Netzwerktechnik und IT-Netze

Chapter 1: Computer Networks and the Internet

Vorlesung im WS 2016/2017

Bachelor Informatik

(3. Semester)

Prof. Dr. rer. nat. Andreas Berl

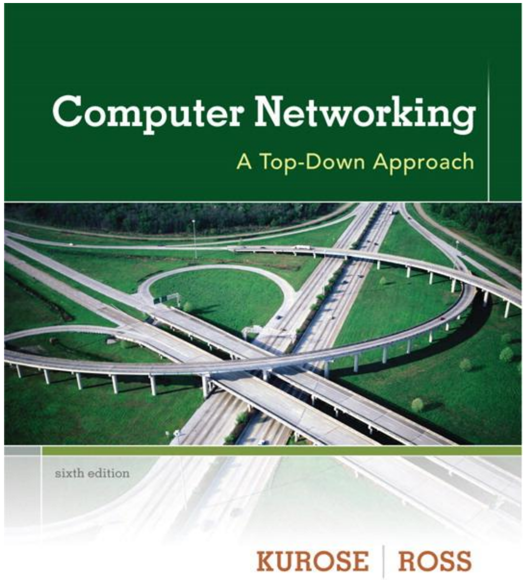
Fakultät für Elektrotechnik, Medientechnik und Informatik

Overview

- Introduction
- Computer Networks and the Internet
- Application Layer
  - WWW, Email, DNS, and more
  - Socket programming
- Transport Layer
- Network Layer
- Link Layer
- P2P Networks
- Firewalls

Introduction

- A note on the use of these power point slides:
  -
  -



Computer Networking: A Top Down Approach
6th edition
Jim Kurose, Keith Ross
Addison-Wesley
March 2012

Chapter 2: Roadmap

- Principles of network applications
- Web and HTTP
- FTP
- Electronic mail
  - SMTP, POP3, IMAP
- DNS
- P2P Applications
- Socket programming with UDP and TCP

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Chapter 2: application layer

- Our goals:
  - Conceptual, implementation aspects of network application protocols
  - Transport-layer service models
  - Client-server paradigm
  - Peer-to-peer paradigm
  - Learn about protocols by examining popular application-level protocols
    - HTTP
    - FTP
    - SMTP / POP3 / IMAP
    - DNS

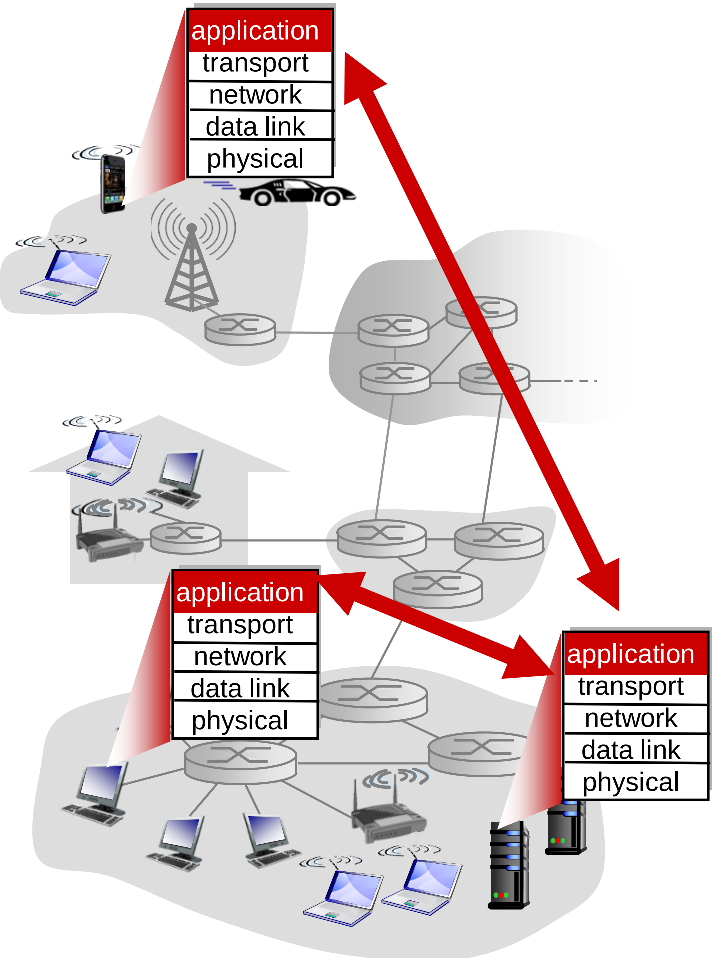
- Our goals:
  - Creating network applications
  - Socket API

Some network apps

- E-mail
- Web
- Text messaging
- Remote login
- P2P file sharing
- Multi-user network games
- Streaming stored video (YouTube, Hulu, Netflix)
- Voice over IP (e.g., Skype)
- Real-time video conferencing
- Social networking
- Search
- …
- …

Creating a network app

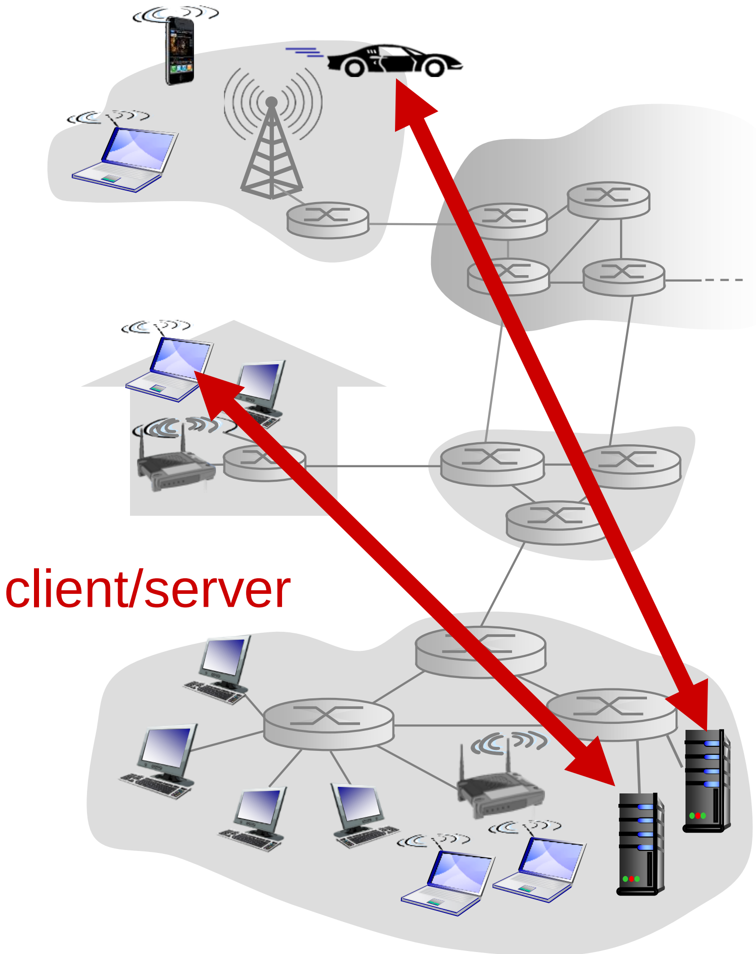
- Write programs that:
  - Run on (different) end systems
  - Communicate over network
  - e.g., Web server software communicates with browser software
- No need to write software for network-core devices
  - Network-core devices do not run user applications
  - Applications on end systems allows for rapid app development, propagation



Application architectures

- Possible structure of applications:
  - Client-server
  - Peer-to-peer (P2P)

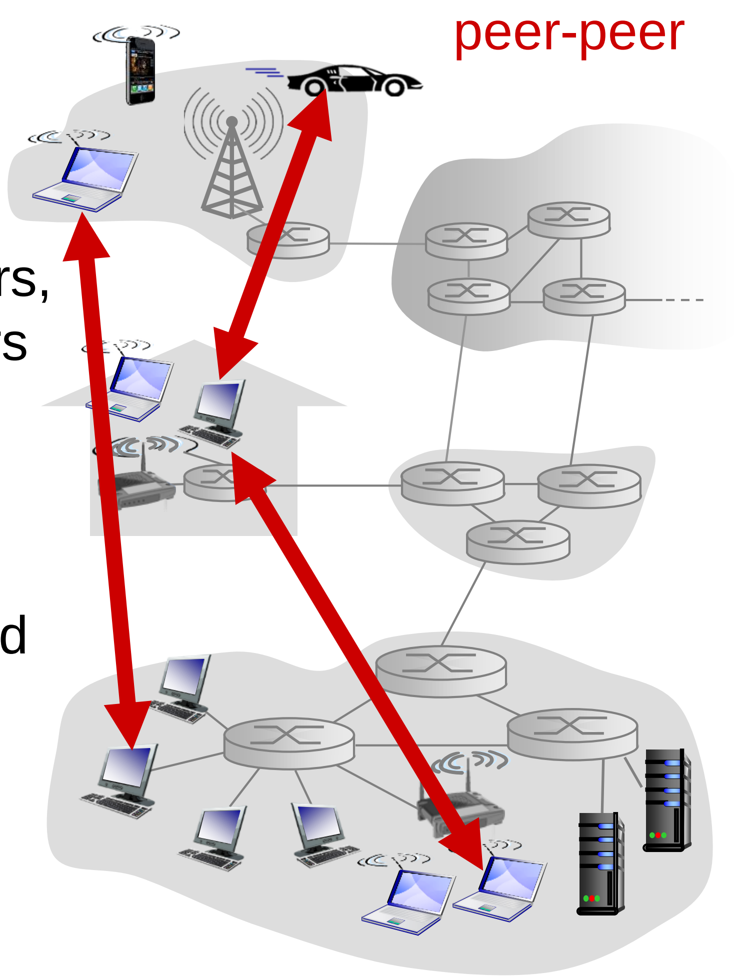
Client-server architecture



- Server:
  - Always-on host
  - Permanent IP address
  - Data centers for scaling
- Clients:
  - Communicate with server
  - May be intermittently connected
  - May have dynamic IP addresses
  - Do not communicate directly with each other

P2P architecture

- No always-on server
- Arbitrary end systems directly communicate
- Peers request service from other peers, provide service in return to other peers
  - Self scalability – new peers bring new service capacity, as well as new service demands
- Peers are intermittently connected and change IP addresses
  - Complex management



Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Processes communicating

- Process: program running within a host
  - Within same host, two processes communicate using inter-process communication (defined by OS)
  - Processes in different hosts communicate by exchanging messages
  - Aside: applications with P2P architectures have client processes & server processes

Clients, Servers

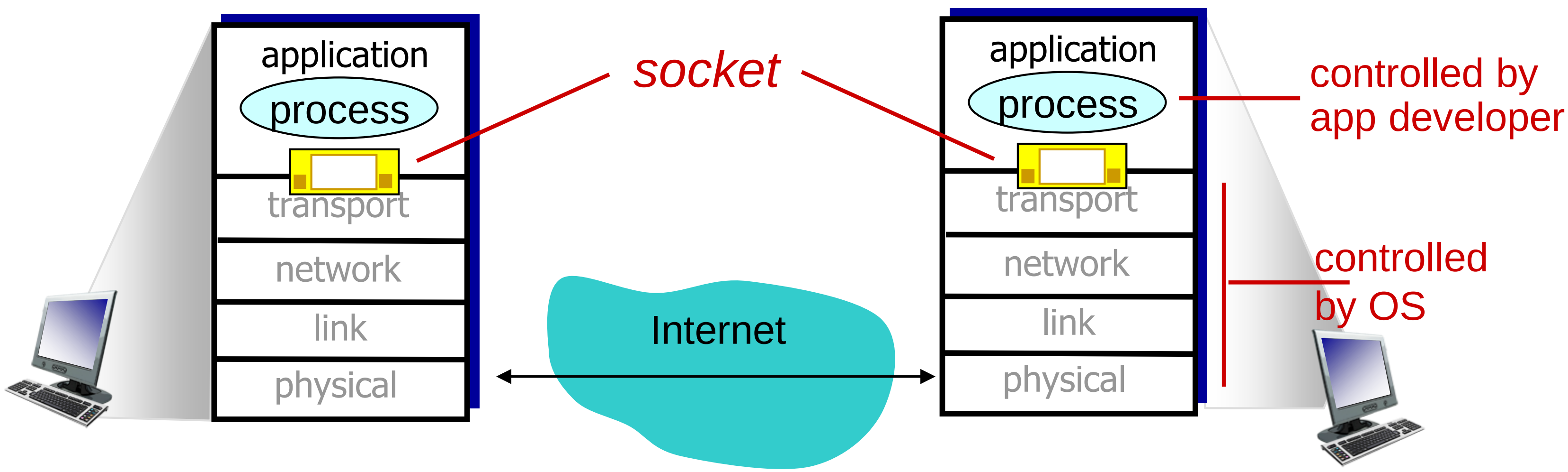
Client process: process that initiates communication

Server process: process that waits to be contacted

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Sockets

- Process sends/receives messages to/from its socket
- Socket analogous to door
  - Sending process shoves message out door
  - Sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process



Addressing processes

- To receive messages, process must have identifier
- Host device has unique 32-bit IP address
- Q: does IP address of host on which process runs suffice for identifying the process?
  - A : no, many processes can be running on same host

- Identifier includes both IP address and port numbers associated with process on host.
- Example port numbers:
  - HTTP server: 80
  - Mail server: 25
- To send HTTP message to gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - Port number: 80
- more shortly…

App-layer protocol defines

- Types of messages exchanged,
  - E.g., request, response
- Message syntax:
  - What fields in messages & how fields are delineated
- Message semantics
  - Meaning of information in fields
- Rules for when and how processes send & respond to messages

- Open protocols:
  - Defined in RFCs
  - Allows for interoperability
  - e.g., HTTP, SMTP
- Proprietary protocols:
  - E.g., Skype

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

What transport service does an app need?

- Data integrity
  - Some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
  - Other apps (e.g., audio) can tolerate some loss
- Timing
  - Some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"
- Throughput
  - Some apps (e.g., multimedia) require minimum amount of throughput to be "effective"
  - Other apps ("elastic apps") make use of whatever throughput they get
- Security
  - Encryption, data integrity, …

Transport service requirements: common apps

Application	Data loss	Throughput	Time sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
text messaging	no loss	elastic	yes and no

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Internet transport protocols services

- TCP service
  - Reliable transport: between sending and receiving process
  - Flow control: sender won't overwhelm receiver
  - Congestion control: throttle sender when network overloaded
  - Connection-oriented: setup required between client and server processes
  - Does not provide: timing, minimum throughput guarantee, security
- UDP service
  - Unreliable data transfer: between sending and receiving process
  - Does not provide: reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Internet apps: application, transport protocols

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (e.g., YouTube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	TCP or UDP

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Securing TCP

- TCP & UDP
  - No encryption
  - Cleartext passwds sent into socket traverse Internet in cleartext
- SSL
  - Provides encrypted TCP connection
  - Data integrity
  - End-point authentication

- SSL is at app layer
  - Apps use SSL libraries, which “talk” to TCP
- SSL socket API
  - Cleartext passwds sent into socket traverse Internet encrypted
  - See Chapter 7

Chapter 2: Roadmap

- Principles of network applications
- Web and HTTP
- FTP
- Electronic mail
  - SMTP, POP3, IMAP
- DNS
- P2P Applications
- Socket programming with UDP and TCP

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

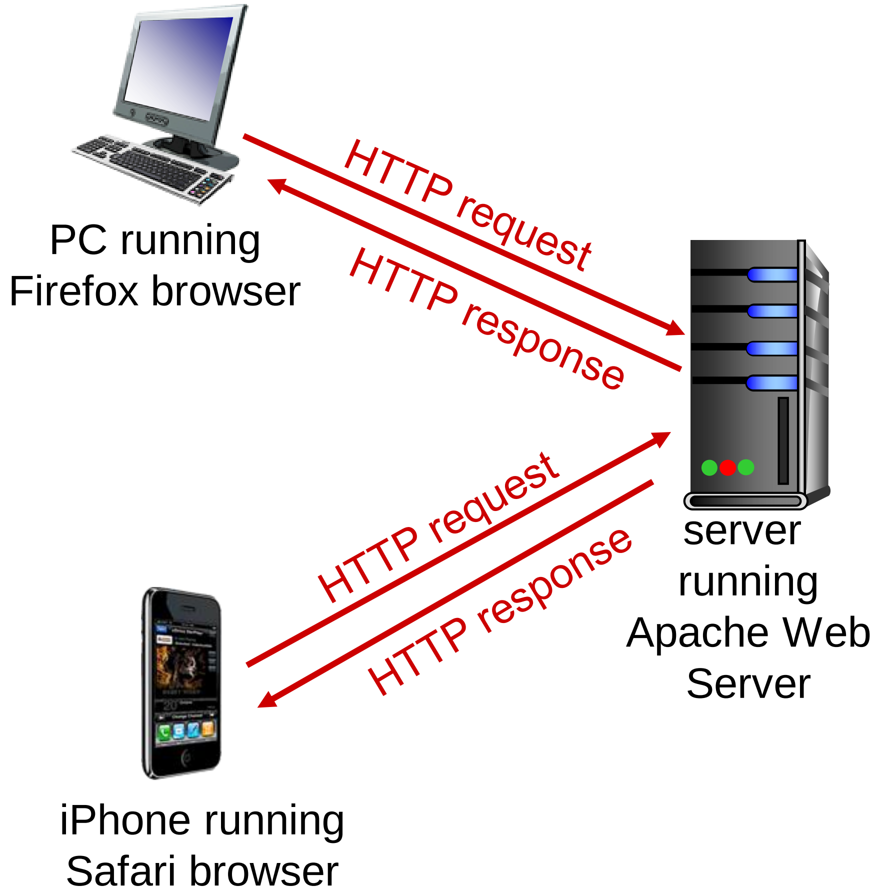
Web and HTTP

- First, a review…
  - Web page consists of objects
  - Object can be HTML file, JPEG image, Java applet, audio file,…
  - Web page consists of base HTML-file which includes several referenced objects
  - Each object is addressable by a URL, e.g.,

$$\underbrace{\texttt{www.someschool.edu}}_{\text{host name}}\underbrace{\texttt{/someDept/pic.gif}}_{\text{path name}}$$

HTTP overview

- HTTP: hypertext transfer protocol
  - Web's application layer protocol
  - Client/server model
    - Client: Browser that requests, receives, (using HTTP protocol) and "displays" Web objects
    - Server: Web server sends (using HTTP protocol) objects in response to requests



HTTP overview (continued)

- Uses TCP
  - Client initiates TCP connection (creates socket) to server, port 80
  - Server accepts TCP connection from client
  - HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
  - TCP connection closed

- HTTP is "stateless"
  - Server maintains no information about past client requests

aside

Protocols that maintain "state" are complex!

- Past history (state) must be maintained
- If server/client crashes, their views of "state" may be inconsistent, must be reconciled

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

HTTP connections

- Non-persistent HTTP
  - At most one object sent over TCP connection
    - Connection then closed
  - Downloading multiple objects required multiple connections
- Persistent HTTP
  - Multiple objects can be sent over single TCP connection between client, server

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Non-persistent HTTP

- Suppose user enters URL:
  - www.someSchool.edu/someDepartment/home.index

1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. “accepts” connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

Time

Non-persistent HTTP (cont.)

4. HTTP server closes TCP connection.

Time

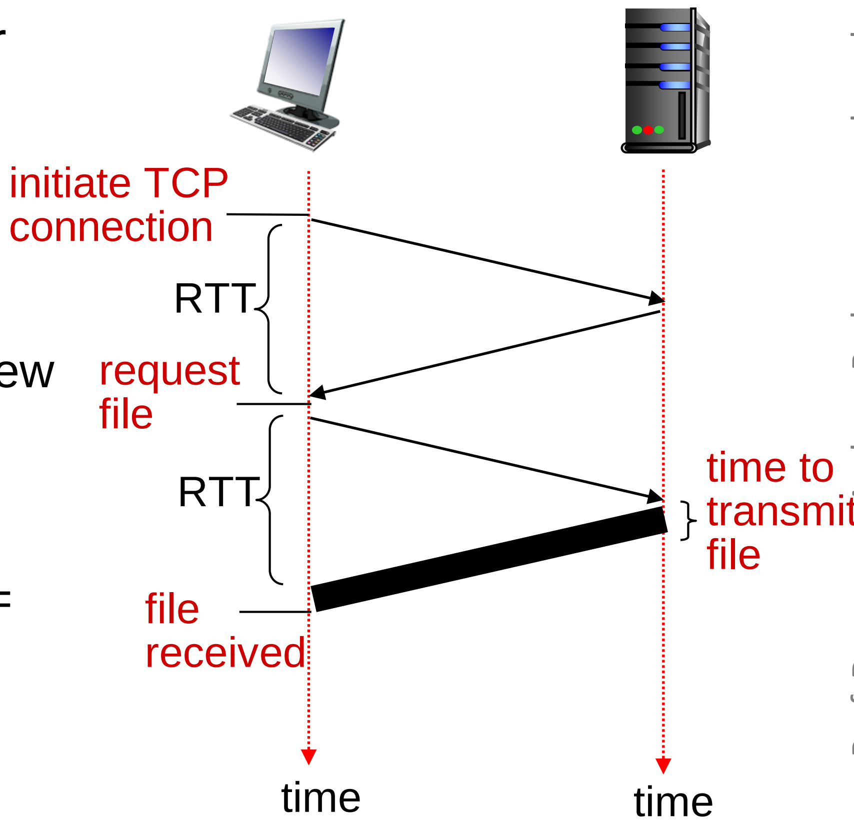
5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

6. Steps 1-5 repeated for each of 10 jpeg objects

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Non-persistent HTTP: response time

- RTT (round trip time): Time for a small packet to travel from client to server and back
- HTTP response time:
  - One RTT to initiate TCP connection
  - One RTT for HTTP request and first few bytes of HTTP response to return
  - File transmission time
  - Non-persistent HTTP response time = 2RTT+ file transmission time



Persistent HTTP

- Non-persistent HTTP issues:
  - Requires 2 RTTs per object
  - OS overhead for each TCP connection
  - Browsers often open parallel TCP connections to fetch referenced objects
- Persistent HTTP:
  - Server leaves connection open after sending response
  - Subsequent HTTP messages between same client/server sent over open connection
  - Client sends requests as soon as it encounters a referenced object

Persistent HTTP

Persistent HTTP **without** pipelining:

- Client sends new request for new objects only, if the response of the previous request has already arrived.
- One RTT for each object.

Persistent **with** pipelining:

- Standard in HTTP/1.1.
- Client sends requests as soon as a reference to an object is found.
- Ideally only a little more than a single RTT for the loading of all referenced objects.

HTTP request message

- Two types of HTTP messages: request, response
- HTTP request message:
  - ASCII (human-readable format)

carriage return character

line-feed character

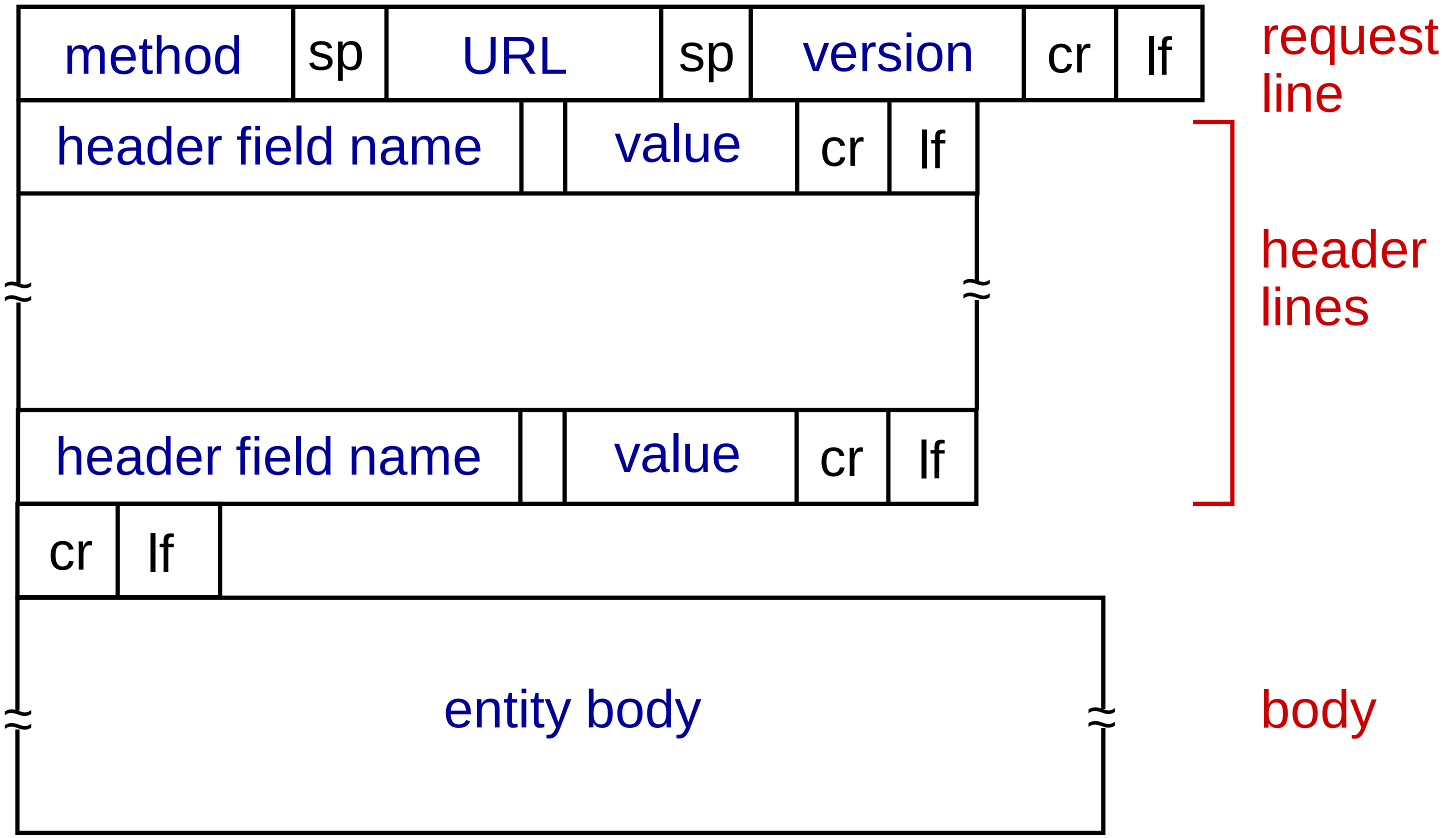
request line (GET, POST, HEAD commands)

header lines

carriage return, line feed at start of line indicates end of header lines

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

HTTP request message: general format



Uploading form input

- POST method:
  - Web page often includes form input
  - Input is uploaded to server in entity body
- URL method:
  - Uses GET method
  - Input is uploaded in URL field of request line:
    - www.somesite.com/animalsearch?monkeys&banana

Method types

- HTTP/1.0:
  - GET
  - POST
  - HEAD
    - Asks server to leave requested object out of response
- HTTP/1.1:
  - GET, POST, HEAD
  - PUT
    - Uploads file in entity body to path specified in URL field
  - DELETE
    - Deletes file specified in the URL field

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

HTTP response message

status line (protocol status code status phrase)

header lines

data, e.g., requested HTML file

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
   GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
   1\r\n
\r\n
data data data data data ...
```

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

HTTP response status codes

- Status code appears in 1st line in server-to-client response message.
  - 200 OK
    - Request succeeded, requested object later in this msg
  - 301 Moved Permanently
    - Requested object moved, new location specified later in this msg (Location:)
  - 400 Bad Request
    - Request msg not understood by server
  - 404 Not Found
    - Requested document not found on this server
  - 505 HTTP Version Not Supported

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Trying out HTTP (client side) for yourself

- Telnet to your favorite Web server:

```
telnet cis.poly.edu 80
```

opens TCP connection to port 80 (default HTTP server port) at cis.poly.edu. anything typed in sent to port 80 at cis.poly.edu

- Request web-page:

```
GET /~ross/ HTTP/1.1
Host: cis.poly.edu
```

by typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

- Look at response message sent by HTTP server!

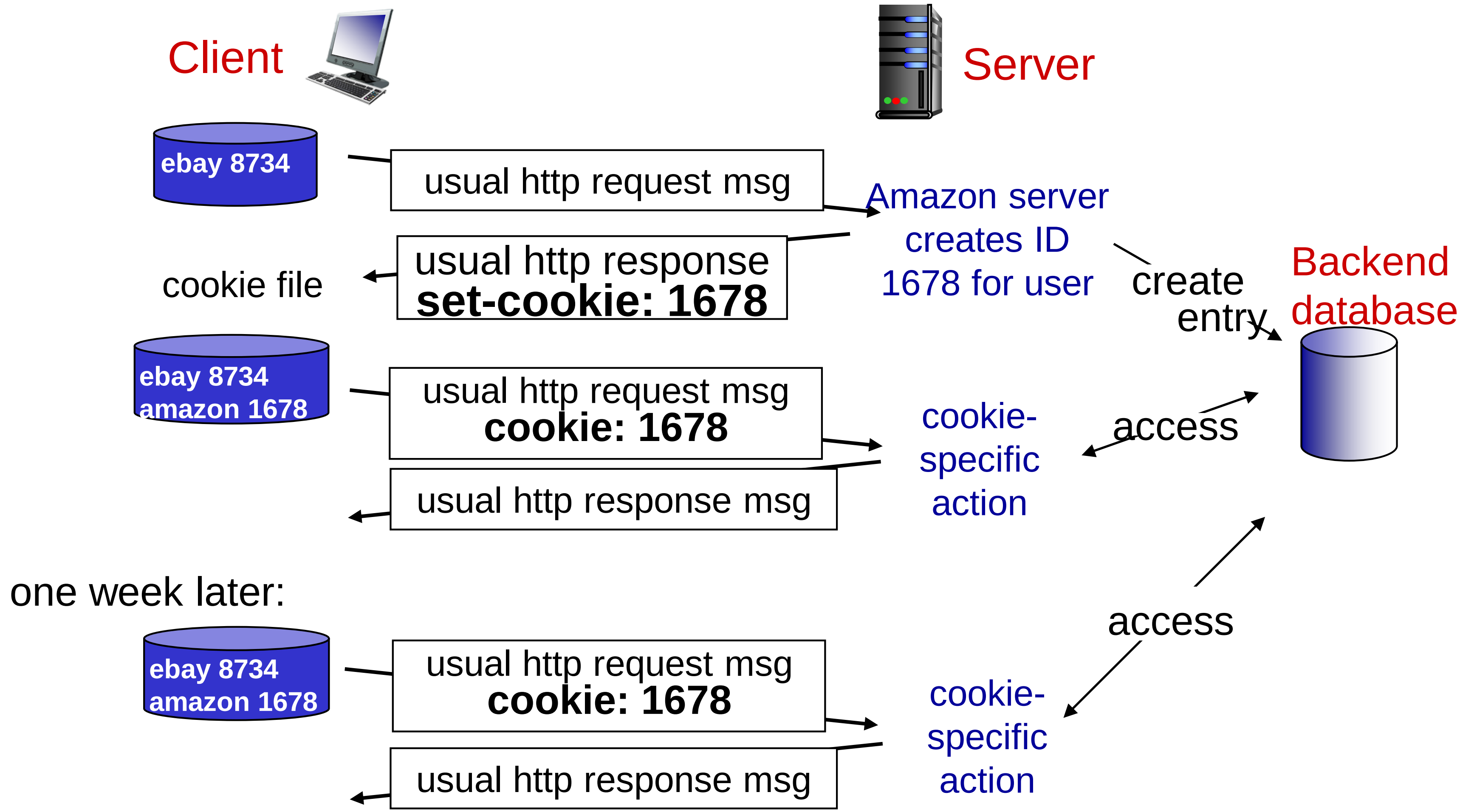
(or use Wireshark to look at captured HTTP request/response)

User-server state: cookies

- Many web sites use cookies
- Four components:
  - Cookie header line of HTTP response message
  - Cookie header line in next HTTP request message
  - Cookie file kept on user's host, managed by user's browser
  - Back-end database at web site

- Example:
  - Susan always accesses Internet from same PC
  - Visits specific e-commerce site for first time
  - When initial HTTP requests arrives at site, site creates:
    - Unique ID
    - Entry in backend database for ID

Cookies: keeping "state" (cont.)



Cookies (continued)

- What cookies can be used for:
  - Authorization
  - Shopping carts
  - Recommendations
  - User session state (web e-mail)
- How to keep "state":
  - Protocol endpoints: maintain state at sender/receiver over multiple transactions
  - Cookies: HTTP messages carry state

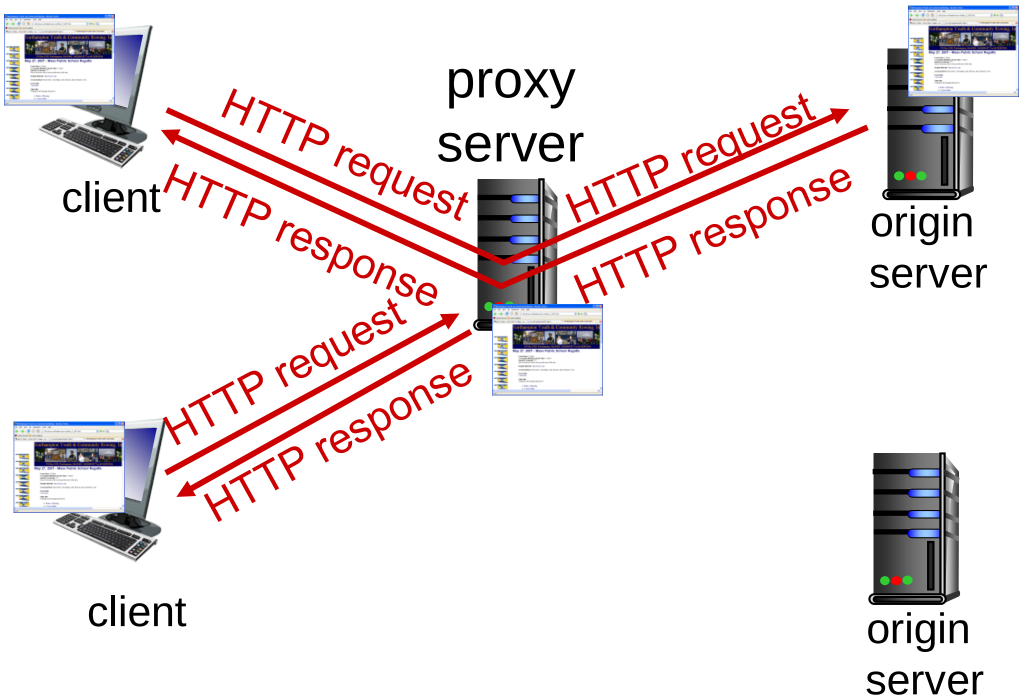
aside

Cookies and privacy:

- Cookies permit sites to learn a lot about you
- You may supply name and e-mail to sites

Web caches (proxy server)

- Goal: satisfy client request without involving origin server
  - User sets browser: Web accesses via cache
  - Browser sends all HTTP requests to cache
    - Object in cache: cache returns object
    - Else cache requests object from origin server, then returns object to client



More about Web caching

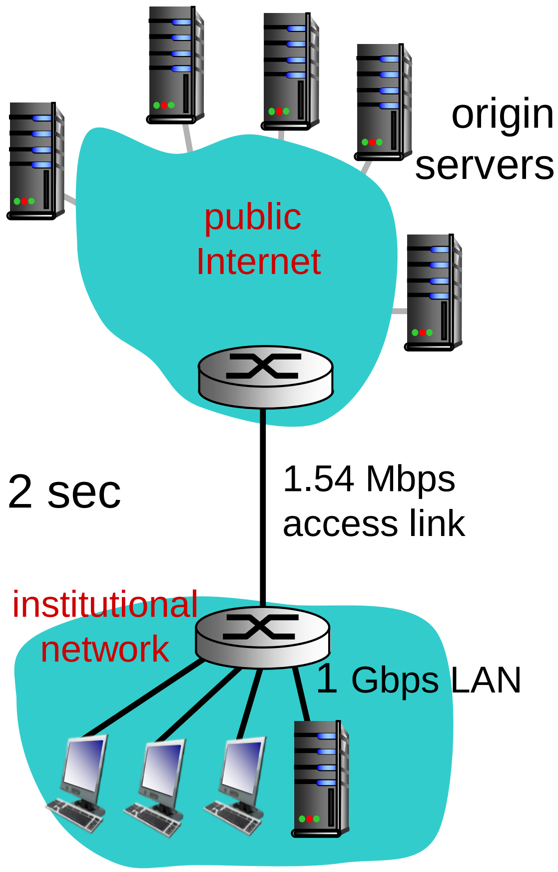
- Cache acts as both client and server
  - Server for original requesting client
  - Client to origin server
- Typically cache is installed by ISP (university, company, residential ISP)

- Why web caching?
  - Reduce response time for client request
  - Reduce traffic on an institution's access link
  - Internet dense with caches: enables "poor" content providers to effectively deliver content (so too does P2P file sharing)

Caching example

- Assumptions
  - AVG object size: 100K bits
  - AVG request rate from browsers to origin servers:15/sec
  - AVG data rate to browsers: 1.50 Mbps
  - RTT from institutional router to any origin server: 2 sec
  - access link rate: 1.54 Mbps
- Consequences:
  - LAN utilization: 15%
  - Access link utilization = 99% *problem!*
  - Total delay = Internet delay + access delay + LAN delay

    = 2 sec + minutes + msecs

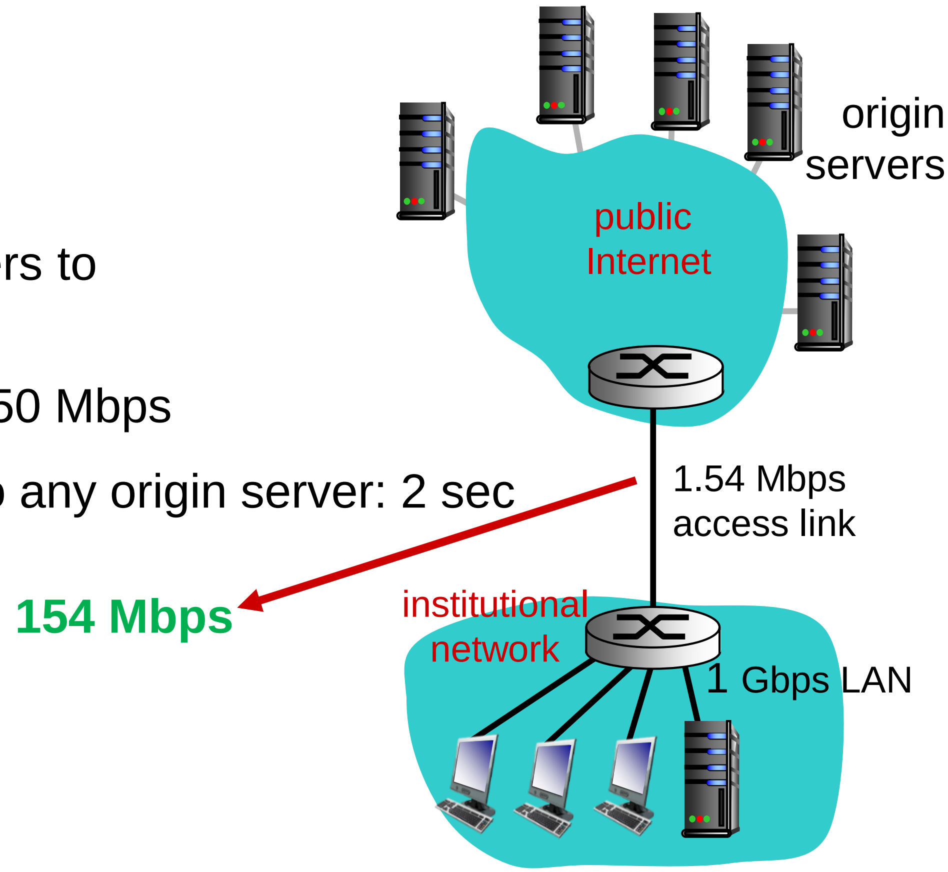


Caching example:
Fatter access link

- Assumptions
  - AVG object size: 100K bits
  - AVG request rate from browsers to origin servers:15/sec
  - AVG data rate to browsers: 1.50 Mbps
  - RTT from institutional router to any origin server: 2 sec
  - access link rate: 1.54 Mbps
- Consequences:
  - LAN utilization: 15%
  - Access link utilization = 9,9%
  - Total delay = Internet delay + access delay + LAN delay

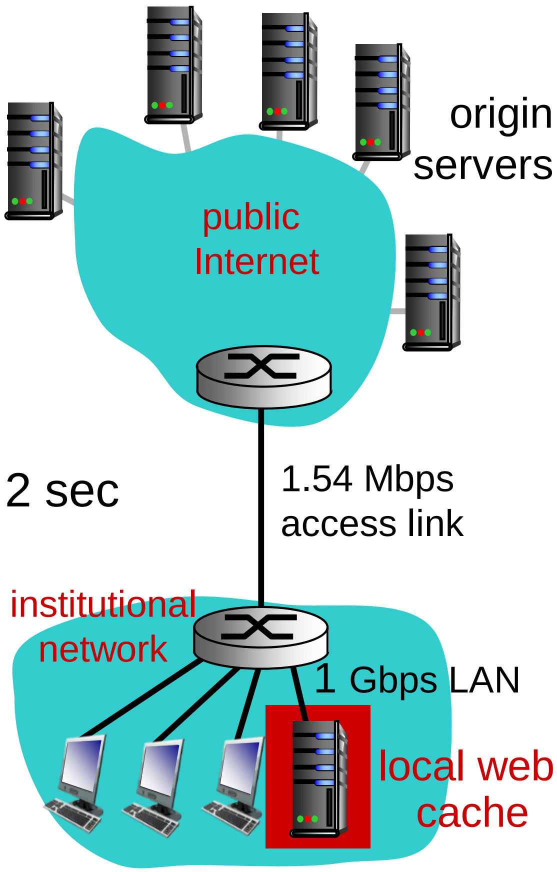
= 2 sec + msecs + msecs

Cost: increased access link speed (not cheap!)



Caching example: Install local cache

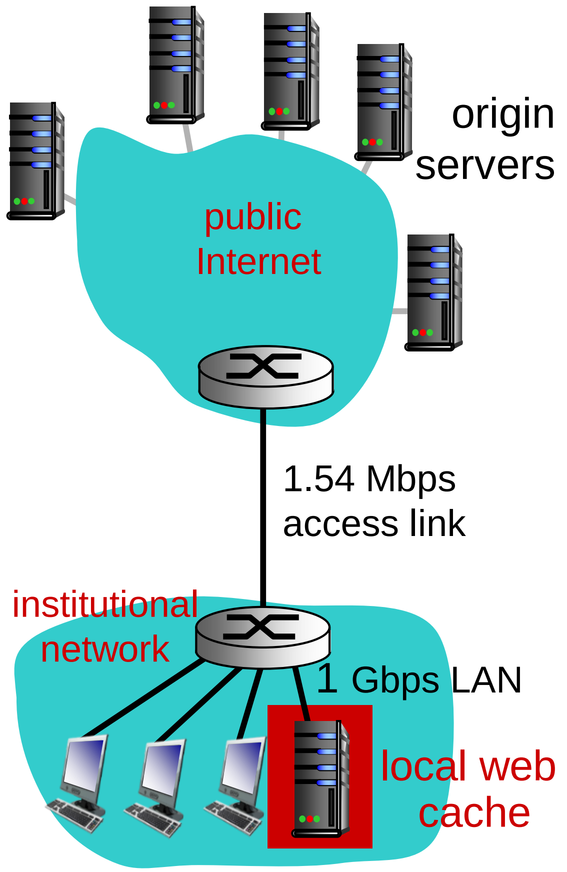
- Assumptions
  - AVG object size: 100K bits
  - AVG request rate from browsers to origin servers:15/sec
  - AVG data rate to browsers: 1.50 Mbps
  - RTT from institutional router to any origin server: 2 sec
  - access link rate: 1.54 Mbps
- Consequences:
  - LAN utilization: ?%
  - Access link utilization = ?%
  - Total delay = ?



Cost: web cache (cheap!)

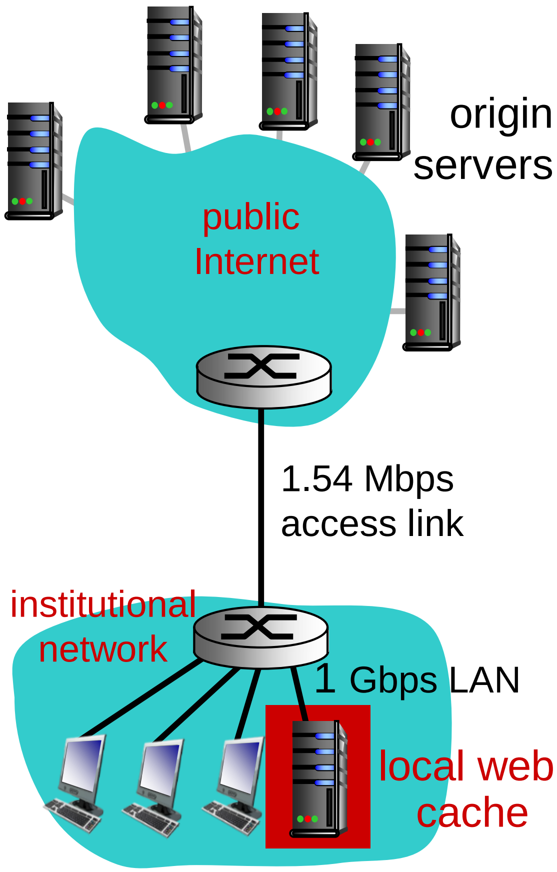
Caching example: Install local cache

- Calculating access link utilization, delay with cache:
  - Suppose cache hit rate is 0.4
    - 40% requests satisfied at cache
  - Access link utilization:
    - 60% of requests use access link
  - Data rate to browsers over access link = 0.6 * 1.50 Mbps = 0.9 Mbps
    - utilization = 0.9/1.54 = **58%**



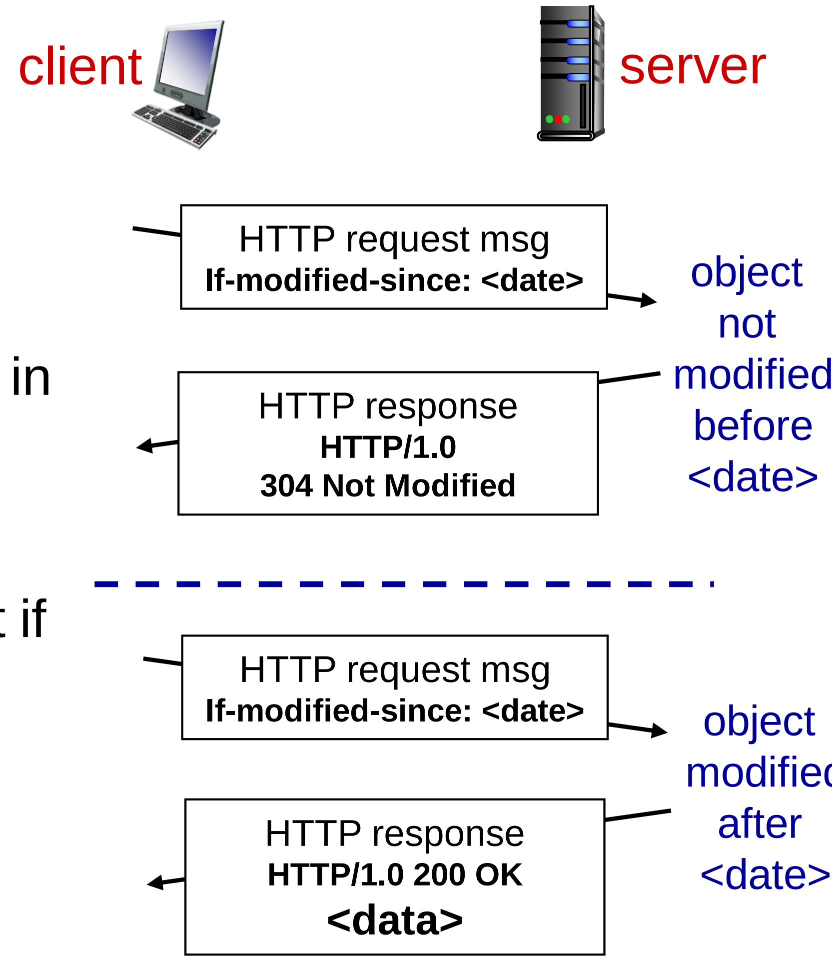
Caching example: Install local cache

- Calculating access link utilization, delay with cache:
  - Total delay
    - = 0.6 * (delay from origin servers) + 0.4 * (delay when satisfied at cache)
    - = 0.6 (2.01) + 0.4 (~msecs)
    - = ~ 1.2 secs
    - Less than with 154 Mbps link (and cheaper too!)



Conditional GET

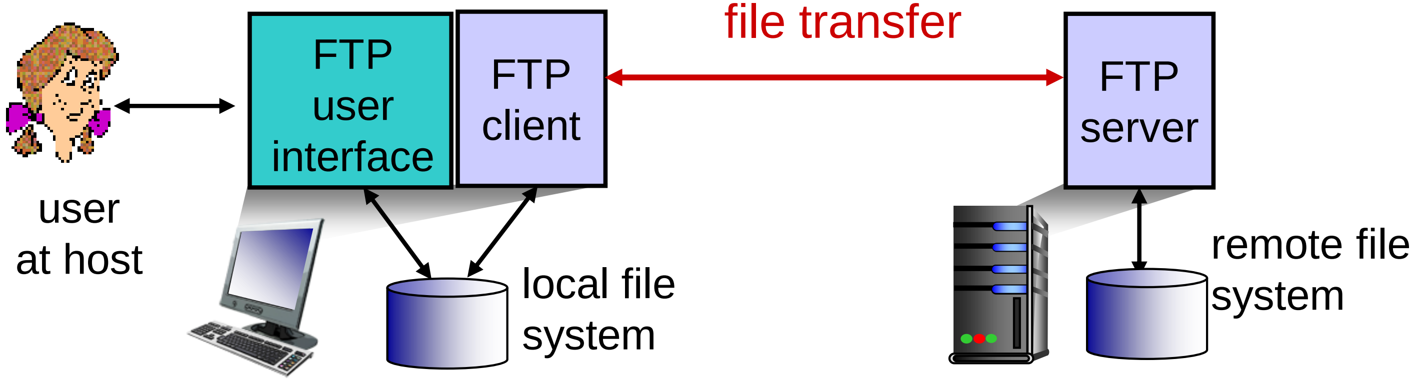
- Goal: don’t send object if cache has up-to-date cached version
  - No object transmission delay
  - Lower link utilization
- Cache: specify date of cached copy in HTTP request
  - If-modified-since: <date>
- Server: response contains no object if cached copy is up-to-date:
  - HTTP/1.0 304 not modified



Chapter 2: Roadmap

- Principles of network applications
- Web and HTTP
- FTP
- Electronic mail
  - SMTP, POP3, IMAP
- DNS
- P2P Applications
- Socket programming with UDP and TCP
- Web service

FTP: the file transfer protocol

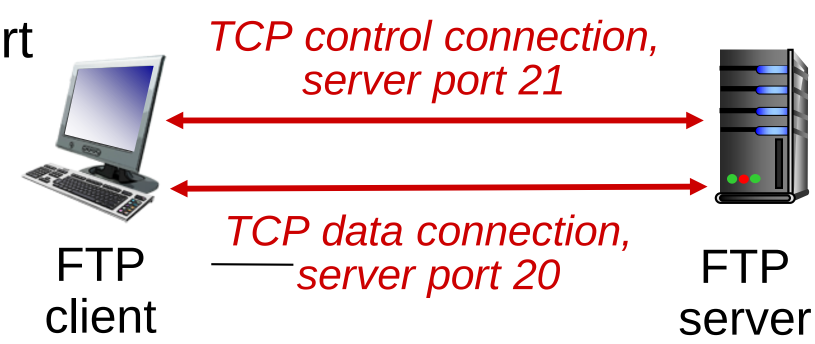


- Transfer file to/from remote host
- Client/server model
  - client: side that initiates transfer (either to/from remote)
  - server: remote host
- FTP: RFC 959
- FTP server: port 21

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

FTP: separate control, data connections

- FTP client contacts FTP server at port 21, using TCP
- Client authorized over control connection
- Client browses remote directory, sends commands over control connection
- When server receives file transfer command, server opens 2nd TCP data connection (for file) to client
- After transferring one file, server closes data connection



- Server opens another TCP data connection to transfer another file
- Control connection: “out of band”
- FTP server maintains “state”: current directory, earlier authentication

FTP commands, responses

- Sample commands
  - Sent as ASCII text over control channel
  - USER username
  - PASS password
  - LIST return list of file in current directory
  - RETR filename retrieves (gets) file
  - STOR filename stores (puts) file onto remote host
- Sample return codes
  - Status code and phrase (as in HTTP)
  - 331 username OK, password required
  - 125 data connection already open; transfer starting
  - 425 can't open data connection
  - 452 error writing file

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

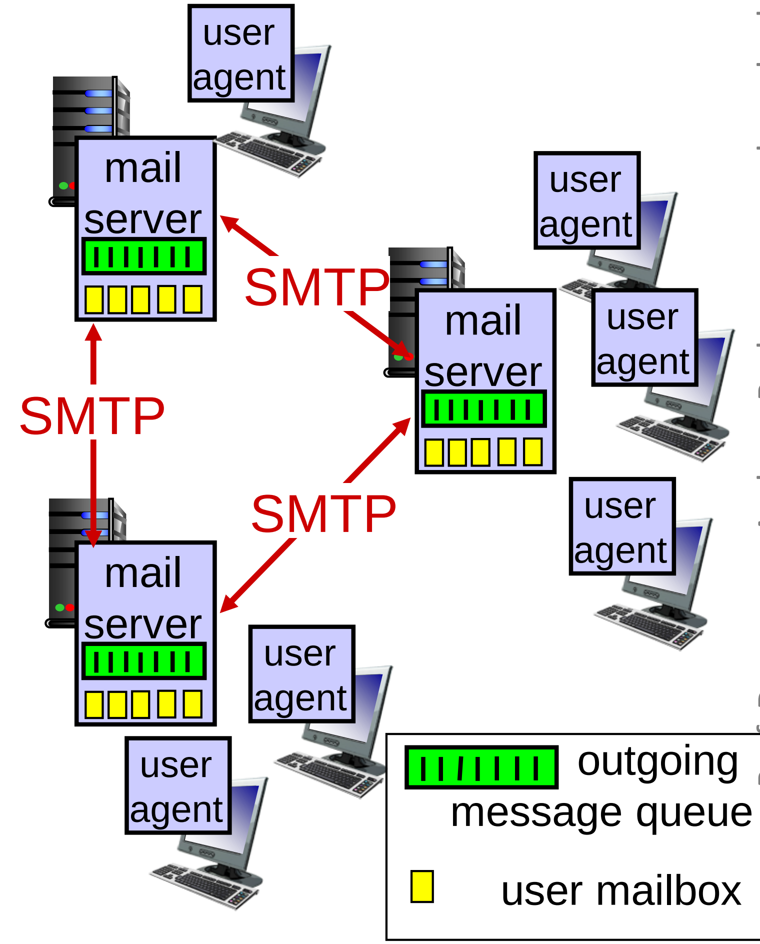
Chapter 2: Roadmap

- Principles of network applications
- Web and HTTP
- FTP
- Electronic mail
  - SMTP, POP3, IMAP
- DNS
- P2P Applications
- Socket programming with UDP and TCP

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

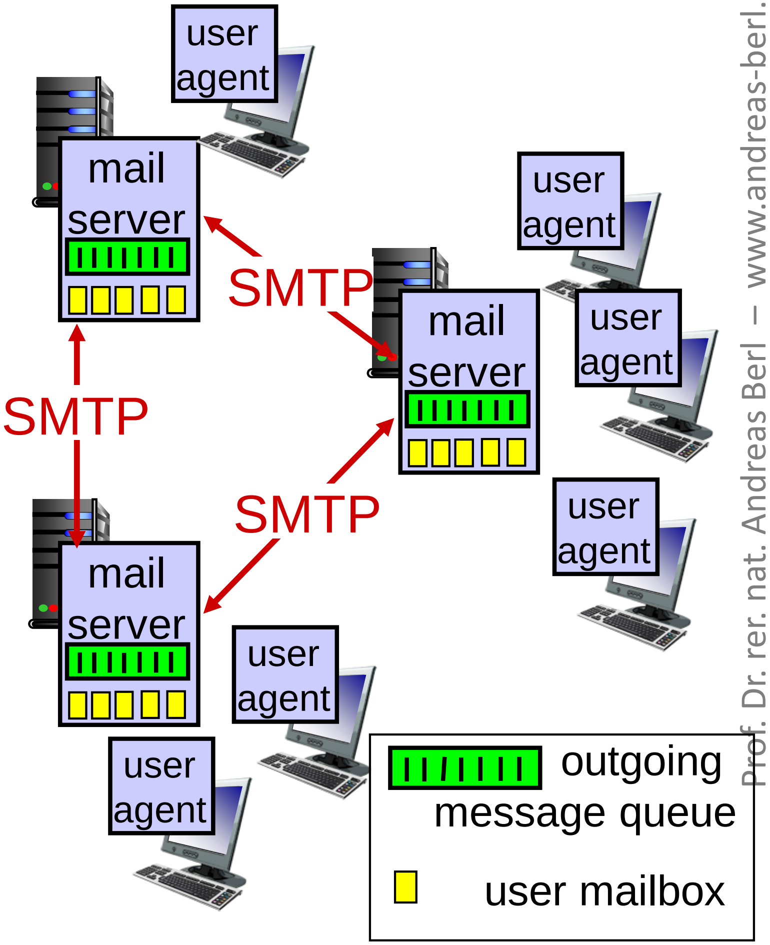
Electronic mail

- Three major components:
  - User agents
  - Mail servers
  - Simple mail transfer protocol: SMTP
- User Agent
  - A.k.a. “mail reader”
  - Composing, editing, reading mail messages
  - E.g., Outlook, Thunderbird, iPhone mail client
  - Outgoing, incoming messages stored on server



Electronic mail: Mail servers

- Mail servers:
  - Mailbox contains incoming messages for user
  - Message queue of outgoing (to be sent) mail messages
  - SMTP protocol between mail servers to send email messages
    - “Client”: sending mail server
    - “Server”: receiving mail server

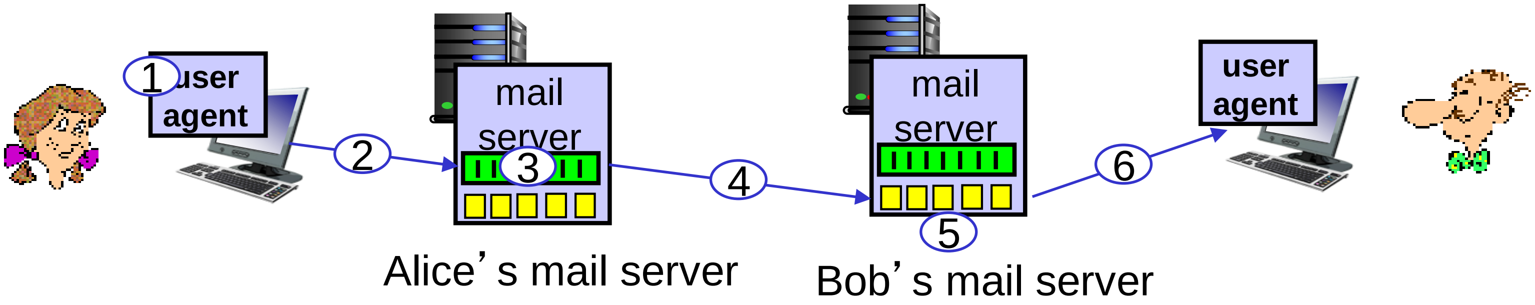


Electronic Mail: SMTP [RFC 2821]

- Uses TCP to reliably transfer email message from client to server, port 25
- Direct transfer: sending server to receiving server
- Three phases of transfer
  - Handshaking (greeting)
  - Transfer of messages
  - Closure
- Command/response interaction (like HTTP, FTP)
  - Commands: ASCII text
  - Response: status code and phrase
- Messages must be in 7-bit ASCI

Scenario: Alice sends message to Bob

1) Alice uses UA to compose message "to" bob@someschool.edu
2) Alice's UA sends message to her mail server; message placed in message queue
3) Client side of SMTP opens TCP connection with Bob's mail server
4) SMTP client sends Alice's message over the TCP connection
5) Bob's mail server places the message in Bob's mailbox
6) Bob invokes his user agent to read message



Sample SMTP interaction

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

Try SMTP interaction for yourself

- Telnet **`servername`** 25
  - Or: use OpenSSL for secured connections
- See 220 reply from server
- Enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

Above lets you send email without using email client (reader)

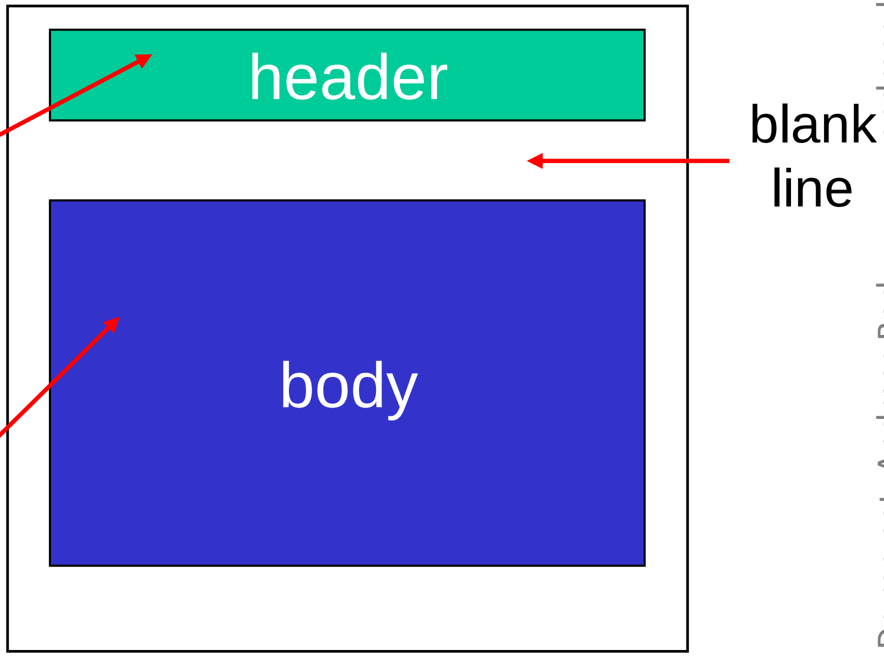
SMTP: final words

- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses `CRLF.CRLF` to determine end of message
- Comparison with HTTP
  - HTTP: pull
  - SMTP: push
  - Both have ASCII command/response interaction, status codes
  - HTTP: Each object encapsulated in its own response msg
  - SMTP: Multiple objects sent in multipart msg

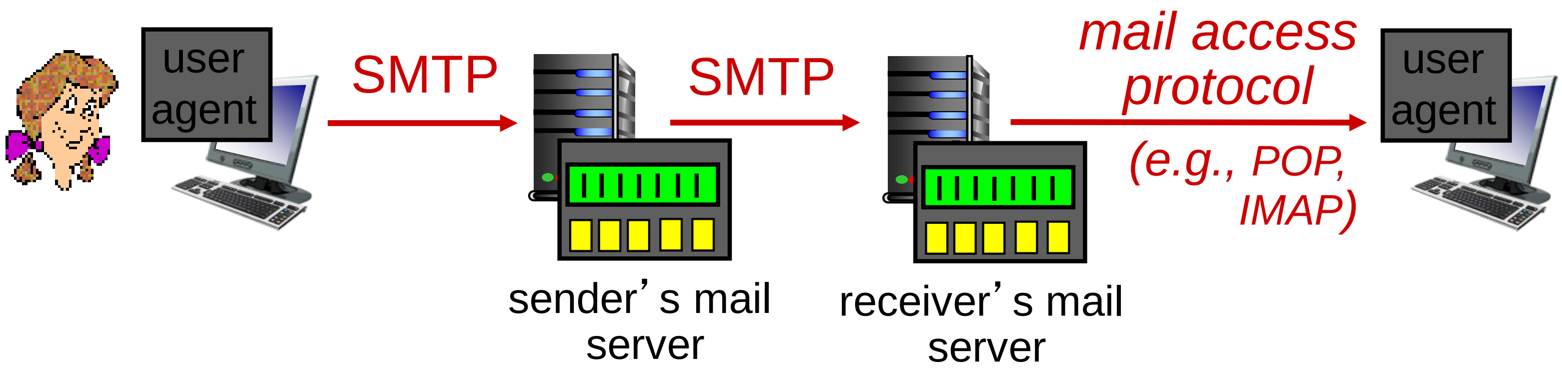
Mail message format

- SMTP: protocol for exchanging email msgs
- RFC 822: standard for text message format:
  - Header lines, e.g.,
    - To:
    - From:
    - Subject:

    Different from SMTP MAIL FROM, RCPT TO: commands!
  - Body: the "message"
    - ASCII characters only



Mail access protocols



- SMTP: delivery/storage to receiver's server
- Mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]: authorization, download
  - IMAP: Internet Mail Access Protocol [RFC 1730]: more features, including manipulation of stored msgs on server
  - HTTP: Gmail, Hotmail, Yahoo! Mail, etc.

POP3 protocol

- Authorization phase
  - Client commands:
    - **user**: declare username
    - **pass**: password
  - Server responses
    - **+OK**
    - **-ERR**
- Transaction phase, client:
  - **list**: List message numbers
  - **retr**: Retrieve message by number
  - **dele**: Delete
  - **quit**

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on

C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

POP3 (more) and IMAP

- More about POP3
  - Previous example uses POP3 “download and delete” mode
    - Bob cannot re-read e-mail if he changes client
  - POP3 “download-and-keep”: copies of messages on different clients
  - POP3 is stateless across sessions
- IMAP
  - Keeps all messages in one place: at server
  - Allows user to organize messages in folders
  - Keeps user state across sessions:
    - Names of folders and mappings between message ids and folder name

Chapter 2: Roadmap

- Principles of network applications
- Web and HTTP
- FTP
- Electronic mail
  - SMTP, POP3, IMAP
- DNS
- P2P Applications
- Socket programming with UDP and TCP
- Web service

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

DNS: domain name system

- People: many identifiers:
  - SSN, name, passport #
- Internet hosts, routers:
  - IP address (32 bit) - used for addressing datagrams
  - "Name", e.g., www.yahoo.com - used by humans
- *Q:* How to map between IP address and name, and vice versa ?

- Domain name system:
  - Distributed database implemented in hierarchy of many name servers
  - Application-layer protocol: hosts, name servers communicate to resolve names (address/name translation)
    - Note: core internet function, implemented as application-layer protocol
    - Complexity at network's "edge"

DNS: services, structure

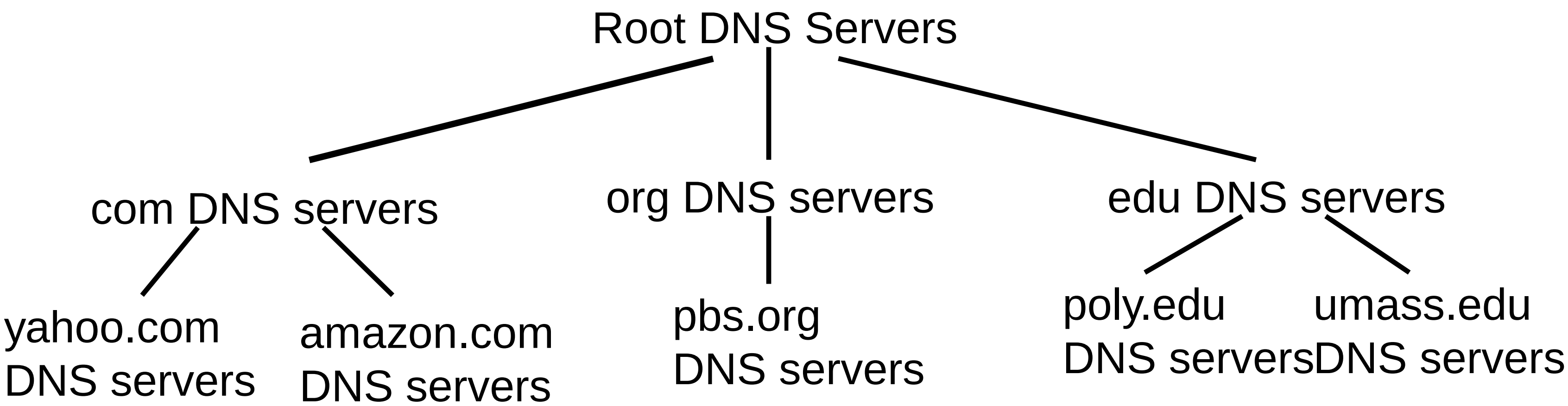
- DNS services
  - Hostname to IP address translation
  - Host aliasing
    - Canonical, alias names
  - Mail server aliasing
  - Load distribution
    - Replicated web servers: many IP addresses correspond to one name

Why not centralize DNS?

- Single point of failure
- Traffic volume
- Distant centralized database
- Maintenance

A: Doesn't scale!

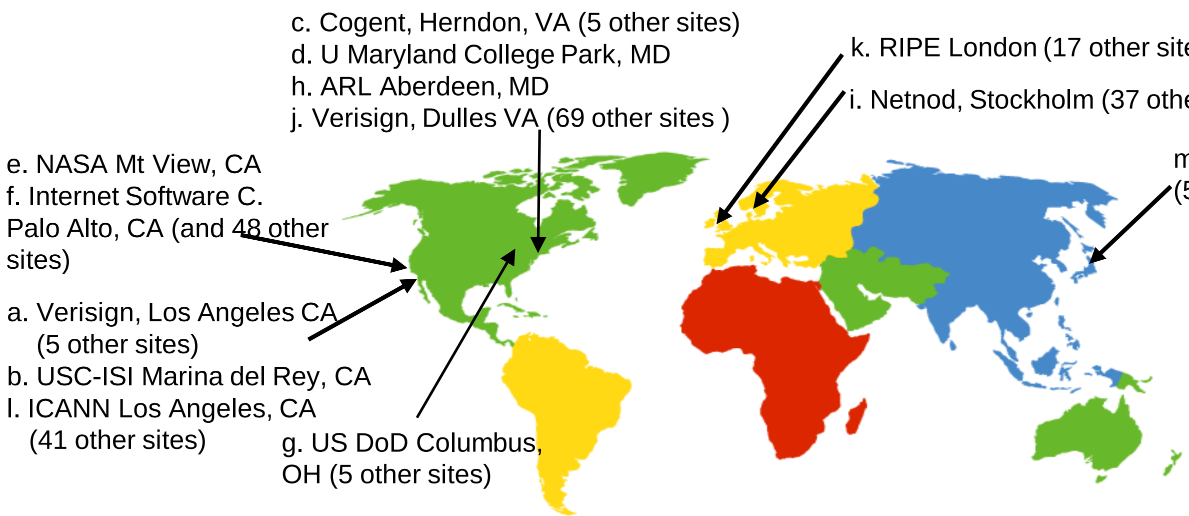
DNS: a distributed, hierarchical database



- Client wants IP for www.amazon.com; 1st approx:
  - Client queries root server to find .com DNS server
  - Client queries .com DNS server to get amazon.com DNS server
  - Client queries amazon.com DNS server to get IP address for www.amazon.com

DNS: root name servers

- Contacted by local name server that can not resolve name
- Root name server:
  - Contacts authoritative name server if name mapping not known
  - Gets mapping
  - Returns mapping to local name server



13 root name "servers" worldwide

TLD, authoritative servers

- Top-level domain (TLD) servers:
  - Responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g.: **UK, FR, CA, JP, DE**
  - Network solutions maintains servers for .Com TLD
  - Educause for .Edu TLD
- Authoritative DNS servers:
  - Organization’s own DNS server(s), providing authoritative hostname to IP mappings for organization’s named hosts
  - Can be maintained by organization or service provider

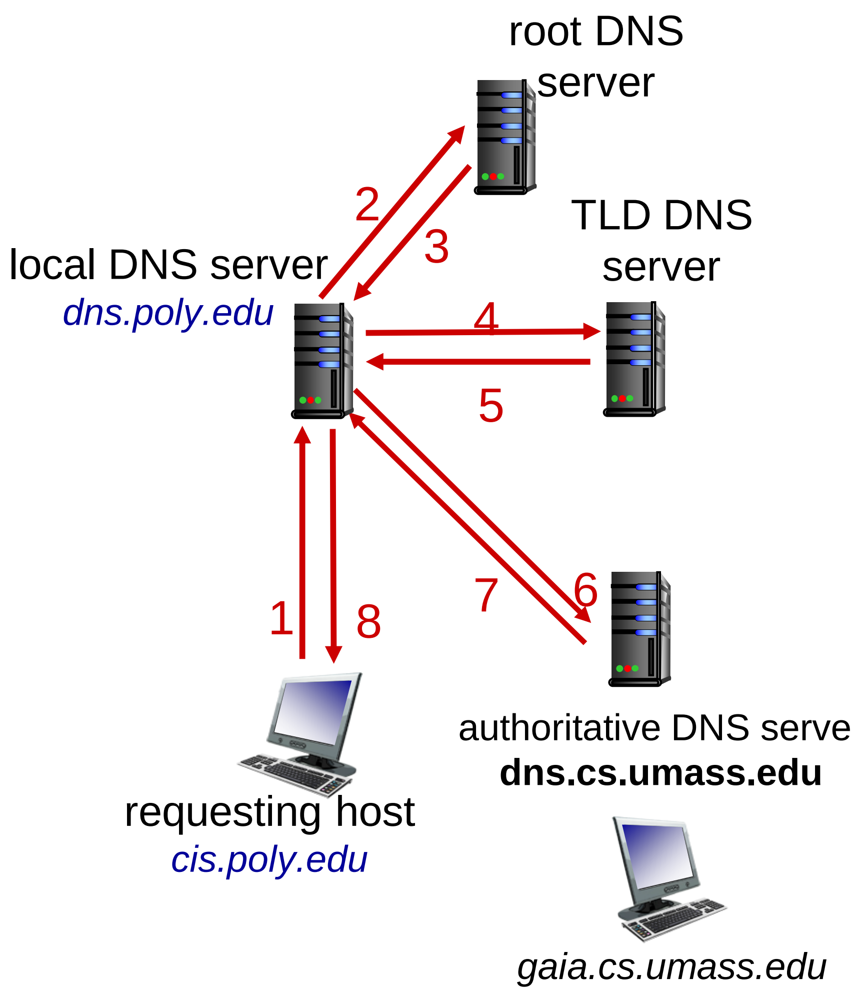
Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Local DNS name server

- Does not strictly belong to hierarchy
- Each ISP (residential ISP, company, university) has one
  - Also called "default name server"
- When host makes DNS query, query is sent to its local DNS server
  - Has local cache of recent name-to-address translation pairs (but may be out of date!)
  - Acts as proxy, forwards query into hierarchy

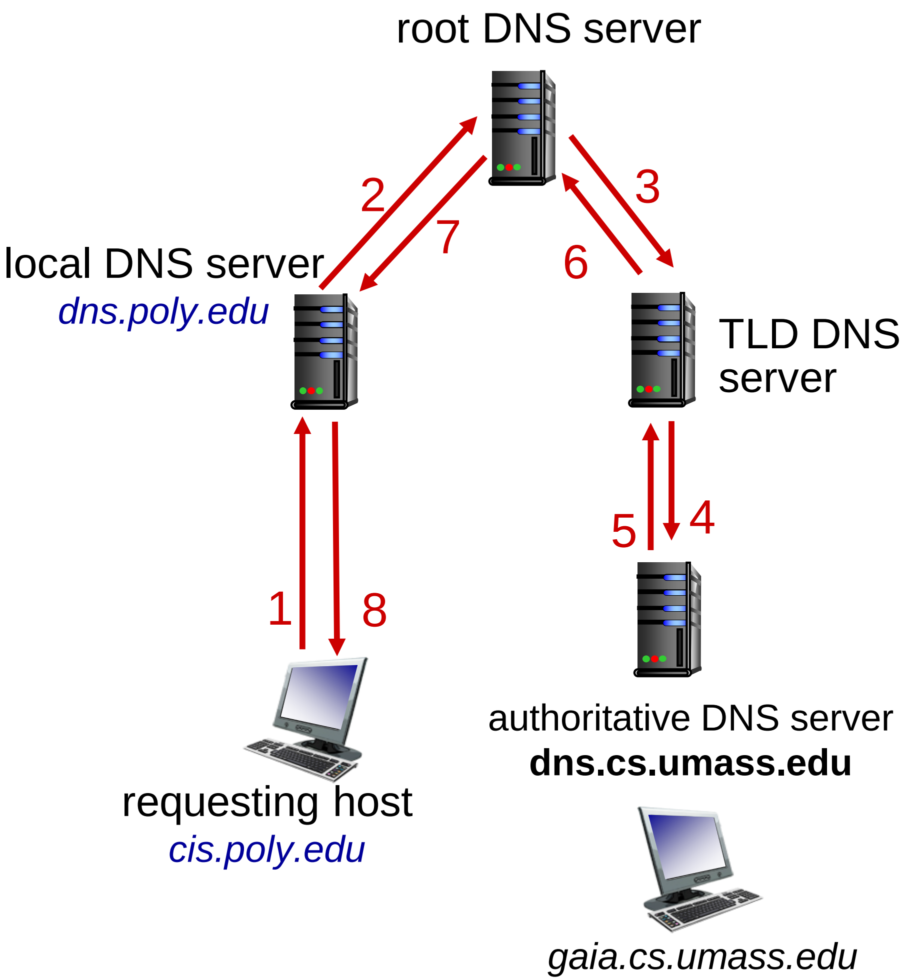
DNS name resolution example

- Host at cis.Poly.Edu wants IP address for gaia.Cs.Umass.Edu
- Iterated query
  - Contacted server replies with name of server to contact
  - “I don’t know this name, but ask this server”



DNS name resolution example

- Recursive query
  - Puts burden of name resolution on contacted name server
  - Heavy load at upper levels of hierarchy?



DNS: caching, updating records

- Once (any) name server learns mapping, it caches mapping
  - Cache entries timeout (disappear) after some time (TTL)
  - TLD servers typically cached in local name servers
    - Thus root name servers not often visited
- Cached entries may be out-of-date (best effort name-to-address translation!)
  - If name host changes IP address, may not be known internet-wide until all TTLs expire
- Update/notify mechanisms proposed IETF standard
  - RFC 2136

DNS records

DNS: Distributed DB storing resource records (RR)

RR format: `(name, value, type, ttl)`

type=A

- **`name`** is hostname
- **`value`** is IP address

type=NS

- **`name`** is domain (e.g., foo.com)
- **`value`** is hostname of authoritative name server for this domain

type=CNAME

- **`name`** is alias name for some "canonical" (the real) name
- **`www.ibm.com`** is really **`servereast.backup2.ibm.com`**
- **`value`** is canonical name

type=MX

- **`value`** is name of mailserver associated with **`name`**

DNS protocol, messages

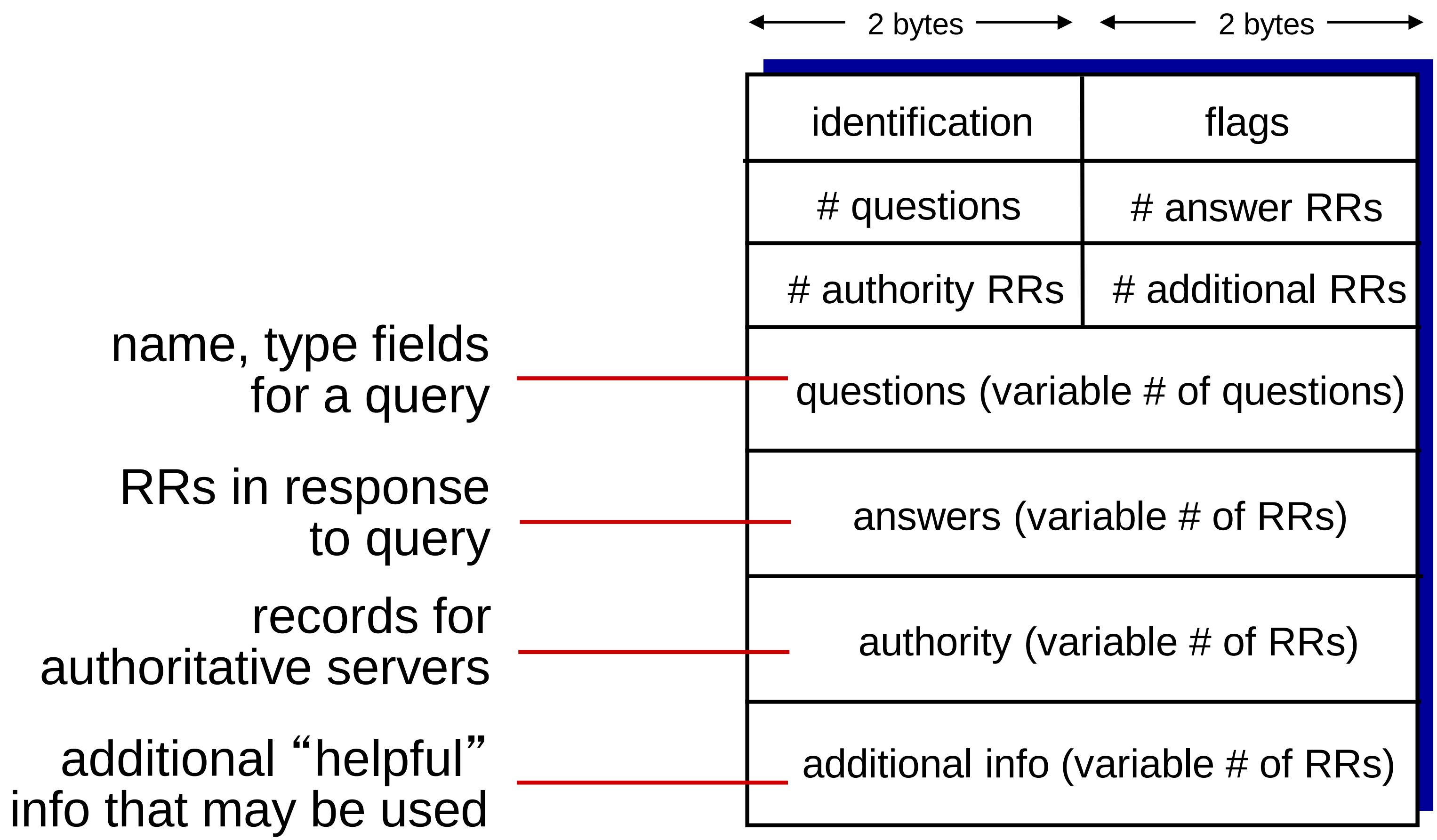
- Query and reply messages, both with same message format

- Msg header
  - Identification: 16 bit # for query, reply to query uses same #
  - Flags:
    - Query or reply
    - Recursion desired
    - Recursion available
    - Reply is authoritative

← 2 bytes →	← 2 bytes →
identification	flags
# questions	# answer RRs
# authority RRs	# additional RRs
questions (variable # of questions)	
answers (variable # of RRs)	
authority (variable # of RRs)	
additional info (variable # of RRs)	

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

DNS protocol, messages



Inserting records into DNS

- Example: new startup “Network Utopia”
- Register name networkuptopia.com at DNS registrar (e.g., Network Solutions)
  - Provide names, IP addresses of authoritative name server (primary and secondary)
  - Registrar inserts two RRs into .com TLD server: (networkutopia.com, dns1.networkutopia.com, NS)
    - (dns1.networkutopia.com, 212.212.212.1, A)
- Exercise:
  - Create authoritative server type A record for www.networkuptopia.com
  - Create type MX record for networkutopia.com

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Attacking DNS

- DDoS attacks
  - Bombard root servers with traffic
    - Not successful to date
    - Traffic Filtering
    - Local DNS servers cache IPs of TLD servers, allowing root server bypass
- Bombard TLD servers
  - Potentially more dangerous

- Redirect attacks
  - Man-in-middle
    - Intercept queries
  - DNS poisoning
    - Send bogus replies to DNS server, which caches
- Exploit DNS for DDoS
  - Send queries with spoofed source address: target IP

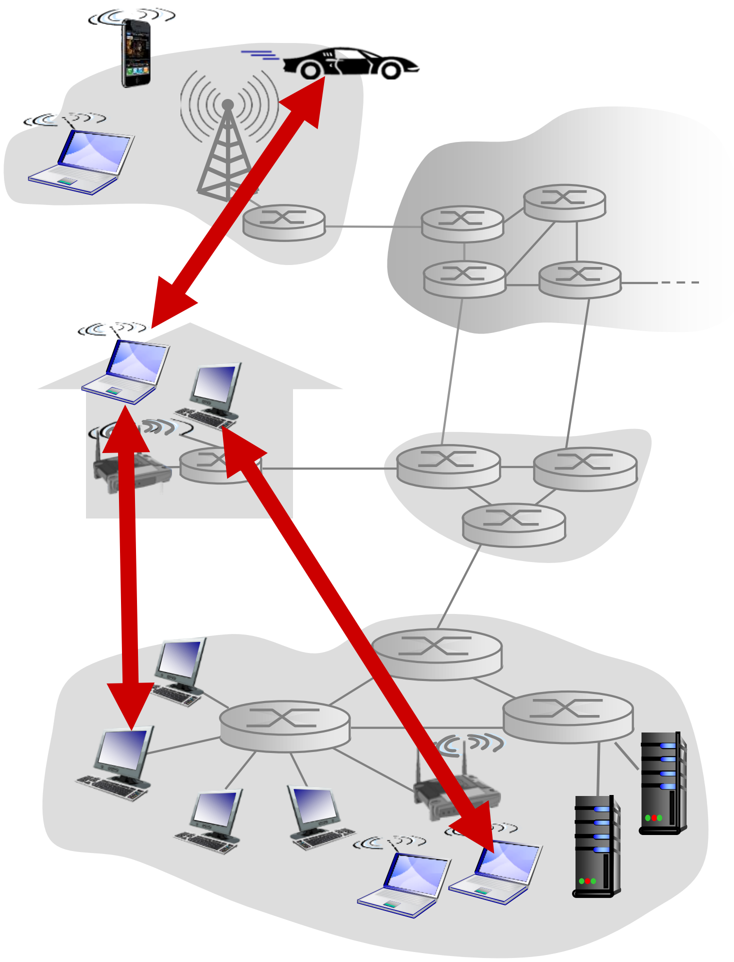
Chapter 2: Roadmap

- Principles of network applications
- Web and HTTP
- FTP
- Electronic mail
  - SMTP, POP3, IMAP
- DNS
- P2P Applications
- Socket programming with UDP and TCP

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

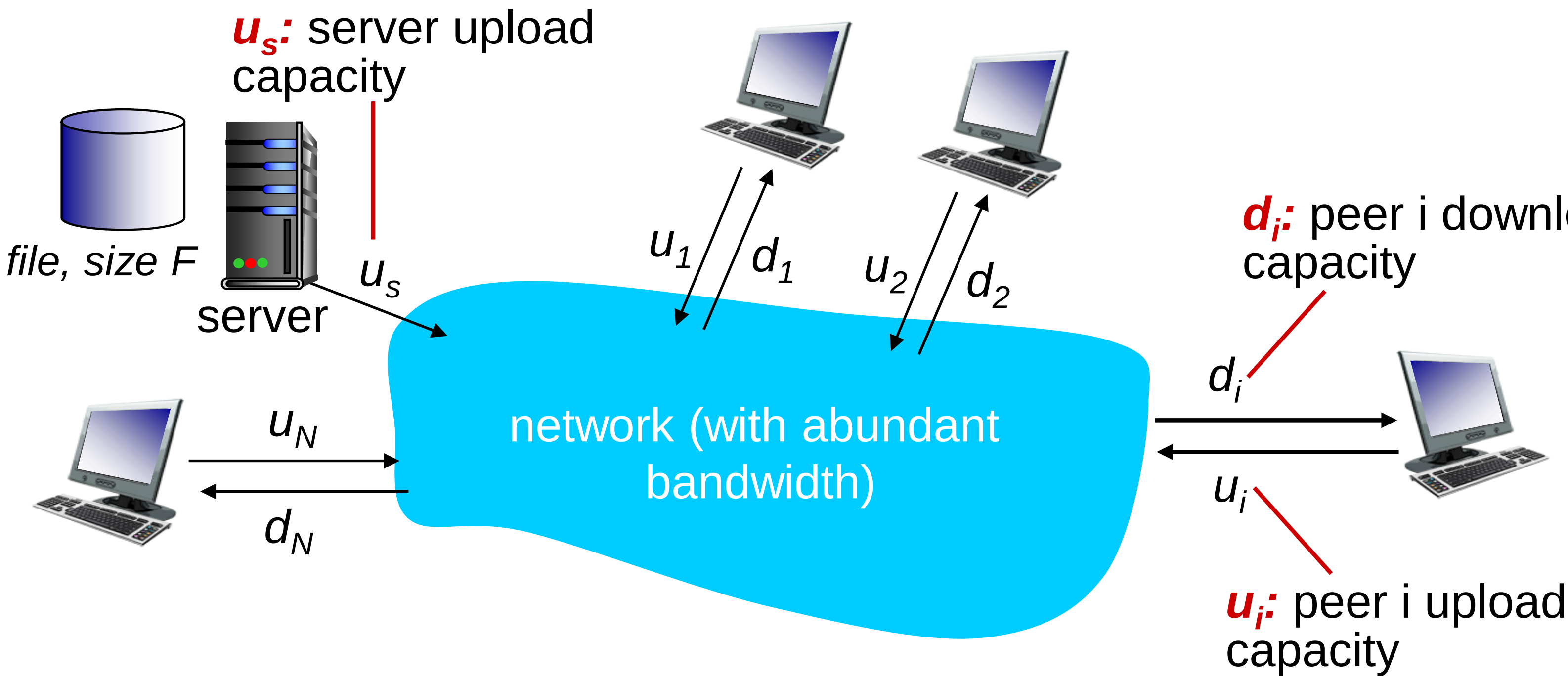
Pure P2P architecture

- No always-on server
- Arbitrary end systems directly communicate
- Peers are intermittently connected and change IP addresses
- Examples:
  - File distribution (BitTorrent)
  - Streaming (Kankan)
  - Voip (Skype)



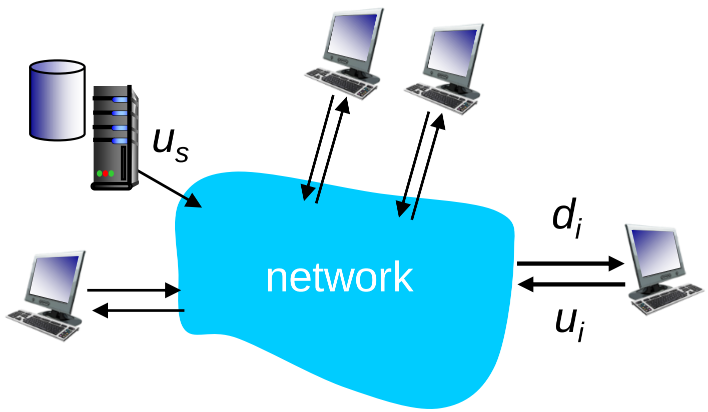
File distribution: Client-Server vs. P2P

- Question: How much time to distribute file (size F) from one server to N peers?
  - Peer upload/download capacity is limited resource



File distribution time: Client-server

- Server transmission: Must upload N file copies in parallel:
  - Time to send one copy: F/u_s
  - Time to send N copies: NF/u_s
- Client: Each client must download file copy
  - d_{min} = min client download rate
  - min client download time: F/d_{min}



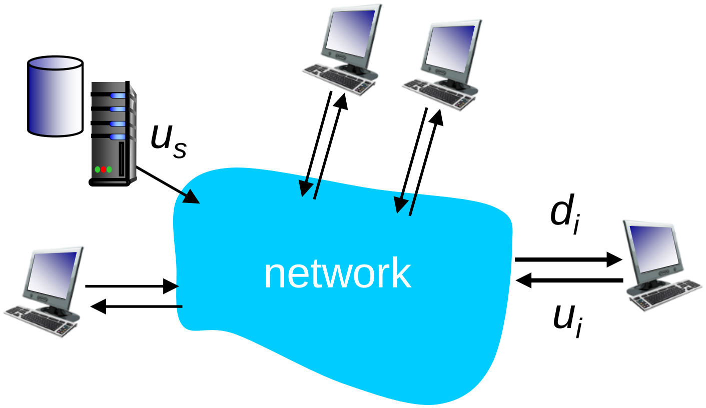
Time to distribute F to N clients using client-server approach

$$D_{c\text{-}s} > max\{NF/u_s, F/d_{min}\}$$

Increases linearly in N

File distribution time: P2P

- Server transmission: Must upload at least one copy
  - Time to send one copy: F/u_s
- Client: Each client must download file copy
  - Min client download time: F/d_{min}



- Clients: As aggregate must download NF bits
  - Max upload rate (limiting max Download rate) is $u_s + \sum u_i$

Time to distribute F to N clients using client-server approach

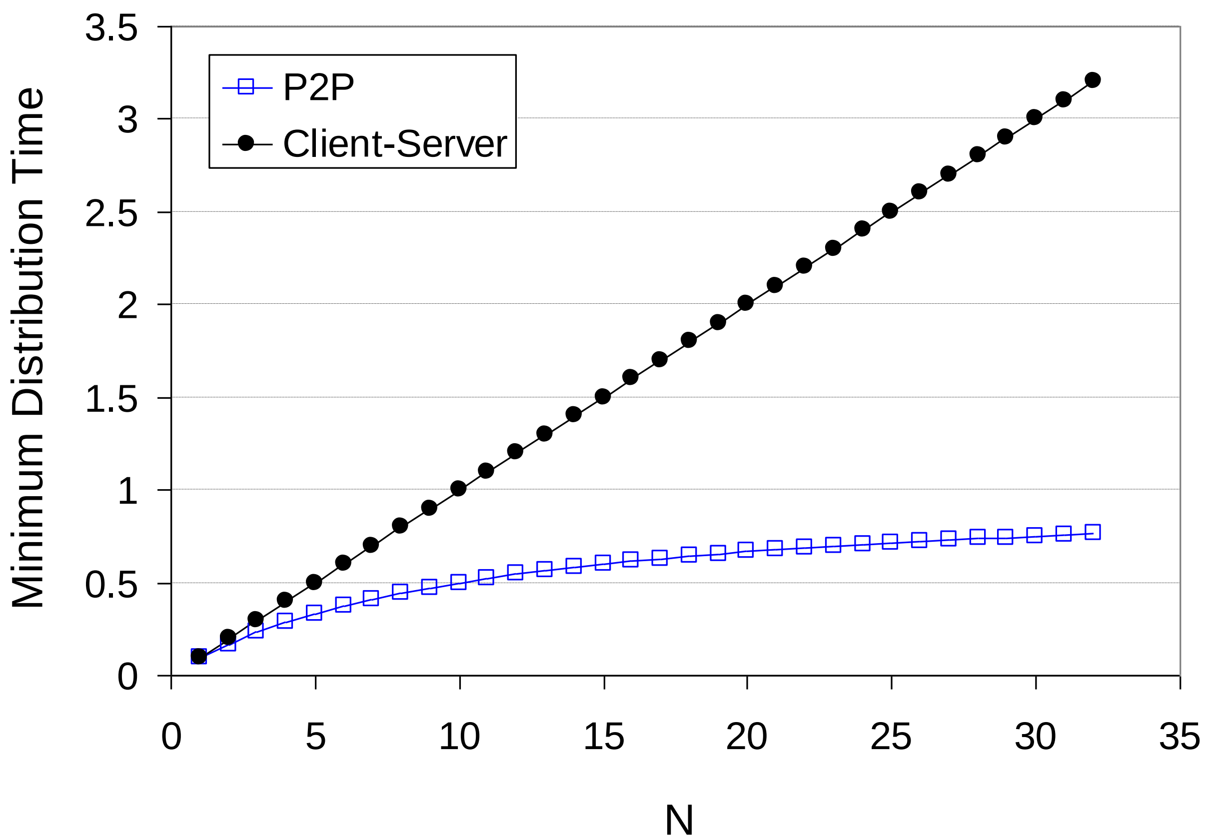
$$D_{P2P} > max\{F/u_s, F/d_{min}, NF/(u_s + \sum u_i)\}$$

Increases linearly in N

… but so does this, as each peer brings service capacity

Client-server vs. P2P: Example

- Client upload rate = u, F/u = 1 hour, u_s = 10u, $d_{min} \geq u_s$



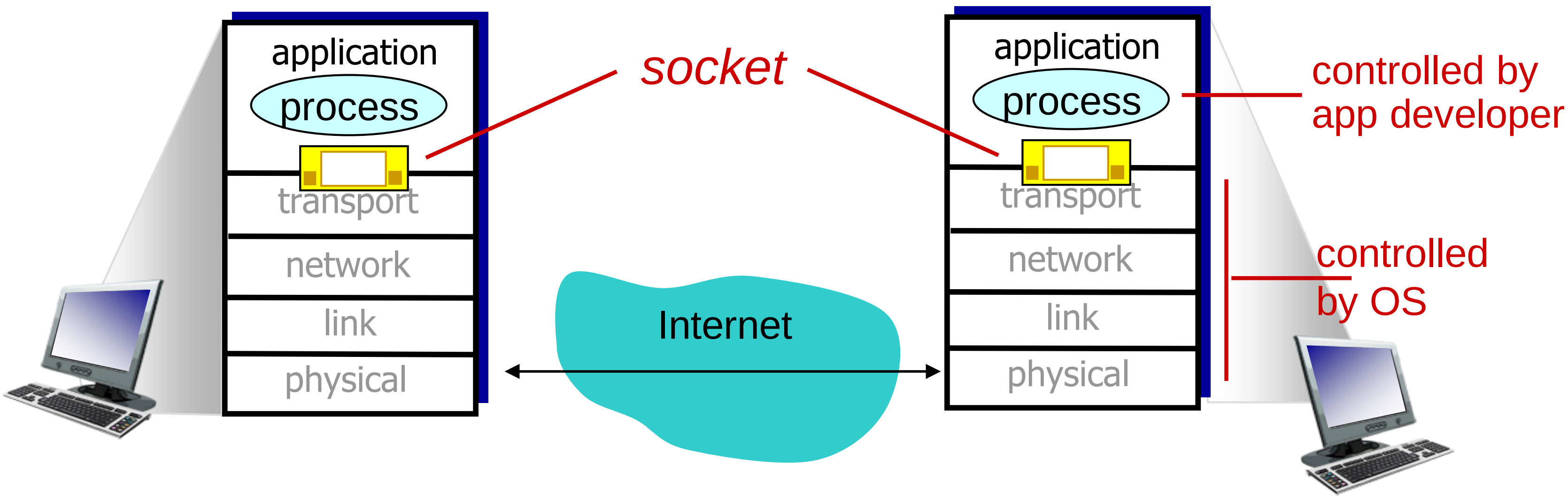
Chapter 2: Roadmap

- Principles of network applications
- Web and HTTP
- FTP
- Electronic mail
  - SMTP, POP3, IMAP
- DNS
- P2P Applications
- Socket programming with UDP and TCP

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Socket programming

- Goal: Learn how to build client/server applications that communicate using sockets
- Socket: Door between application process and end-end-transport protocol



- Two socket types for two transport services:
  - UDP: unreliable datagram
  - TCP: reliable, byte stream-oriented

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Socket programming with UDP

- UDP: no “connection” between client & server
  - No handshaking before sending data
  - Sender explicitly attaches IP destination address and port # to each packet
  - Receiver extracts sender IP address and port# from received packet
- UDP: transmitted data may be lost or received out-of-order
- Application viewpoint:
  - UDP provides unreliable transfer of groups of bytes (“datagrams”) between client and server

Socket programming with TCP

- Client must contact server
  - Server process must first be running
  - Server must have created socket (door) that welcomes client's contact
- Client contacts server by:
  - Creating TCP socket, specifying IP address, port number of server process
  - When client creates socket: client TCP establishes connection to server TCP

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Socket programming with TCP

- When contacted by client, server TCP creates new socket for server process to communicate with that particular client
  - Allows server to talk with multiple clients
  - Source port numbers used to distinguish clients (more in chap 3)
- Application viewpoint
  - TCP provides reliable, in-order byte-stream transfer (“pipe”) between client and server

Socket-Programmierung – ein Beispiel in Java

Ziel: lernen, wie man eine Client/Server-Anwendung programmiert, die über Sockets kommuniziert

Socket-API

- Eingeführt in BSD4.1 UNIX, 1981
- Sockets werden von Anwendungen erzeugt, verwendet und geschlossen
- Client/Server-Paradigma
- Zwei Transportdienste werden über die Socket-API angesprochen:
  - Unzuverlässige Paketübertragung
  - Zuverlässige Übertragung von Datenströmen

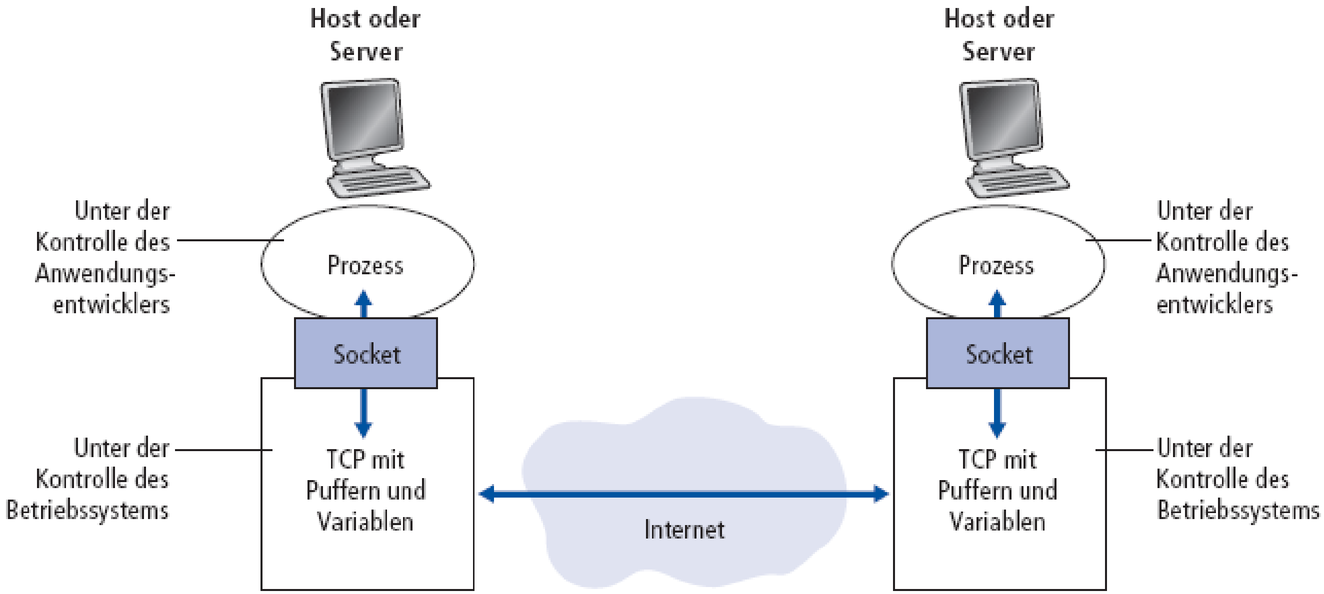
Socket

Eine Schnittstelle auf einem Host, kontrolliert durch das Betriebssystem, über das ein Anwendungsprozess sowohl Daten an einen anderen Prozess senden als auch von einem anderen Prozess empfangen kann.
Ein Socket ist eine Art “Tür” zum Computernetzwerk.

Socket-Programmierung mit TCP

Socket: eine „Tür“ zwischen dem Anwendungsprozess und dem Transportprotokoll (UDP oder TCP)

Wichtigster TCP-Dienst (aus Sicht der Anwendung): Zuverlässiger Transport eines Byte-Stroms vom sendenden zum empfangenden Prozess



Socket-Programmierung mit TCP

Client kontaktiert Server

- Server-Prozess muss laufen
- Server muss einen Socket (eine Tür) angelegt haben, der Client-Anfragen entgegennimmt

Vorgehen im Client:

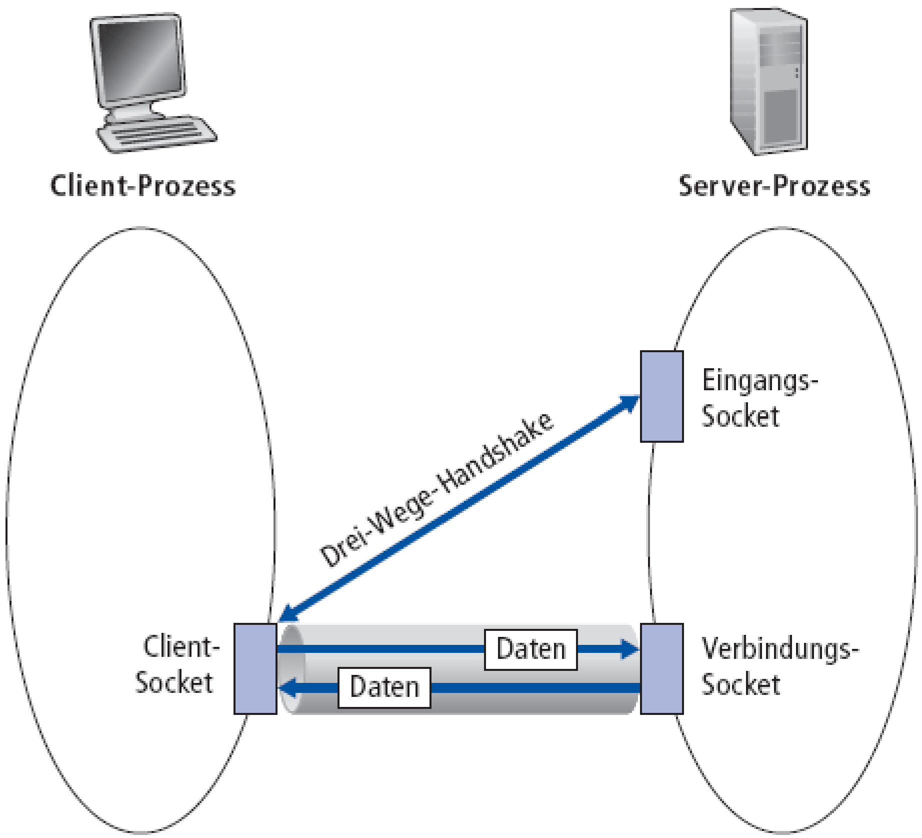
- Anlegen eines Client-TCP-Sockets
- Angeben von IP-Adresse und Portnummer des Server-prozesses
- Durch das Anlegen eines Client-TCP-Sockets wird eine TCP-Verbindung zum Server-prozess hergestellt

- Wenn der Serverprozess von einem Client kontaktiert wird, dann erzeugt er einen neuen Socket, um mit diesem Client zu kommunizieren
  - So kann der Server mit mehreren Clients kommunizieren
  - Portnummern der Clients werden verwendet, um die Verbindungen zu unterscheiden

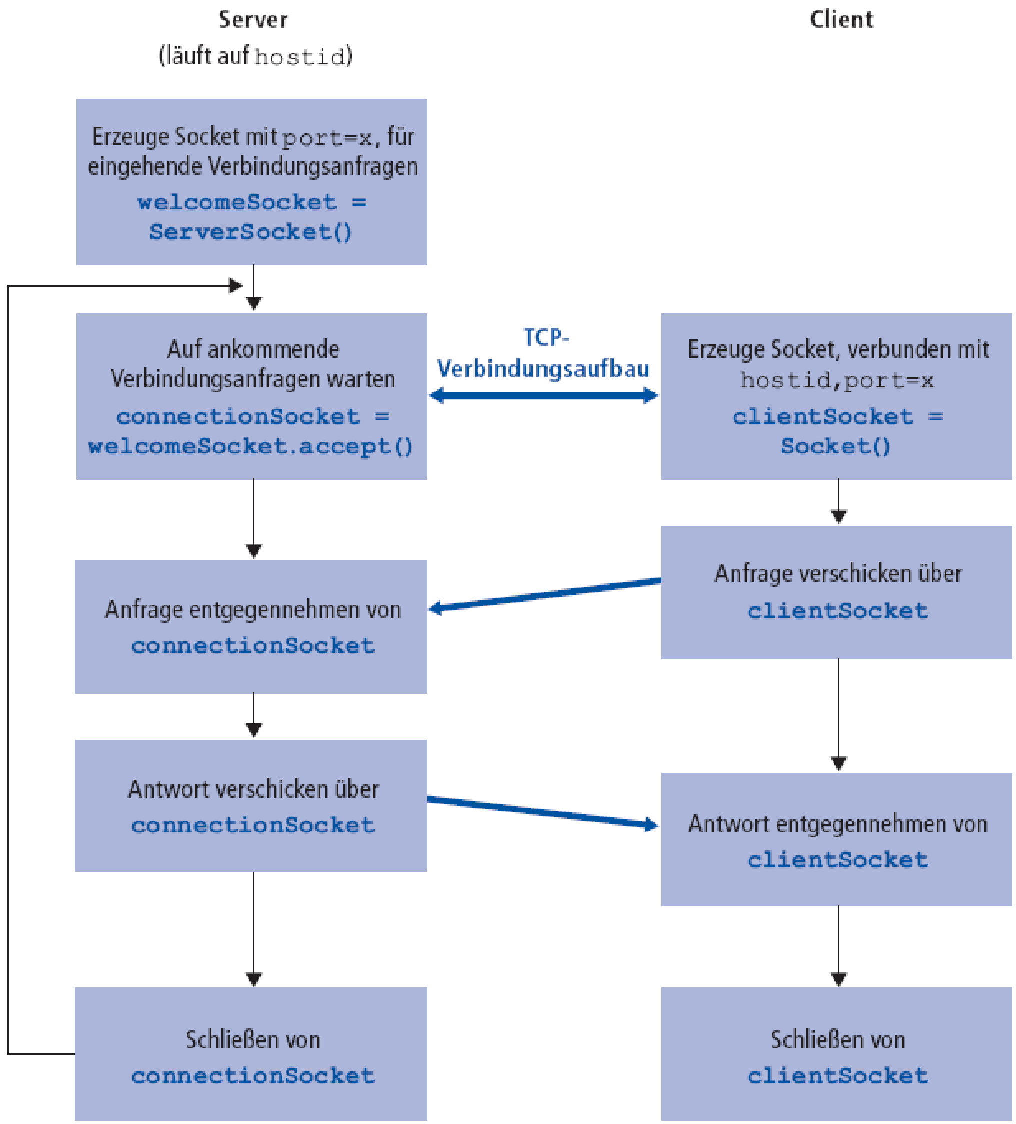
Anwendungsperspektive

TCP stellt einen zuverlässigen, reihenfolgeerhaltenden Transfer von Bytes zwischen Client und Server zur Verfügung

Zusammenspiel der Sockets

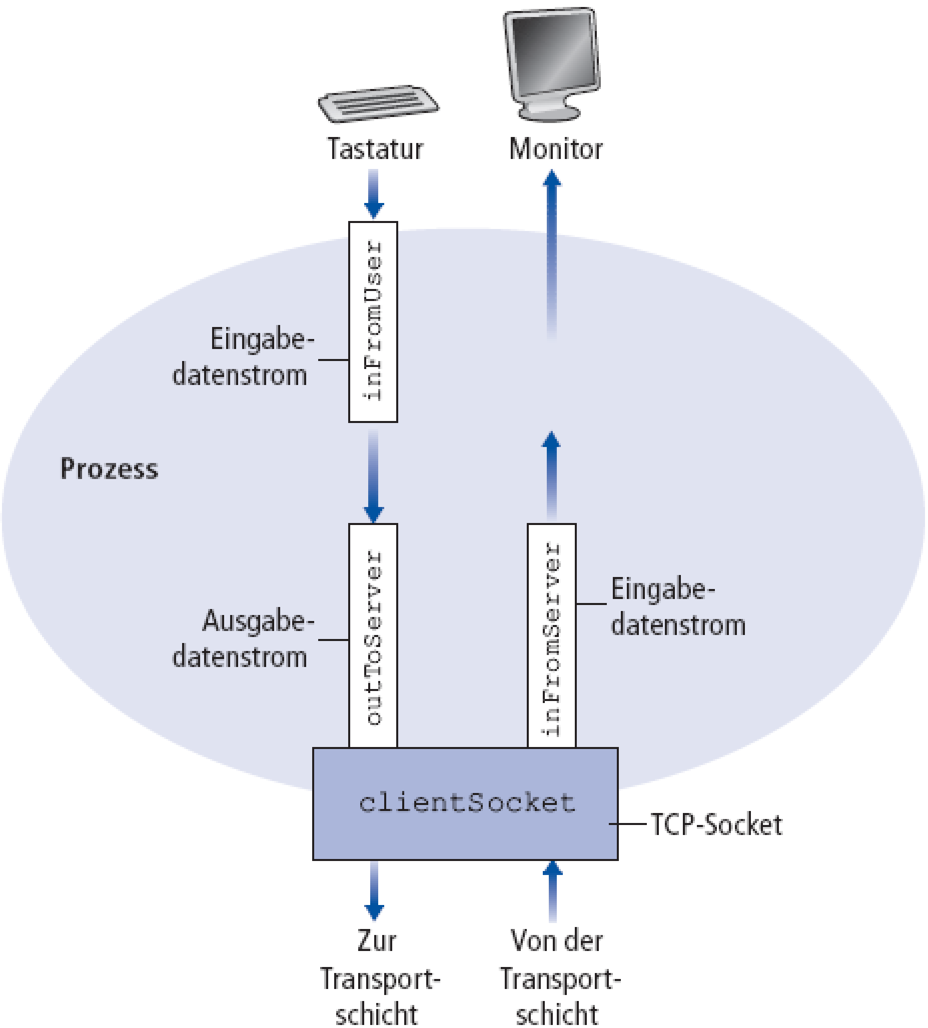


Client/Server-Socket-Programmierung: TCP



(Daten-)Ströme

- Ein Strom ist eine Folge von Bytes, die in einen Prozess hinein- oder aus ihm hinausfließen
- Ein Eingabestrom ist mit einer Quelle verbunden, z.B. Tastatur oder Socket
- Ein Ausgabestrom ist mit einer Senke verbunden, z.B. dem Monitor oder einem Socket



Socket-Programmierung mit TCP

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Beispiel für eine Client/Server-Anwendung:

1) Client liest Zeilen von der Standardeingabe (**inFromUser** Strom) und sendet diese über einen Socket (**outToServer** Strom) zum Server

2) Server liest die Zeile vom Socket

3) Server konvertiert die Zeile in Großbuchstaben und sendet sie zum Client zurück

4) Client liest die konvertierte Zeile vom Socket (**inFromServer** Strom) und gibt sie aus

Beispiel: Java-Client (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient {

    public static void main(String args[]) throws Exception  {
        String sentence;
        String modifiedSentence;

         Scanner inFromUser= new Scanner(System.in);
         sentence = inFromUser.nextLine();


        Socket socket= new Socket("hostname", 6789);
```

Eingabstrom anlegen, Satz vom Nutzer holen

Client-Socket anlegen, mit dem Server verbinden

Beispiel: Java-Client (TCP)

Ausgabestrom anlegen, mit Socket verbinden

```
PrintStream outToServer
        = new PrintStream(socket.getOutputStream());
```

Eingabestrom anlegen, mit Socket verbinden

```
Scanner inFromServer
        = new Scanner(socket.getInputStream());
```

Zeile an Server schicken

```
outToServer.println(sentence + '\n');
```

Zeile vom Server lesen

```
modifiedSentence = inFromServer.nextLine();

System.out.println("FROM SERVER: " + modifiedSentence);

socket.close();

    }
}
```

Beispiel: Java-Server (TCP)

```
import java.io.*;
import java.net.*;

class TCPServer {

  public static void main(String args[]) throws Exception {
     String clientSentence;
     String capitalizedSentence;

     ServerSocket serverSocket = new ServerSocket(6789);


     while(true) {

        Socket socket = serverSocket.accept();

        Scanner inFromClient
          = new Scanner(socket.getInputStream());
```

Socket für ein-gehende Anfragen anlegen (Port 6789)

An diesem Socket auf Anfragen von Clients warten

Neuer Socket!!!

Eingabestrom anlegen, mit Socket verbinden

Beispiel: Java-Server (TCP), Forts.

Ausgabestrom anlegen, mit Socket verbinden

```
PrintStream outToClient
    = new PrintStream(socket.getOutputStream());
```

Zeile vom Socket lesen

```
clientSentence = inFromClient.nextLine();

capitalizedSentence = clientSentence.toUpperCase() + '\n';
```

Zeile an Client schicken

```
outToClient.println(capitalizedSentence);
}
    }
  }
```

Ende der while-Schleife, auf den nächsten Client warten

Socket-Programmierung mit UDP

UDP: keine „Verbindung“ zwischen Client und Server

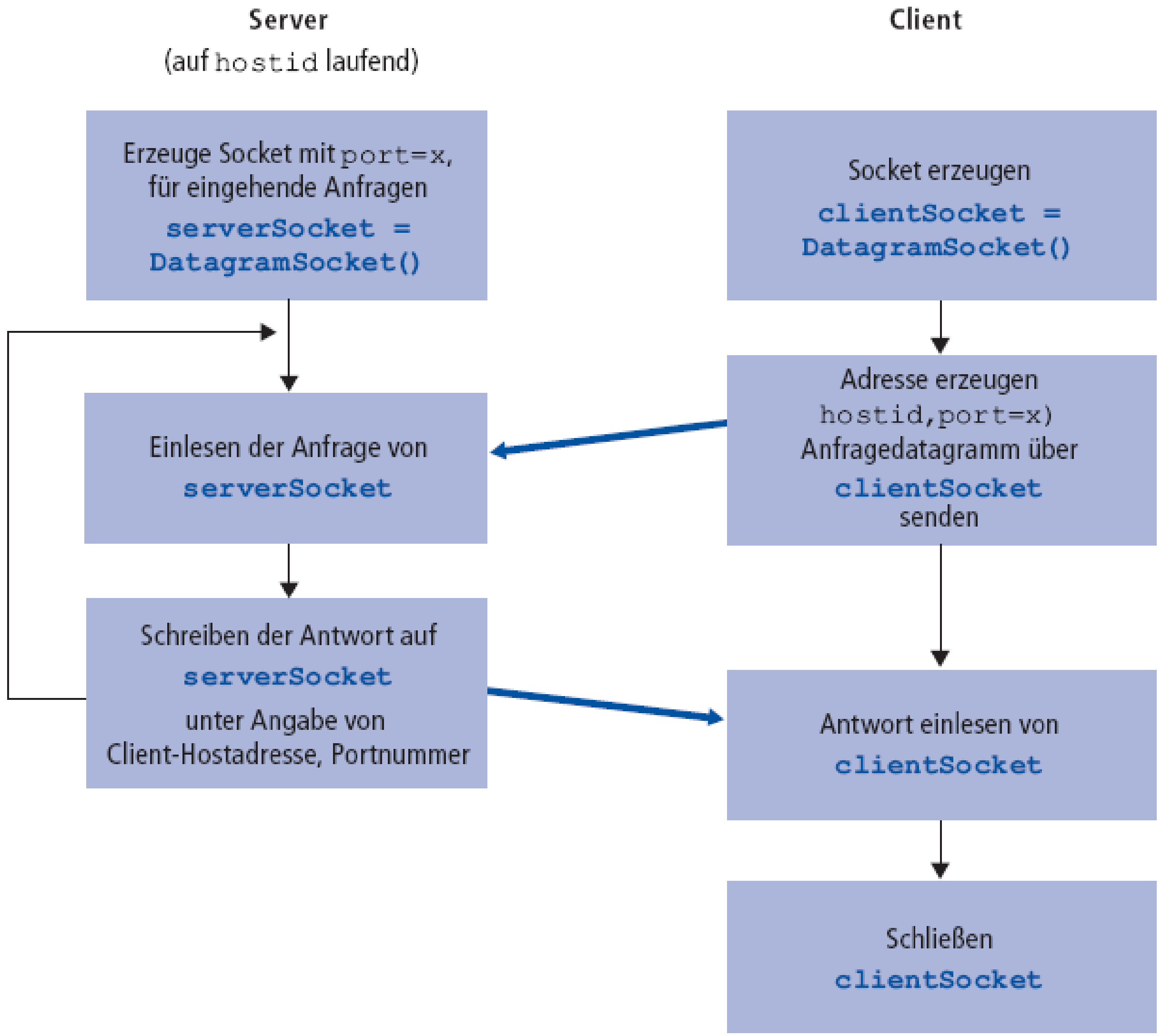
- Kein Verbindungsaufbau
- Sender hängt explizit die IP-Adresse und Port-nummer des empfangenden Prozesses an jedes Paket an
- Server liest die IP-Adresse und die Port-nummer des sendenden Prozesses explizit aus dem empfangenen Paket aus

UDP: Pakete können in falscher Reihenfolge empfangen werden oder ganz verloren gehen

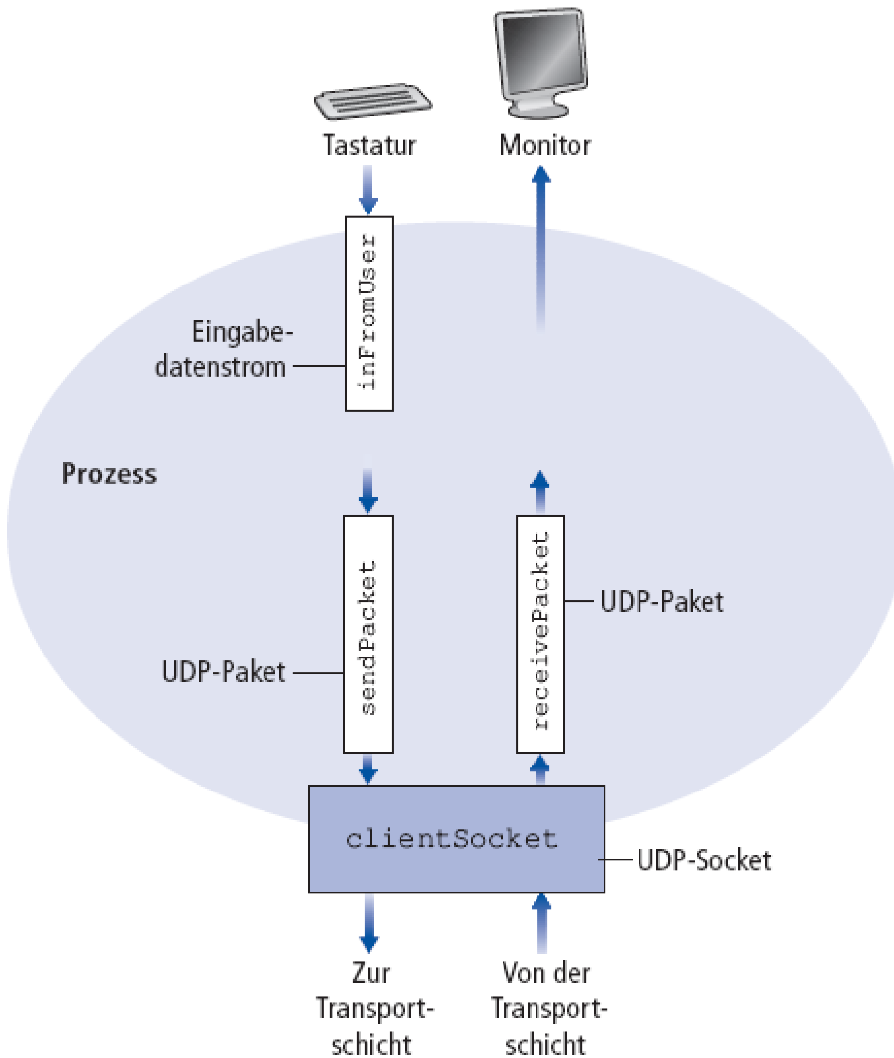
Anwendungsperspektive

UDP stellt einen unzuverlässigen Transport einer Gruppe von Bytes (“Paket”) zwischen Client und Server zur Verfügung

Client/Server-Socket-Programmierung: UDP



Beispiel: Java-Client (UDP)



Beispiel: Java-Client (UDP)

Anlegen des ClientSocket

Übersetzen von hostname in eine IP-Adresse über DNS

Eingabstrom anlegen, Satz vom Nutzer holen

```
import java.io.*;
import java.net.*;

class UDPClient {
    public static void main(String args[]) throws Exception
    {
      DatagramSocket datagramSocket = new DatagramSocket();

      InetAddress IPAddress = InetAddress.getByName("hostname");

      byte[] sendData = new byte[1024];
      byte[] receiveData = new byte[1024];

      Scanner inFromUser= new Scanner(System.in);
      String sentence = inFromUser.readLine();


      sendData = sentence.getBytes();
```

Beispiel: Java-Client (UDP)

Paket anlegen: Daten, Länge, IP, Port

```
DatagramPacket sendPacket =
  new DatagramPacket(sendData, sendData.length, IPAddress, 9876);
```

Paket an Server schicken

```
datagramSocket.send(sendPacket);

DatagramPacket receivePacket =
  new DatagramPacket(receiveData, receiveData.length);
```

Paket vom Server empfangen

```
datagramSocket.receive(receivePacket);

String modifiedSentence =
   new String(receivePacket.getData());

System.out.println("FROM SERVER:" + modifiedSentence);
datagramSocket.close();
}
}
```

Beispiel: Java-Server (UDP)

```
import java.io.*;
import java.net.*;

class UDPServer {
  public static void main(String args[]) throws Exception
    {

      DatagramSocket datagramSocket= new DatagramSocket(9876);

      byte[] receiveData = new byte[1024];
      byte[] sendData  = new byte[1024];

      while(true)
        {

          DatagramPacket receivePacket =
             new DatagramPacket(receiveData, receiveData.length);
          datagramSocket.receive(receivePacket);
```

Datagramm-Socket auf Port 9876 anlegen

Platz für das zu empfangende Paket reservieren

Paket empfangen

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Beispiel: Java-Server (UDP), Forts.

IP und Port des Clients bestimmen

Zu sendendes Paket anlegen

Paket über Socket senden

```
String sentence = new String(receivePacket.getData());

InetAddress IPAddress = receivePacket.getAddress();

int port = receivePacket.getPort();

        String capitalizedSentence = sentence.toUpperCase();

sendData = capitalizedSentence.getBytes();

DatagramPacket sendPacket =
  new DatagramPacket(sendData, sendData.length, IPAddress,
              port);

datagramSocket.send(sendPacket);
}
}
}
```

Ende der while-Schleife, auf nächstes Paket warten

Chapter 2: summary

Our study of network apps now complete!

- Application architectures
  - Client-server
  - P2P
- Application service requirements:
  - Reliability, bandwidth, delay
- Internet transport service model
  - Connection-oriented, reliable: TCP
  - Unreliable, datagrams: UDP
- Specific protocols:
  - HTTP
  - FTP
  - SMTP, POP, IMAP
  - DNS
  - P2P: BitTorrent, DHT
- Socket programming: TCP, UDP sockets

Chapter 2: summary

Most importantly: learned about protocols!

- Typical request/reply message exchange:
  - Client requests info or service
  - Server responds with data, status code
- Message formats:
  - Headers: fields giving info about data
  - Data: info being communicated

- Important themes:
  - Control vs. Data msgs
  - In-band, out-of-band
  - Centralized vs. Decentralized
  - Stateless vs. Stateful
  - Reliable vs. Unreliable msg transfer
  - “Complexity at network edge”