

- Browsers keep copies of web objects.
- Servers can indicate if objects should be cached or not.
- Before downloading an object again, ask server for presence of changes.

Content-Delivery Networks (CDN)

- Used by all major content providers (Google, Facebook, NetFlix, ...).
- Provided by companies of their own (Akamai, Amazon, MS Azure, ...).
- Approach similar to web proxy, but not in local networks.

Web Proxy

- Company networks use them.
- HTTP connections are made through server (sees all files).
- Caches HTML, CSS, JS files.
- First accesses will go to destination.
- Subsequent accesses are served out of the cache.
- Accesses are faster and independent of Internet conditions.

Trend: Fog Computing

Enables caching data at *Central Office* (wired) or *Base Stations* (wireless).

HTTP | Connections

HTTP runs on TCP, so connection establishment and teardown is required.

Non-Persistent Connections

- After retrieving one object, the TCP sockets is discarded.
- Downloading multiple objects requires multiple connections.

Problem: Every item is sent with one additional RTT for connection establishment.

Persistent Connections

- One (in modern browser multiple) connection(s) established to the server.
- Requests executed over one TCP connection, not closing it afterwards.

Benefit: Each request/response takes only one RTT plus payload transmission time.

Quiz

🌐 Scenario

- Website with one HTML file (size: 10kB) and 3 JPG images (size: 82kB each)
- Forward-Trip Time for both directions: 1s
- Throughput: 1Mbps
- Service time per request: 0s

❓ How long does it take from navigating to the page until all content is loaded?

Conversions: $1Mbps = 1024kbps = 128kBps; RTT = 2 \cdot FTT = 2s$

🔗 Non-Persistent Connections

- 4 Requests:
  - $4 \cdot RTT$ for TCP connection.
  - $4 \cdot RTT$ for request/response.
- 4 Files: $10kB + 3 \cdot 82kB = 256kB$
  - $256kB / 128kBps = 2s$
- Result: $8 \cdot 2s + 2s = 18s$

🔗 Persistent Connections

- 4 Requests:
  - $1 \cdot RTT$ for TCP connection.
  - $4 \cdot RTT$ for request/response.
- 4 Files: see left.
- Result: $5 \cdot 2s + 2s = 12s$
- 33% less time than non-persistent.

HTTP | Future Directions

WebSockets

- Motivation:
  - HTTP implements Request/Response pattern.
  - Sending data from server to client requires prior request.
  - No means to efficiently push data from server to client (long-polling, busy poll etc. were not good).
- Implementation:
  - Full-duplex channel between web servers and browsers.
  - TCP-based, but features HTTP handshake.
  - Standardized in RFC6455.

HTTP 2.0

- Initial development driven by Google (SPDY protocol).
- Goals:
  - Compatibility with HTTP 1.1.
  - Latency decrease!
    - Compress headers.
    - Push data from server to client (proactively).
    - Pipeline requests.
    - Request multiplexing.
- Standardized in RFC7540. Published in May 2015.

Google QUIC

- Naively: HTTP over UDP. Latency counts!

Transport Layer Security (TLS)

HTTP | Security Considerations

Pure HTTP is insecure

- No encryption.
  Data sent in clear, can be eavesdropped. No confidentiality.
- No authentication.
  Requests and responses can be forged or replayed.
- No integrity.
  Everyone can tamper with data.

Historically: Security was no concern. Exchanging research results was nothing to be attacked.

Today: e.g. Online Banking.

Security Goals

- Confidentiality.
  Encrypt requests and responses.
- Authenticity.
  Sign requests and responses, agree on keys and roles. Ensure that client is talking to the right server.
- Integrity.
  Messages are signed, so that tamperings are detected.
- Usability.
  Provide out-of-the-box solution on ***transport layer*** so app developers don't have to create security measures on their own.

SSL, TLS and HTTPS

History

- First solution developed by Netscape and called Secure Sockets Layer (SSL).
- SSL served as a basis to the Transport Layer Security (TLS) standard.
- TLS alone usually not provided, instead combined with application.

HTTPS

- HTTP nowadays uses TLS instead of TCP directly, turning it into HTTPS.
- HTTPS uses port 443 instead of 80.
- Servers are equipped with certificates to prove their identity.
- Web browsers are shipped with root certificates to check the trustworthiness of server certificates.

HTTPS does not secure against every attack. Be skeptic and check your browser's additional info if in doubt.

TLS | Components

Handshake Protocol

- Prove identities.
- Agree on algorithms, keys and other parameters.
- Compute a master key for the connection using shared / exchanged secrets.

Record Protocol

- Manipulates messages coming from the application layer.
- Fragment into blocks convenient for further manipulations.
- Optional compressing.
- Protect integrity using HMAC (e.g. MD5, SHA2, ...).
- Encrypt using symmetric-key cipher (DES, AES, ...).
- Pass to transport-layer (e.g. TCP).

Wrap-Up

Take-Home Messages

- **Layers** simplify system design, are everywhere, but come at a cost.
- **RFCs** specify the Internet and more.
- **Application Layer** is where services to the user are implemented.
- **Client-Server** and **Peer-to-Peer** are the two architectures for app communication.
- Applications have **distinct requirements** regarding multiple facets.
- **HTTP** and **the Web** allow sharing documents and providing install-free applications.
- **TLS** provides a reusable solution for applying security on packets on transport.

Further Reading

- Kurose-Ross "Computer Networking"
  - Sec. 2.1 (App Layer)
  - Sec. 2.2 (HTTP)
  - Sec. 8.6 (SSL)