**CACS Part ONE**

Version Control Systems (VCSs)

* Synonyms: Version/source code/revision control system, source code management (VCS, SCM) (Torvalds calls ist SCM)
  + Collaboration on repository of documents
  + Each document going through various versions/revisions
  + Each document improved by various authors (eg Linux kernel)

Major VCS features

* keeps track of **history** 
  + Who changed what why when?
  + Restore/inspect old versions if necessary
* VCS supports **merging** of versions into **unified/integrated** version
  + Integrate intermediate versions of single file with changes by multiple authors
* Copying of files is **obsolete** with VCSs -> just use tags
* Various VCS exist

Git - A Decentralized VCS

* By Linus Torvalds
* Every author has **own copy** of all documents and their history
* Supports **offline** work without server connectivity
* Of course, collaboration requires network connectivity
* Distributed trust/control/visibility/surveillance

Key Terms

* Fork/clone repository: Create copy of repository
* Clone: Remote Copy on your Machine
* Fork: Copy of the repository
* Commit: Make some/all changes permanent
* Push: Publish commits to the repository
* Fetch(pull): Receive commits from the remote repository
* Git repositories are public, but not everyone is allowed to change files
  + Possibility –> fork

Branches

* Alternative Versions you can commit to
* Possible to Develop without disturbing by others

Merge:

* Combine changes into one branch
* Example: fixes are merged into stable branch (merge commit)

Git Codes

* "git intit" -> make a new git repository at this place
* git clone [address] -> clones a repository from a page
* git status shows potential changes
* Git add [..] adds to staged for commitment
* Git checkout discharges changes
* Git diff hows changes, which are not in staged mode
* git diff –cached -> Output shows difference between staged changes and last committed version
* Git commit –m "What I changed" - > commits change
* Git log shows commit history

Undoing Changes

* Undo premature commit that only exists **locally** -> ÜBEN
  + git reset HEAD~
  + (**Don’t** do this for commits that exist in remote places)
* Execute git status and git log
* Undo git add with git reset

Restore committed version

* Execute git checkout -- <file>
* **Warning**: Local changes are **lost**
* Stashing Changes <https://www.git-scm.com/docs/git-stash>
* Execute git stash
* If you performed git checkout ... on previous slide, change some file first
* Execute git status and find yourself on previous commit
* Apply saved changes
* Possibly on different branch or after git pull
* Execute git stash apply
* May lead to conflicts, to be resolved manually

Branching

* = work on different branch
* git checkout -b testbranch
* -> "-b" creates the new branch, the checkout switches to the new branch
* Without-b you just switch

Remotes

* git remote –v -> shows the url of the server
* Show remote repositories, whose changes you track
  + When you grab a remote repository, the remote repository is called origin
  + By git init, there is no remote repository
* Git upstream leads to pushing into the previous forked repo

Approach:

* Set up source project as remote upstream:
  + git remote add upstream <HTTPS-URL of source project>
* Fetch upstream: git fetch upstream
* Integrate upstream/master into your master, maybe with [rebase](https://oer.gitlab.io/oer-courses/cacs/Git-Introduction.html" \l "slide-merge-vs-rebase):
  + git checkout master
  + git rebase upstream/master
* Push updated master to your fork: git push

Merge vs Rebase

Suppose you created branch for new feature and committed on that branch; in the meantime, somebody else committed to master

Merge creates **new** commit to combine both branches

* Including all commits
* Keeping parallel history

Rebase rewrites feature branch on master

Difference: Puts the project into one state.

Git Workflows

In Order to collaborate, the team needs to agree on Git Workflow (the way they work), see <https://www.atlassian.com/git/tutorials/comparing-workflows>

**Distributed Systems**

Communication and Collaboration

* Communication often takes place via Internet
* Collaboration is often supported by using Internet technologies
  + ERP, CRM, e-learning systems
  + File Sharing
  + Programming, eg. Git
* Distributed Systems are ubiquity (urbiqär)
  + Decentralized, heterogeneous, evolving
  + Variety pf physical networks

Distributed Systems - Definition

“a collection of independent computers that appears to its users as a single coherent system”

Internet vs Web

* = network of networks
  + Connectivity for heterogeneous devices over various protocols
* Web is an application using the Internet
  + Clients and servers talk HTTP (another protocol) over TCP/IP -> GET requests
* Internet and Web are and contain DSs

**Technical Challenges**

* No shared memory but message passing
* Concurrency
* Autonomy and heterogeneity
* Neither global clock nor global state
* Independent failures
* Hostile environment, safety vs security

**Goals of DS:**

* Make resources accessible
* Openness, accepted standards → transfer data from anywhere to everywhere (no boundaries)
* Transparency → appears as one system
* interoperability
* Scalability → numerical (users), geographical, administrative (admin parties involved)

**Transparency** = Invisibility (hide complexity)

* **Location** → no need to know the physical location of data centers
* **Migration** → location of objects eg, when migrating between servers
* **Replication** → no need to know, if objects are replicated
* **Failure** → hidden from clients

#### Scalability

Scale up: More RAM, CPU

Dimensions:

* Numerical (number of users)
* Geographical (distance over the system is scattered)
* Administrative (number of controlling people

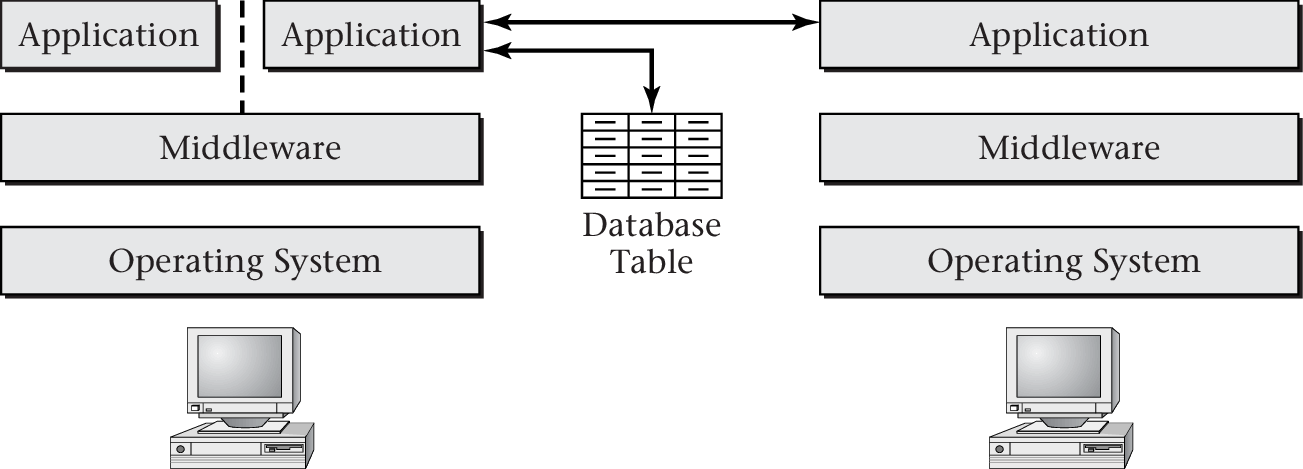
Scalability Techniques

* Replication ( =copy on multiple machines/nodes)
  + Increased availability
  + Reduced latency
  + Increased throughput (balanced load)
  + Challenge: keep sync
* Caching (= to save (intermediate) results close to client)
  + Reduced load on server
  + Increased availability, throughput and reduced latency
* Partitioning (= spread data or services among multiple machines)
  + Each node responsible for subset
  + Reduced availability
  + Reduced latency and increased throughput

System Models

* DS share important properties
* Types of models
  + Physical models
    - Computers, devices, their interconnections
  + Architectual models
    - Entities and their Roles (ERM)
  + Fundamental models
    - Interaction, consistency, security

Layering

* abstractions to hide complexity
* problem at hand is decomposed into manageable components
* Design becomes (more) modular
* 

Middleware

* Software layer for distributed systems @PASCAL
* Hide heterogeneity

Layered Network Models

* Upcoming session: Layering as core mechanism of network models

Clocks

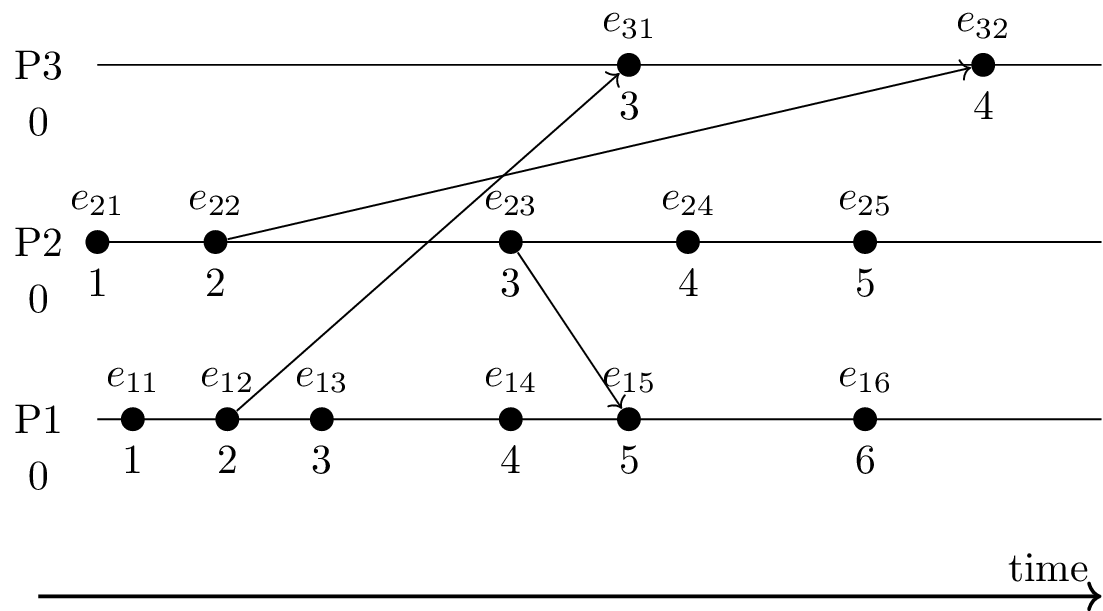
* Every computer with own internal clock
* Every internal clock with own clock drift rate
  + Clocks vary significantly unless corrections are applied
* Different correction approaches
* Two extremes
  + Asynchronous
    - Nothing is known about relative timing
  + Synchronous
    - Time is under control, different processes can proceed in lock-step
* Depending on
  + relative speeds of processes
  + time delay in delivering messages
  + clock drift

Logical Time (Lamport)

* = ordered via “happened before” relation
  + Without reference to physical clock

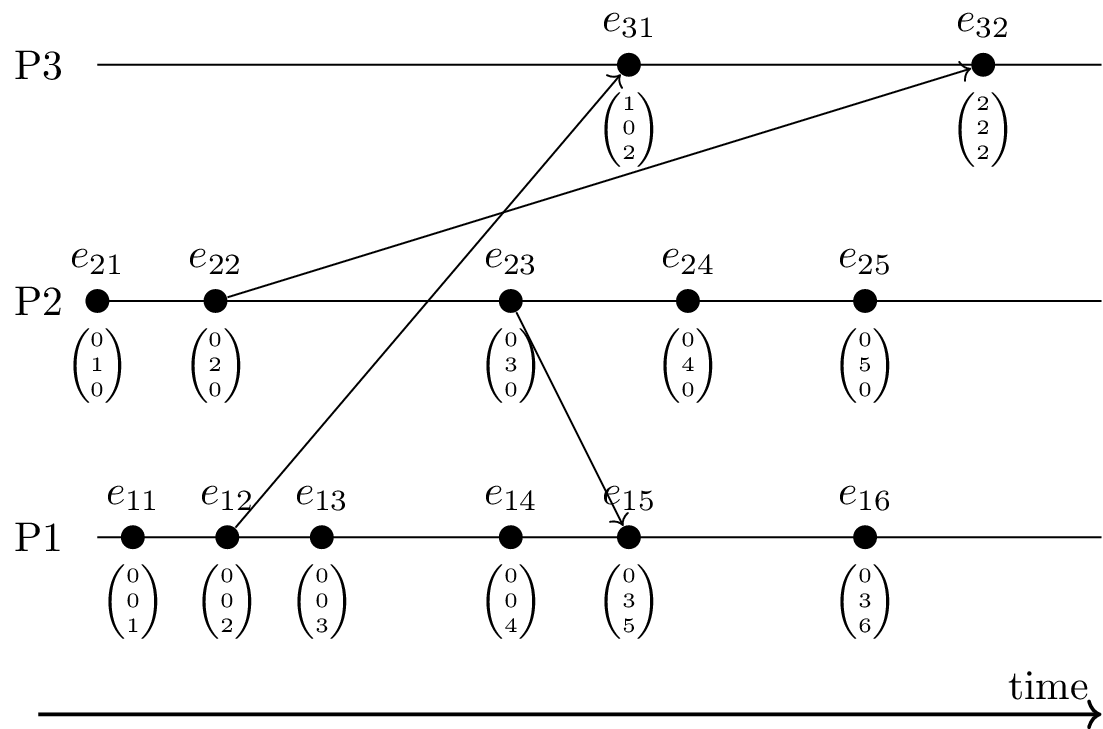
Lamport Timestamps

* Each process starts with logical clock (initially 0) – incremented for each event
* Diagonal arrows represent messages



* If *e*→*f* then *l*(*e*)<*l*(*f*)
  + E.g., *e*11→*e*32 and 1 = *l*(*e*11)<*l*(*e*32) = 4

Vector Clocks



* Clock Timestamp = vector of logical timestamps
* One component per location -> Incremented locally
* Conflicts/concurrency visible: incomparable vectors
  + Actions at different locations without taking all previous events into account (e.g., e23 vs e14; merged at e15)

Constituency

Example YouTube-Servers:

* Since there are multiple YouTube-Servers - each counts separately
* When no updates occur in some time – syncing – same level of information are serially processed
* Consistency requires consensus in DS
  + Integrity
  + Serializability
* Byzantine Generals – same start but no sync - problem

The Internet

*Communication and Collaboration*

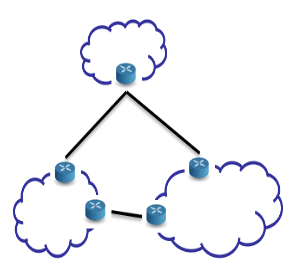
* Communication frequently takes place via the **Internet**
  + Telephony
  + Instant messaging
  + E-Mail
  + Social networks
* Collaboration frequently supported by tools using **Internet** technologies
  + All of the above means for communication
  + ERP, CRM, e-learning systems
  + File sharing: Sciebo, etherpad, etc.
  + Programming (which subsumes file sharing): Git, subversion, etc.
* All of the above are instances of **DSs**

*1.3.2 Ubiquity of the Internet*

* The Internet is everywhere
  + Decentralized, heterogeneous, evolving
  + Variety of applications
  + Variety of physical networks and devices
    - [Cloud computing](https://en.wikipedia.org/wiki/Cloud_computing), browser as access device
* IT permeates our life
  + [Internet of Things](https://en.wikipedia.org/wiki/Internet_of_things) (IoT)  
    2.1 (Computer) Networks

What is a network? A **network** can be defined recursively as

* two or more nodes/devices/hosts connected by a link
  + (e.g., copper, fiber, nothing)



* or two or more networks connected by one or more nodes (with necessary links)
  + (e.g., gateway, router)

2.2 Internet vs Web

* The [Internet](https://en.wikipedia.org/wiki/Internet) is a **network** of networks
  + **Connectivity** for heterogeneous devices
  + Various **protocols**, some details on [later slide](https://oer.gitlab.io/oer-courses/cacs/Internet.html" \l "/slide-ip-udp-tcp)
    - IPv4 and IPv6 to send messages between devices on the Internet
    - TCP and UDP to send messages between processes on Internet devices
      * (E.g., process of Web browser talks with remote process of Web server)
      * TCP: Reliable full-duplex byte streams
      * UDP: Unreliable message transfer
* The Web is an **application** using the Internet
  + Clients and servers talking **HTTP** over TCP/IP
    - E.g., GET requests asking for **HTML** pages
    - Web servers provide resources to Web clients (browsers, apps)
* Internet and Web **are** and **contain** [DSs](https://oer.gitlab.io/oer-courses/cacs/Distributed-Systems-Introduction.html)

2.3 Heterogeneity

* Internet is network of networks
* Potentially each network with
  + independent administrative control
  + different applications and protocols
  + different performance and security requirements
  + different technologies (fiber, copper, wired, wireless)
  + different hardware and operating systems
* How to overcome heterogeneity?

3 Layering and Protocols

3.1 Layering

General technique in Software Engineering and Information Systems

* Use **abstractions** to **hide complexity**
  + Abstractions naturally lead to **layering**
  + **Alternative** abstractions at each layer
    - Abstractions specified by **standards/protocols/APIs -> MODULAR**

3.2 Network Models/Architectures

* **Models** frequently have different **layers** of abstraction
  + Goal of layering: Reduce complexity
    - Each layer offers **services** to higher layers
    - Layer **interface** defines how to access its services from higher layers
* **Peer entities**, located at same layer on different machines, communicate with each other
  + **Protocols** describe rules and conventions of communication
    - E.g., message formats, sequencing of events
* **Network architecture** = set of layers and protocols

3.3 Protocol Layers

* Each protocol instance talks virtually to its **peer**

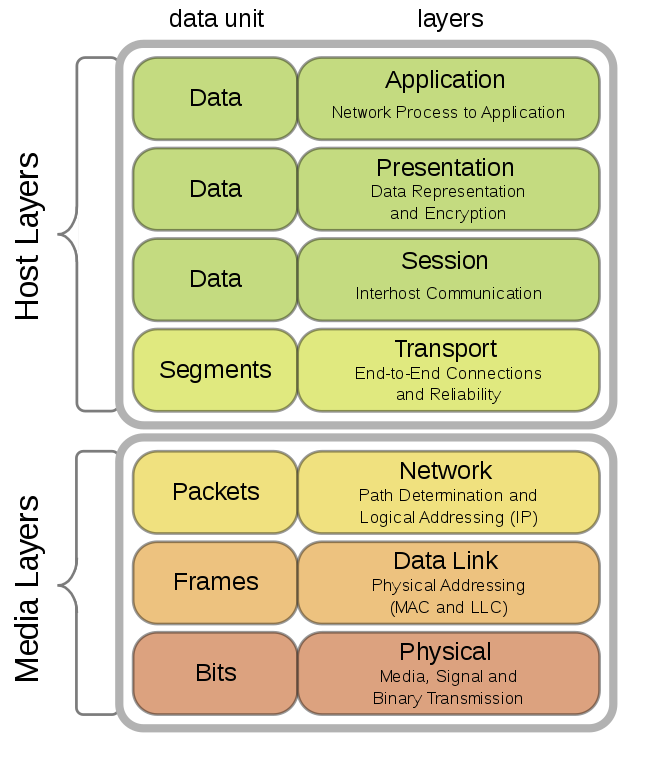
Layer 5 
Layer 4 
Layer 3 
(5) Instance 
(4) Instance 
(3) Instance 
(5) - Protocol 
(4) - Protocol 
(3) - Protocol 
(5) Instance 
(4) Instance 
(3) Instance 

* + E.g., HTTP GET request from Web browser to Web server
* Each layer communicates only by using the one below
  + E.g., Web browser asks lower layer to transmit GET request to Web server
  + Lower layer **service** accessed by an **interface**
* At bottom, messages are carried by the medium

4 Internet and OSI Models

"OSI model 
-Foc ne+wor-köng 
EVAÆS 
@bØrk 
I don? ik 
useful b Is ood 
+0 know 
L A YERS 
1 elec+rt•cal eng i neeOl'n3 såuf$, 
wires, •frequencies, i 
L: E+herne+ pco+oca + o*hers 
3: IP (IP addresses) 
L.I.. TCP + uop (po«tö 
: noboå7 euee abou}åhese 
: H T TP and -FreenJS 
does 
is L q proxy mean? 
10Qå bQlqncer 
is IQbelleJ "L 9 'l i + 
Usually i 4 looks 
1-104: heaJe,' inside 
YOU' HTTP packe+s. 
on15 know 
I P aJdressu! 
I don '+ even know 
+001 
wha+ poc+ is 
lei alone 
•t snores la-tee 
'l above P Q cke4 say S 

+

 Ap NDP

presten

4.1 OSI Reference Model

International standard

* Seven layer model to connect different systems
  + Media Layers
    1. Electric submission: Sends bits as signals
    2. Ethernet protocol: Sends frames of information
    3. IP: Sends packets from source host over multiple links to destination host
  + Host layers
    1. TCP + UDP Provides end-to-end delivery
    2. Manages task dialogs
    3. Converts different representations
    4. HTTP eg: Provides functions needed by users/applications

*4.1.2 Where are Top and Bottom?*

* In layered architectures, lower layers represent more technical details while higher layers abstract away details
  + E.g., in the OSI model the top layer (7) is the application layer, which does not care about technical communication details
* The [previous drawing](https://oer.gitlab.io/oer-courses/cacs/Internet.html" \l "/slide-drawing-osi) does not follow that convention when showing layers, but implicitly assumes it anyways (layer 3 “ignores layers 4 and above”)

4.2 OSI Model on Internet

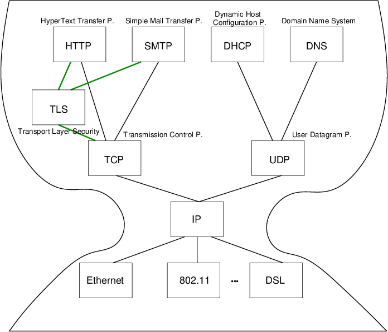
Internet architecture involves following subset of OSI layers

* Application layer
  + E.g., Web (HTTP), e-mail (SMTP), naming (DNS)
* (Presentation and session omitted, part of application protocols)
* Transport layer
  + E.g., TCP, UDP
* Network layer
  + Unifying standard: Internet Protocol (IP; v4, v6)
  + Everything over IP, IP over everything
* Data link layer
  + E.g., Ethernet, WiFi, cellular phone network, satellite link

4.3 Internet Standards

* Defined by [Internet Engineering Task Force](https://www.ietf.org/) (IETF)
* Each standard specified by set of RFCs (Requests For Comments)
  + But not every RFC is a standard, e.g., [April fool’s day](https://en.wikipedia.org/wiki/April_Fools'_Day_RFC)
  + Statuses: Informational, Experimental, Best Current Practice, Standards Track, Historic
  + Community process
    - Everyone may submit Internet Draft; typically, produced by IETF working groups
    - Afterwards peer reviewing; eventually, publication as RFC
    - David Clark: [“We reject kings, presidents and voting. We believe in rough consensus and running code.”](https://www.ietf.org/tao.html)

*4.3.1 Internet Architecture*

* “Hourglass design”
* 
* IP is focal point
  + “Narrow waist”
  + Application independent!
    - Everything over IP
  + Network independent!
    - IP over everything

*4.3.2 IP, UDP, and TCP*

* [IP](https://en.wikipedia.org/wiki/Internet_Protocol) (Internet protocol)
  + Offers best-effort host-to-host connectivity
    - **Best effort**: Try once, no effort to recover from transmission errors
    - Connection-less delivery of datagrams
* Transport layer alternatives
  + [UDP](https://en.wikipedia.org/wiki/User_Datagram_Protocol) (User Datagram Protocol)
    - Extends IP towards **best-effort application-to-application** connectivity
      * Ports identify applications/processes (e.g., 53 for DNS)
      * Connection-less
      * no check for correct transimission, no check, if everything was transmitted in the right way
* [TCP](https://en.wikipedia.org/wiki/Transmission_Control_Protocol) (Transmission Control Protocol)
  + Offers **reliable application-to-application** connectivity
    - Ports identify applications/processes (e.g., 80/443 for Web servers)
    - Full-duplex byte stream
    - Three-way handshake to establish **connection**
    - Acknowledgements and timeouts for **retransmissions**

@ bØrk 
+0 send 
-faend caf pic+ure 
bu+ our ne+work connec%oo 
bad ! f he packe+s keep 
oe+h% IoW do 
cescue 
TCP I's nefrwork pro+ocol 
you sea dæ+q 
V Every -Bime you 100k 
we bpage, you're TCP 
delivered 
NEW YORK TIMES 
how TCP works 
O open connec+ian 
CCD 
SYNhCk 
keep + rack 04 whQ± 
30+ sen+ succeSSfullY 
received 
hrs* 
1027q32 
message..s 
by •Yes Atk 
O re-icy wh 
en ne cessqr 
Send mare m 
O close connec4ian 
( also *kete check sums ) 

5 Internet Communication

5.1 IP Stack Connections

Host 
Application 
Transport 
Intemet 
Link 
Network Topology 
Router 
Router 
Data Flow 
application-specific 
end-to-end communication services 
(for applications/processes) 
Intemet 
Link 
Intemet 
Link 
Host 
B 
Application 
Transport 
Intemet 
Link 
Ethernet 
Fiber, 
Satellite, 
Ethemet 

Routers use the Internet-Layer to forward each datagram

Transport layer protocols such as UDP and TCP provide end-to-end communication services for applications, while the application layer accommodates protocols such as HTTP, SMTP, and DHCP, each of which relies on transport layer protocol services for end-to-end communication.

*5.1.1 Drawing on MAC Addresses*

Wha-4ts a.- 
compt-J4er- on 
in+eme+ has Q 
ne}work 
hello 
YOU me 
nekwock 
cad 
MAC Qådcess 
how do 1 know 
on 
else 
Someone 
sqme nef-work isof 
041 my packe+s.Q 
you don* *hafts one 
(eqsor, we use HfTPS 
•t Secuce nehd0(k4 
MAC address ? 
moce a:} t drawin s. •vns. cc! 
you make ret-Jed-r 
w i+k E4her neb/ every 
PQCke+ ge+s serd 40 MAC 
address 
here is cat 
oa:sg: 
Oa•.S6 
your has Q 
rnYS adJresse.Sl 
+0 MAC addresses 
messaae •$or Iq2.o.2.772 
seni 
ARP foc 

*5.1.2 Drawing of Packet*

'Souk 
@bMk 
yoo 
webpqe like 
Faceboo i} 
comes into your- 
compuåe( many 
acke-is 
Small p 
see w 
*hose look like! 
PRCkds ace spli+ ido 
few sec* 
(oc "he Oers 
moce 
Qna+omy of a.- PQCke+ 
dta»nngs. 3 vas. ca. 
eherne+ / w i 
MAc 
address 
inåeme+ pro+ocol 
FROM: 172.%.2.s f0•. 123. q.2.32 
Sequence number: g 77392 
by+es 
checksum•. 
sen+ so 
de+ec+ coercp}ed 
data. 
•from: port qq79 80 
ATT p ( or wk Q+evec) 
GET / HTTP 11.1 
Hos+: 3 003 le. com 
Accep4- Language en-US 
chame 
cons+an+l' as your 
packe+ moves be-hi-reo 
com - 
in chare 04 
3 e+h•ns your packet 
+0 right server 
(like address on 
an envelope ) 
in charge 
preven+i% d 
cot-rum '00 hetpt% 
you ceåry Ios+ pqeke6 
video s teamtnq useS 
UDP ins4eqd. CD p 
does +0 be 
you're 
+0 send 

*5.1.3 Typical Communication Steps* Prerequisites

* Internet communication requires numeric IP addresses
  + **Lookup** of IP addresses for human readable names via **DNS**
    - DNS is request-reply protocol
    - DNS client (e.g., th
    - e browser) asks DNS server for IP address of name, e.g., query for [www.wwu.de](http://www.wwu.de/) may result in 128.176.6.250
    - (And more)
* LAN communication requires MAC (media access control, hardware) addresses
  + MAC address: Hardware address of network card (e.g., Ethernet, WiFi)
    - E.g., 02:42:fa:5c:4a:4a
  + **Lookup** of MAC addresses for IP addresses via **ARP** (Address Resolution P.)
    - Send ARP request (broadcast) into local network, asking for MAC address of given IP addresses
* Ex.: Send HTTP message M to host [www.wwu.de](http://www.wwu.de/)
  1. Perform **DNS** lookup for [www.wwu.de](http://www.wwu.de/)
     + Returns IP address 128.176.6.250
  2. [Encapsulate](https://oer.gitlab.io/oer-courses/cacs/Internet.html" \l "/slide-encapsulation-demux) M by adding TCP header
     + Source and destination TCP **ports**: Numbers that identify processes
  + Typically, destination port 80 for Web servers with HTTP
  + Encapsulate TCP segment by adding IP header
    - Source and destination IP addresses
    - [Demux key](https://oer.gitlab.io/oer-courses/cacs/Internet.html" \l "/slide-encapsulation-demux) to indicate that TCP segment is contained
* Ex.: Send HTTP message M to host [www.wwu.de](http://www.wwu.de/)
  1. Perform DNS lookup for [www.wwu.de](http://www.wwu.de/)
  2. Encapsulate with TCP header
  3. Encapsulate with IP header
  4. **Routing** decision to determine IP address of **next hop** router
     + Returns IP address IP\_R within sender’s network
     + E.g., 128.176.158.1 at my work, 192.168.178.1 at home
  5. **ARP** lookup to determine MAC address for IP\_R
     + E.g., 0:0:c:7:ac:0
  6. Encapsulate IP datagram with LAN-specific header with MAC address, send via LAN to router
  + Routers repeat steps (4) - (6) to forward M to final destination

5.2 Encapsulation

Application 
TCP 
Net 
[request from browser B 
at port 42042 
for server S at port 80] 
GET index.html 
420421801.. 
GET index.html 
420421801. 
GET index.html 
RI IBIOx08001 
420421801.. 
GET index.html 
LAN-specific header, 
including MAC addresses 
R21R1bIOx08001 
420421801.. 
GET index.html 
RI 
GET index.html 
s 
420421801.. 
GET index.html 
420421801.. 
GET index.html 
SIR2bIOx08001 
420421801.. 
GET index.html 

5.3 Encapsulation and Demux Keys

* **Encapsulation**
  + Protocol specific header added for each layer
    - Starting from “pure” application message
    - Headers prepended when moving down the protocol stack
  + Headers “unwrapped” when moving up again
* **Demux** key
  + Identifies recipient protocol at next higher layer
  + Different protocols use different forms of demux keys (see previous slide)
    - Ethernet header contains **type** field (IPv4 = 0x0800, ARP = 0x0806)
    - IP header contains **protocol** field (TCP = 6, UDP = 17)
    - TCP header contains **port** (application id) as demux key

Quiz:

* Every device that is part of the Internet implements the Internet Protocol -> right
* 128.176.6.250 is a valid address with IPv4 -> right
* Some ranges of IP addresses are private in the sense that they can be reused in different networks (not part of presentation) -> right
* IP is a best-effort protocol, which means that the loss of messages is countered with retry mechanisms -> wrong :(

6 End-to-End Argument

6.1 Network: Core, Edge, Endpoint

* Network **core**: Devices **implementing** the network
  + Routers, switches
* Network **edge**: Devices **using** the network
  + Computers, “smart” devices, IoT devices
* **Endpoints** of communication: Distributed **applications**
  + Processes that send and receive messages
    - E.g., your e-mail client, your Web browser, your messenger
    - Beware: Who is the other end for your browser? Who for your mail client and messenger? - trustable recipient?

(  
6.2 Overarching Question

* What functionality to implement in the network core, what within communication endpoints?
  + Observations
    - If functionality is available as Internet standard, every application can immediately use it. No need to reinvent wheels.
    - Simplicity and generality of protocols increase potential for re-use, e.g., IP allows to connect “everything.”
  + Answer to question given in [[SRC84]](https://oer.gitlab.io/oer-courses/cacs/Internet.html" \l "/slide-bibliography): End-to-end argument
  + Intuition
    - Some functionality needs application knowledge
    - Such functionality cannot be implemented inside the net
    - In general, application functionality should not be implemented in the net

)

6.3 End-to-End Definition

* Quotes from [[SRC84]](https://oer.gitlab.io/oer-courses/cacs/Internet.html" \l "/slide-bibliography)
  + “The principle, called the end-to-end argument, suggests that functions placed at low levels of a system may be redundant or of little value when compared with the cost of providing them at that low level.”
  + “The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible. (Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.)”
* End to end deals with the question of good protocol design

6.4 End-to-End Example

* Careful file transfer
  + Read file from disk, transfer over Internet, write to disk at remote end
  + Possible errors, leading to corrupted data
    - Disk error
    - Software errors in file system, file transfer, network protocol, buffering or copying
    - Hardware errors (e.g., processor or memory failures)
    - Network failures/attacks (messages lost or bits changed)
    - Crash in the middle of the transfer
  + Possible solutions
    - Lots of “small” tests
    - One end-to-end checksum check, with retry in case of errors
* How many “small” tests will be necessary?
  + Notice: A test regarding network transfer does not help much since all other types of errors can still corrupt data
    - Hence, an **end-to-end check** will be **necessary** anyways
  + However, from a **performance** perspective, a single end-to-end check may be costly
    - Consider transfer of some GB, which may take a long time
      * The end-to-end check detects individual errors only after full transfer
      * In contrast, intermediate checks may identify individual bit errors early, allowing partial retries

6.5 End-to-End Security

* Above observations also apply to [security goals of confidentiality and integrity](https://oer.gitlab.io/OS/Operating-Systems-Security.html" \l "slide-security-goals)
  + Confidentiality and integrity are [end-to-end security goals](https://oer.gitlab.io/OS/Operating-Systems-Security.html" \l "slide-end-to-end-security)
  + If you want them, you must neither rely on link level nor hop-by-hop assurances
    - As offered by, e.g., [WPA variants](https://en.wikipedia.org/wiki/Wi-Fi_Protected_Access), IPsec, VPN, De-Mail
* You must protect your data inside your applications (end-to-end)
  + Recall e-mail and messaging mentioned above

*6.5.1 Hybrid End-to-End Encryption*

0 
、 VI 。 
0 

6.6 Then vs Now

Rethinking the Design of the Internet -> Design was first in a safe world...

* Challenges since 1980s
  + Untrustworthy world, e.g., attacks, spam, DDoS
    - Need more mechanism in the core to enforce “good” behavior?
  + More demanding applications, e.g., video streaming
    - Best effort model may not be good enough, need intermediate storage sites for streaming?
  + ISP service differentiation
    - Different pieces of content provided with different QoS guarantees?
  + Rise of third-party involvement
    - Officials of organizations or governments interpose themselves
  + Less sophisticated users
    - From initial experts to Joe Sixpack, who may be overwhelmed by complexity in endpoints
* [RFC 3724](https://tools.ietf.org/html/rfc3724), 2004: End-to-end is still relevant, though
  + End-to-end manages state at the edges, not the core
    - Failures in core do not affect application state
  + Protection of innovation, reliability, trust

Web

2.1 History of the Web

* 1945, [Vannevar Bush: As we may think](http://www.theatlantic.com/unbound/flashbks/computer/bushf.htm)
  + Memex for information storage
  + Associative indexing (Hyperlinks)
* 1989, [article by Tim Berners-Lee](https://www.w3.org/History/1989/proposal.html)
  + Distributed hypertext system, “»web« of notes with links”
  + Initially for cooperation among physicists at CERN
* May 1991
  + Distributed information system based on HTML, HTTP, and client software at CERN
* August 1991
  + Availability of CERN files announced in <news:alt.hypertext>
    - <http://groups.google.com/group/alt.hypertext/msg/395f282a67a1916c>
* 1992, NCSA **Web Server** available
  + National Center for Supercomputing Applications, University of Illinois, Urbana-Champaigne
* 1993, Mosaic **browser** created at NCSA
* 1994, [World Wide Web Consortium](https://www.w3.org/) (W3C) founded by Tim Berners-Lee
  + Publication of technical reports and “recommendations”
* Now
  + Web 2.0, Semantic Web, cloud computing, browser as access device

2.3 WWW/Web

* Standards
  + W3C ([HTML 4 Specification](https://www.w3.org/TR/html4/intro/intro.html))
    - “The World Wide Web (Web) is a network of information resources.”
  + [HTTP/1.1 Specification (RFC 7230)](https://tools.ietf.org/html/rfc7230)
    - “The Hypertext Transfer Protocol (HTTP) is a stateless application-level protocol for distributed, collaborative, hypertext information systems.”
* Distributed information system
  + Client-Server architecture
    - Web clients (browsers) and servers exchange HTTP messages based on [Internet](https://oer.gitlab.io/oer-courses/cacs/Internet.html) standards
  + Sample Web standards (application layer of Internet architecture)
    - [URIs](https://oer.gitlab.io/oer-courses/vm-neuland/03-URIs.html) (Uniform Resource Identifiers, generalize [URLs](https://en.wikipedia.org/wiki/URL) and [URNs](https://en.wikipedia.org/wiki/Uniform_Resource_Name))
    - HTTP (now)
    - ((X)HTML)

3 HTTP

3.1 HTTP

* Hypertext Transfer Protocol
  + [HTTP/1.1, RFC 7230](https://tools.ietf.org/html/rfc7230)
    - Plain text messages, discussed subsequently
  + [HTTP/2, RFC 7540](https://tools.ietf.org/html/rfc7540)
    - Adds frame format with compression
    - [Adoption increasing](https://w3techs.com/technologies/details/ce-http2/all/all), from 15% in July 2017 to 28% in July 2018 to 33.4% (as of 2019-07-03; after peak of about 37% in June 2019)
  + [HTTP/3](https://en.wikipedia.org/wiki/HTTP/3) upcoming
* Request/response protocol
  + Specific message format
  + Several access methods
  + Requires [reliable transport protocol](https://oer.gitlab.io/oer-courses/cacs/Internet.html" \l "slide-ip-udp-tcp)
    - Typically TCP/IP, port 80 (or port 443 for [HTTPS](https://en.wikipedia.org/wiki/HTTPS))

3.2 Excursion: Manual Connections

* [HTTP](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol) (before HTTP/2) and [SMTP](https://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol) are plain text protocols
  + With encrypted variants [HTTPS](https://en.wikipedia.org/wiki/HTTPS) and [SMTPS](https://en.wikipedia.org/wiki/SMTPS) (or [STARTTLS](https://en.wikipedia.org/wiki/STARTTLS))
* Enables experiments on the command line
  + Type (or copy&paste) request, see server response
  + For unencrypted connections, telnet can be used (preinstalled or available for lots of OSs)
  + For encrypted connections, gnutls-cli can be used (part of [GnuTLS](https://www.gnutls.org/), which is [free software](https://oer.gitlab.io/OS/Operating-Systems-Motivation.html" \l "slide-free-software))
    - TLS = [Transport Layer Security](https://en.wikipedia.org/wiki/Transport_Layer_Security)
      * Successor to SSL
      * Layer between application layer and TCP, recall [Internet architecture](https://oer.gitlab.io/oer-courses/cacs/Internet.html" \l "slide-internet-architecture)
      * Secure channels based on [asymmetric cryptography](https://oer.gitlab.io/OS/Operating-Systems-Security.html" \l "slide-asym-intuition)

*3.2.1 Warnings*

* Next two slides demonstrate how to type HTTP commands (for an improved understanding of the protocol)
  + Subsequent examples with [www.informationelle-selbstbestimmung-im-internet.de](http://www.informationelle-selbstbestimmung-im-internet.de/) **require** GnuTLS
    - Server redirects from port 80 to port 443
  + If your manual typing is too slow, connections may **time out** (e.g., “Peer has closed the GnuTLS connection”)
  + Also, use of backspace or cursor keys may destroy connections
* Suggestion: Type in text editor and copy&paste into command line

*3.2.2 telnet*

* Original telnet purpose: Login to remote host
  + **Insecure** plaintext passwords
  + Nowadays, remote login performed with [Secure Shell](https://en.wikipedia.org/wiki/Secure_Shell), ssh
* Establish **TCP connection** to destination port
  + telnet [www.google.de](http://www.google.de/) 80 (port 80 for HTTP)
    - (For variants without visual feedback possibly followed by ctrl-+ or ctrl-], set localecho [enter] [enter])
    - GET / HTTP/1.1 [enter]
    - Host: [www.google.de](http://www.google.de/) [enter] [enter]
    - (Context for above lines [soon](https://oer.gitlab.io/oer-courses/cacs/Web-and-E-Mail.html" \l "/slide-http-request))
  + **Beware**: Buggy telnet implementations may stop sending after first line (use Wireshark to verify)

*3.2.3 gnutls-cli*

Establish [TLS](https://en.wikipedia.org/wiki/Transport_Layer_Security) protected TCP connection with [GnuTLS](https://www.gnutls.org/)

* Alternative to telnet on previous slide
* gnutls-cli --crlf [www.informationelle-selbstbestimmung-im-internet.de](http://www.informationelle-selbstbestimmung-im-internet.de/)
  + (HTTPS on port 443 by default)
  + GET /chaosreader.html HTTP/1.1 [enter]
  + Host: [www.informationelle-selbstbestimmung-im-internet.de](http://www.informationelle-selbstbestimmung-im-internet.de/) [enter] [enter]
* [SMTP](https://oer.gitlab.io/oer-courses/cacs/Web-and-E-Mail.html" \l "slide-smtp) for e-mail, port 587 as alternative to 25
  + gnutls-cli --crlf --starttls -p 587 secmail.uni-muenster.de
    - (Type ehlo localhost, then starttls; press ctrl-d to enter TLS mode; needs authentication)

3.3 Excursion: Browser Tools

Modern browsers offer developer tools

* E.g., press ctrl-shift-I with Firefox
* Tools to inspect HTML, CSS, Javascript
* Tools to inspect HTTP traffic (Network tab)
  + Live view on browser requests and server responses
    - With details on timing, caching, headers
* Console with error messages

3.4 HTTP Messages

* Requests and responses
  + Generic message format of [RFC 822](https://tools.ietf.org/html/rfc822), 1982 (822→2822→[5322](https://tools.ietf.org/html/rfc5322))
    - Originally for e-mail, extensions for binary data
      * Lines [end with CRLF](https://en.wikipedia.org/wiki/Newline), \r\n below
  + Messages consist of
    - Headers
      * In HTTP always a distinguished start-line (request or status)
      * Then zero or more headers
    - Empty line
    - Optional message body
  + Sample GET **request** (does not have a body)
    - GET /chaosreader.html HTTP/1.1\r\n
  + Host: [www.informationelle-selbstbestimmung-im-internet.de\r\n](http://www.informationelle-selbstbestimmung-im-internet.de/r/n)
  + \r\n
* Excerpt of sample HTTP **response** to previous GET request
  + HTTP/1.1 200 OK\r\n
* Date: Wed, 08 Apr 2020 13:30:10 GMT\r\n
* Server: Apache\r\n
* Last-Modified: Wed, 24 Jul 2019 12:25:46 GMT\r\n
* ETag: "2cd1-58e6c6898dce2"\r\n
* Content-Length: 11473\r\n
* more headers omitted
* Content-type: text/html; charset=utf-8\r\n
* \r\n
* HTML code as body

3.5 HTTP Methods

* Case-sensitive (capital letters)
  + GET (Request for resource, see [section 4.3.1](https://tools.ietf.org/html/rfc7231" \l "section-4.3.1))
  + HEAD (Request information on resource, see [section 4.3.2](https://tools.ietf.org/html/rfc7231" \l "section-4.3.2))
  + POST (Transfers entity, see [section 4.3.3](https://tools.ietf.org/html/rfc7231" \l "section-4.3.3))
    - Annotations, postings, forms, database extensions
  + PUT (Creates new resource on server, see [section 4.3.4](https://tools.ietf.org/html/rfc7231" \l "section-4.3.4))
  + DELETE (Deletes resource from server, see [section 4.3.5](https://tools.ietf.org/html/rfc7231" \l "section-4.3.5))
  + CONNECT (Establish tunnel with proxy, see [section 4.3.6](https://tools.ietf.org/html/rfc7231" \l "section-4.3.6))
  + OPTIONS (Asks for server capabilities, see [section 4.3.7](https://tools.ietf.org/html/rfc7231" \l "section-4.3.7))
  + TRACE (Tracing of messages through proxies, see [section 4.3.8](https://tools.ietf.org/html/rfc7231" \l "section-4.3.8))

3.6 Conditional GET

* GET under conditions
  + Requires (case-insensitive) request header
    - (Can be used by browser to check if [cached](https://oer.gitlab.io/oer-courses/cacs/Web-and-E-Mail.html" \l "slide-http-caching) version still fresh)
    - If-Modified-Since
    - If-Match
    - If-None-Match
* Example
  + Request
    - GET /chaosreader.html HTTP/1.1   
      Host: [www.informationelle-selbstbestimmung-im-internet.de](http://www.informationelle-selbstbestimmung-im-internet.de/)   
      If-None-Match: "2cd1-58e6c6898dce2"
  + Response
    - HTTP/1.1 304 Not Modified   
      Date: Wed, 08 Apr 2020 14:07:31 GMT   
      additional headers

3.7 Sample Status Codes

* Three digits, first one for class of response
  + 1xx: Informational - Request received, continuing process
    - 100: Continue - Client may continue with request body
  + 2xx: Successful - Request successfully received, understood, and accepted
    - 200: OK
  + 3xx: Redirection - Further action necessary to complete request
    - 302: Found (temporarily under different URI)
    - 303: See Other (redirect to different URI in Location header)
    - 304: Not Modified ([previous slide](https://oer.gitlab.io/oer-courses/cacs/Web-and-E-Mail.html" \l "/slide-conditional-get))
  + 4xx: Client Error - Request with bad syntax or cannot be fulfilled
    - 403: Forbidden
    - 404: Not Found
  + 5xx: Server Error - Server failed for apparently valid request

4 Server State and Cookies

4.1 State Models

* **Stateless**: Server does not maintain client state
  + Advantages
    - Simplified server design, reduced resource usage
    - State changes on server do not require client notifications
    - Recovery (restart after server crash) “simple”: No client state to restore
  + E.g.: HTTP, DNS
    - Server forgets client after request
    - No session
* **Stateful**: Server maintains client state
  + E.g., file server with table of pairs (Client, File) for caching
    - Keep track which client has current version
    - Performance improvement via locality
  + Recovery requires to restore consistent state

4.2 Stateful Web Applications

* HTTP is stateless
  + Yet, Web applications often maintain client state
    - E.g., personalized session after login
      * Virtual shopping cart
      * Shopping history, preferences
      * Exercises in Learnweb
  + Solution for stateful applications
    - Manage state of related requests as session outside HTTP
    - Use HTTP messages to transfer session IDs (next slide)

4.3 Session IDs

* Session ID = Identifier to connect subsequent/related requests and responses
  + Typical variant: Client-side storage of IDs in browser
    - ID sent by server S, stored by browser ([cookie](https://oer.gitlab.io/oer-courses/cacs/Web-and-E-Mail.html" \l "/slide-cookies) or [local storage](https://developer.mozilla.org/en/docs/Web/API/Window/localStorage))
    - Browser includes IDs set by S for every subsequent visit of S
      * Think of automatic ID card (whose contents you do not understand)
      * My browsers remove cookies and clear local storage upon exit
  + Alternative: Server-side, session ID embedded in dynamically generated URIs
    - May hinder caching
      * URI does not identify resource any longer

*4.3.1 Cookies*

= HTTP State Management Mechanism

* Idea:
  + Client stores data sent by server
  + Client sends this data with subsequent requests
    - Without understanding that data at all
* Details
  + Cookie is **named byte string**
  + Server transfers cookie in Set-Cookie (2) header in response
    - Set-Cookie: Version 0/Netscape and [RFC 6265](https://tools.ietf.org/html/rfc6265)
    - Set-Cookie2: Version 1, [RFC 2965](https://tools.ietf.org/html/rfc2965)
    - (Besides, JavaScript may create cookie at client)
  + Client sends cookie in Cookie header in requests
* Note: Sometimes you may read that cookies are text files
  + That is usually wrong, misleading, and irrelevant
  + Modern browsers store cookies as rows in a relational database
    - Storage in filesystem or database is an implementation detail
* Cookies have name, value, optional [attributes/flags](https://en.wikipedia.org/wiki/HTTP_cookie" \l "Cookie_attributes), e.g.:
  + Expires, Max-Age
    - Determine lifetime of cookie
    - If both missing: “Session” cookie to be deleted when browser exits
  + Domain
    - DNS domain of servers to which the cookie should be sent

5 Caching

5.1 HTTP Caching

* Caching reduces latency and server load for identical requests
* HTTP [caching](https://oer.gitlab.io/oer-courses/cacs/Distributed-Systems-Introduction.html" \l "slide-caching) assumptions
  + URI identifies resource, stability, client-independence
* Semantic transparency
  + Caching is not visible to users
  + Response from cache is equivalent to hypothetical one from server

5.2 HTTP Caching Mechanisms

* Expiration
  + Server may indicate expiration date in [Expires](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Expires) or [Cache-Control](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control) header
* Validation
  + After expiration date, cache must check whether resource still usable
  + May return new expiration date
    - [Conditional GET](https://oer.gitlab.io/oer-courses/cacs/Web-and-E-Mail.html" \l "slide-conditional-get) (“Slow hit”)

5.3 HTTP Caching Rules

* Complex rules, lots of details
  + (Some details on [Cache Control](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control) header)
* Server may limit caching
  + no-store, no-cache, must-revalidate
* Client may
  + enforce validation
    - no-cache
  + forbid caching
    - no-store

6 Proxies

6.1 Web Proxies

Web proxy server is intermediary between client and server

* Acts as server to client
  + Proxy accepts request from client
    - Then acts as client to server to obtain response
  + Proxy delivers response to client
* Acts as client to server
  + Proxy sends request of real client to server
    - Server just sees some client request
  + Proxy obtains response from server

6.2 Sample Proxy Applications

* [Cache](https://oer.gitlab.io/oer-courses/cacs/Web-and-E-Mail.html" \l "/slide-http-caching)
* Firewall/Content filter
* Anonymizer, e.g., [Tor](https://www.torproject.org/)
  + My [privacy policy](https://oer.gitlab.io/privacy.html) recommends surfing via Tor
* Debugging tool
  + E.g., intercept and analyze app network data
* Surrogate/Reverse proxy, Content Delivery Network (CDN)
  + [Replicated](https://oer.gitlab.io/oer-courses/cacs/Distributed-Systems-Introduction.html" \l "slide-replication) contents, inbound messages intercepted and redirected, e.g.:
    - Load balancing
    - Geographical diversity (reduced latency, increased availability)

7 E-Mail

7.1 E-Mail Basics

* Among oldest Internet applications
* Message format
  + Based on [RFC 822](https://tools.ietf.org/html/rfc822), 1982 (later taken up in HTTP)
  + Extended with [Multipurpose Internet Mail Extensions](https://en.wikipedia.org/wiki/Multipurpose_Internet_Mail_Extensions) (MIME)
    - Content-Type (type of data contained in message)
    - Content-Transfer-Encoding (how data in message body is encoded)
* Plaintext messages
  + E-mail is like **postcard**, written with **erasable pencil**
    - Neither [confidentiality nor integrity](https://oer.gitlab.io/OS/Operating-Systems-Security.html" \l "slide-security-goals)
  + Learn self-defense, use [GnuPG](https://oer.gitlab.io/OS/Operating-Systems-Security.html" \l "slide-gnupg) if you don’t like this
    - SSL/TLS insufficient approach, recall [end-to-end security](https://oer.gitlab.io/oer-courses/cacs/Internet.html" \l "slide-end-to-end-security)

7.2 Message Transfer

* Terminology
  + **Mail User Agent** (MUA): Your mail reader
    - E.g., browser, Thunderbird, Emacs
  + **Mail Transfer Agent** (MTA): Mail server/daemon
    - E.g., sendmail, exim, postfix
* **Simple Mail Transfer Protocol**, 1982 (SMTP, RFC 821→2821→[5321](https://tools.ietf.org/html/rfc5321))
  + Outgoing messages, MUA-to-MTA, MTA-to-MTA
    - Plaintext (TCP/IP, port 25)