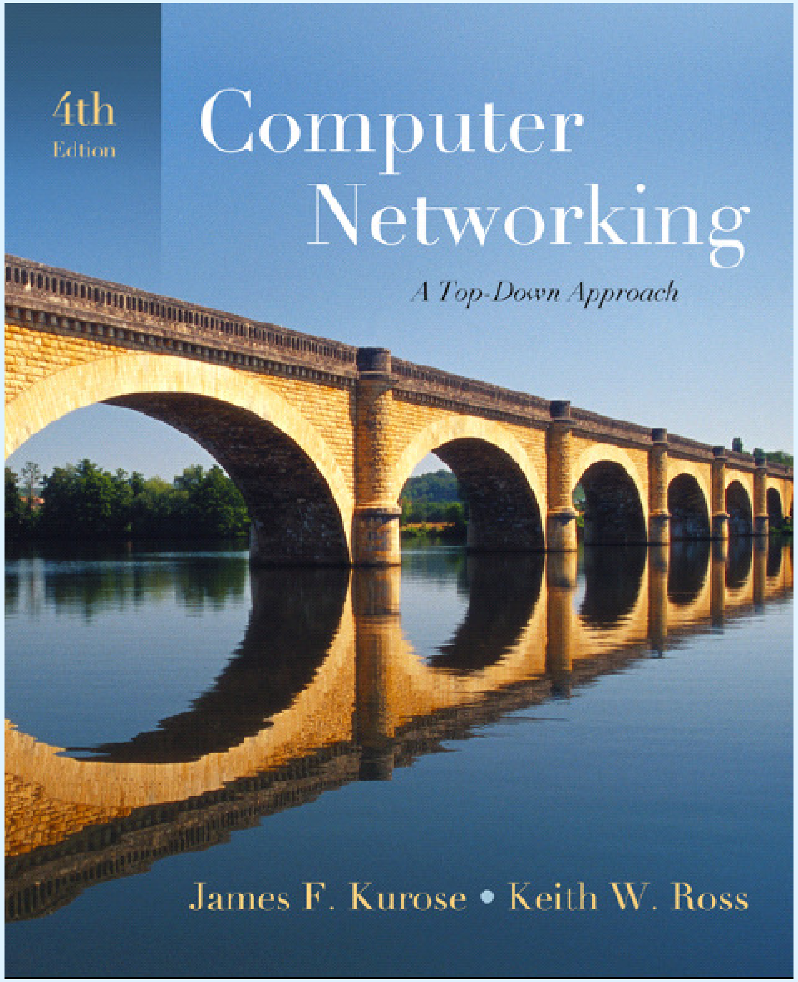
Application layer

Bildspelet omfattar till stor del bilder som hör till följande bok:



A note on the use of these ppt slides:

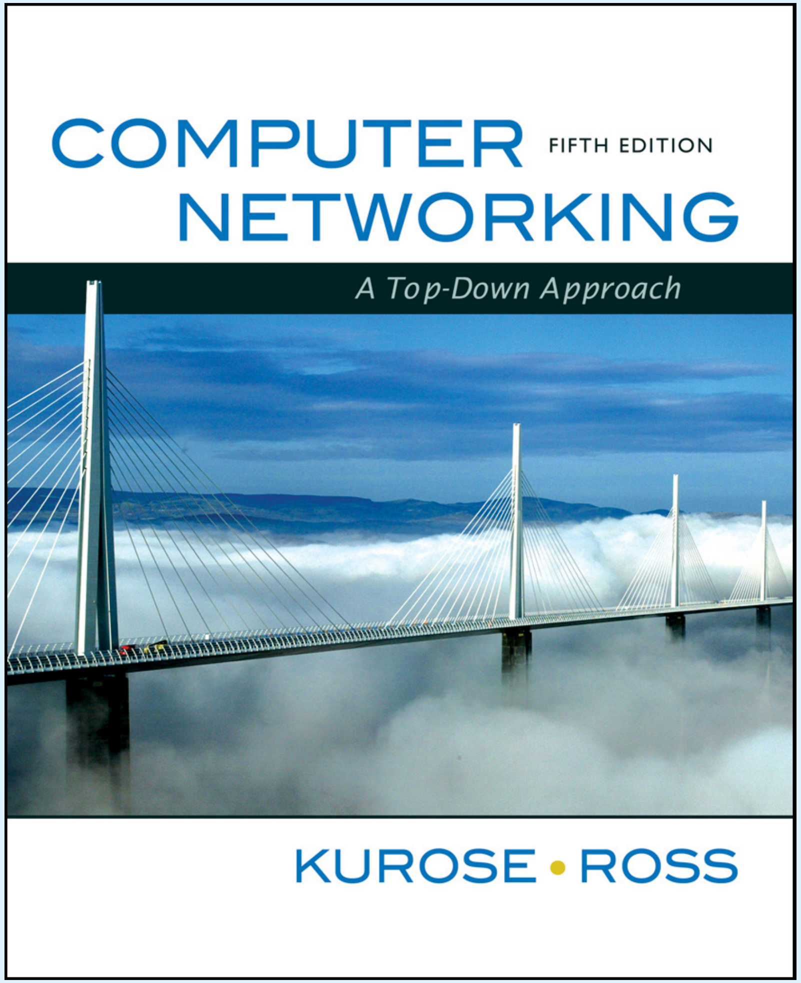
We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we'd like people to use our book!)
- If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

Computer Networking: A Top Down Approach , 4th edition.
Jim Kurose, Keith Ross, Addison-Wesley, July 2007.

Dessutom 15 bilder från följande bok:



A note on the use of these ppt slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we'd like people to use our book!)
- If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

Computer Networking: A Top Down Approach , 5th edition.
Jim Kurose, Keith Ross, Addison-Wesley, April 2009.

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

Public-domain protocols:

- defined in RFCs
- allows for interoperability
- e.g., HTTP, SMTP

Proprietary protocols:

- e.g., Skype

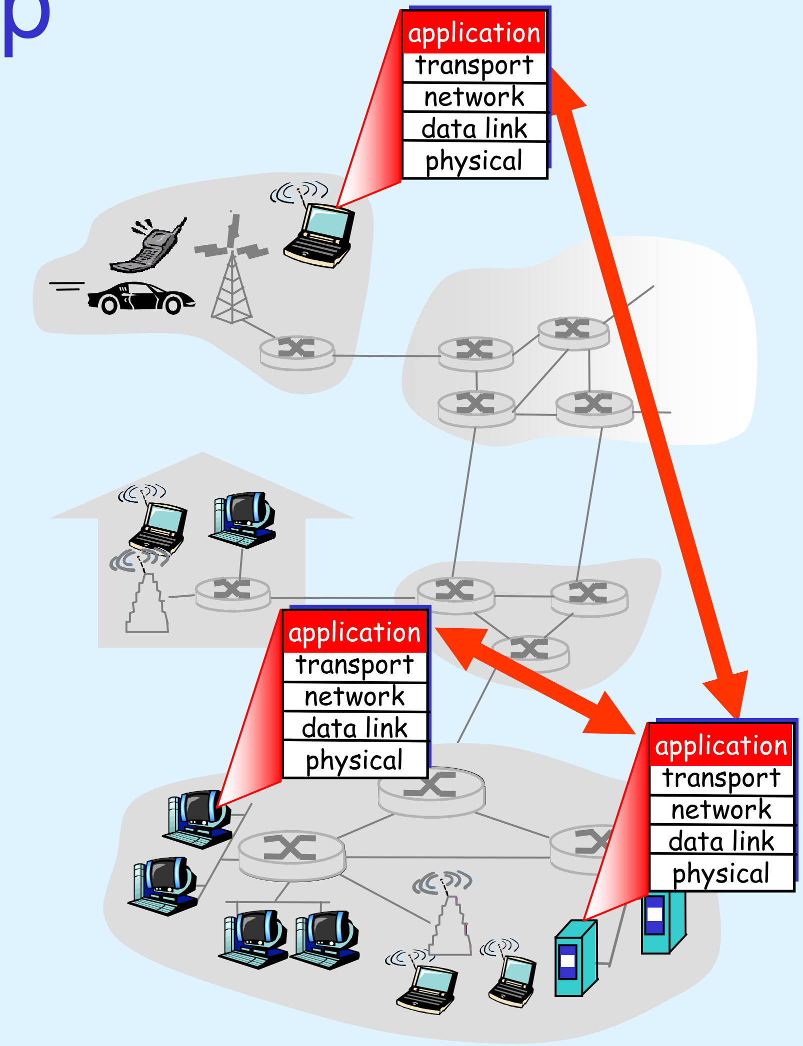
Creating a network app

write programs that

- run on (different) *end systems*

little software written for devices in network core

- network core devices do not run user applications



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**

Client process: process that initiates communication

Server process: process that waits to be contacted

- Note: applications with P2P architectures have client processes & server processes

Adressering av applikationsprogram

- Värden adresseras med dess IP-adress
- Applikationsprogrammet i värden adresseras genom sitt applikationsprotokoll
- Applikationsprotokollet adressera av sin SAP, dvs. porten
- Porten finns i gränssnittet mellan transportskiktet och applikationsskiktet

Transport service requirements of common apps

Application	Data loss	Bandwidth	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
instant messaging	no loss	elastic	yes and no

Internet transport protocols services

TCP service:

- *connection-oriented:* setup required between client and server processes
- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not provide:* timing, minimum bandwidth guarantees

UDP service:

- *connectionless*
- *unreliable data transfer* between sending and receiving process
- *does not provide*: connection setup, reliability, flow control, congestion control, timing, or bandwidth guarantee

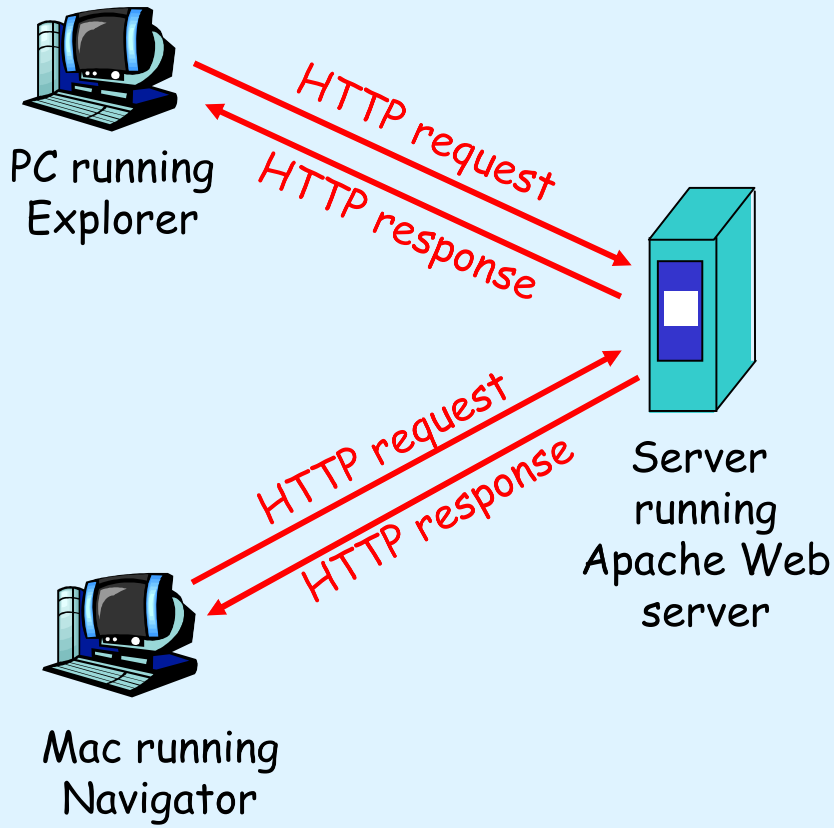
Web and HTTP

- Web page consists of base HTML-file which includes several referenced objects
- Each object is addressable by a URL

HTTP overview

HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2068



HTTP overview (continued)

Uses TCP:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP 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

HTTP connections

Nonpersistent HTTP

- At most one object is sent over a TCP connection.
- HTTP/1.0 uses nonpersistent HTTP

Persistent HTTP

- Multiple objects can be sent over single TCP connection between client and server.
- HTTP/1.1 uses persistent connections in default mode

Persistent HTTP

<u>Nonpersistent HTTP issues:</u>

- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

<u>Persistent HTTP</u>

- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection

<u>Persistent ***without*** pipelining:</u>

- client issues new request only when previous response has been received
- one RTT for each referenced object

<u>Persistent ***with*** pipelining:</u>

- default in HTTP/1.1
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

HTTP request message

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

request line (GET, POST, HEAD commands)

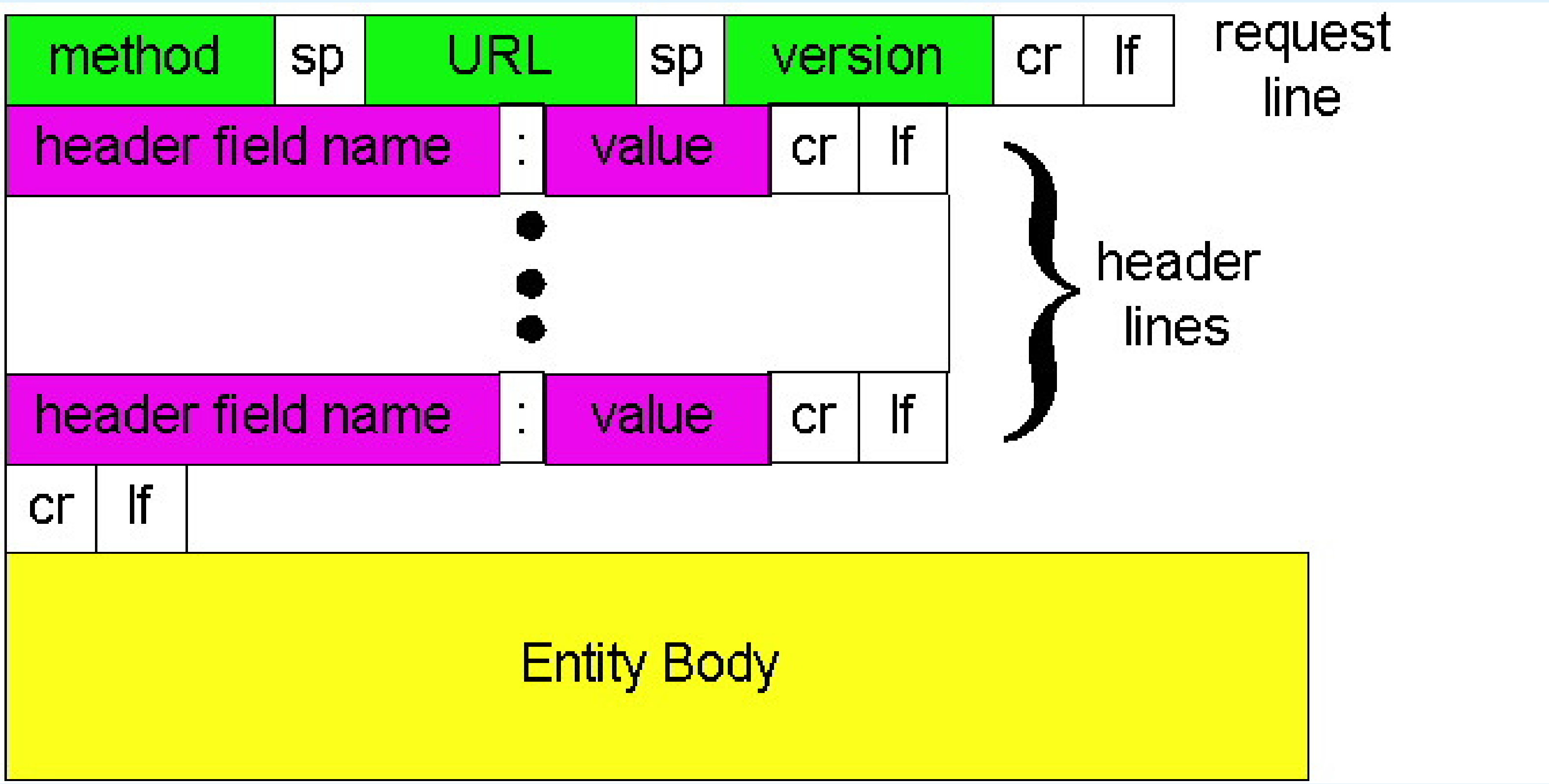
```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

header lines

Blank line (extra carriage return, line feed)

Carriage return, line feed indicates end of message

HTTP request message: general format



Uploading form input

Post method:

- Web page often includes form input
- Input is uploaded to server in entity body

GET method:

- Uses the URL
- Input is uploaded in URL field of request line:

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

HTTP response message

status line
(protocol
status code
status phrase)

header
lines

data, e.g.,
requested
HTML file

```
HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …...
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```

HTTP response status codes

A few sample codes:

`200 OK`

- request succeeded, requested object later in this message

`301 Moved Permanently`

- requested object moved, new location specified later in this message (Location:)

`400 Bad Request`

- request message not understood by server

`404 Not Found`

- requested document not found on this server

`505 HTTP Version Not Supported`

HTTP-Exempel (1)

1. På kommandprompten
   telnet basen.oru.se 80 [Enter]

2. Skriv "i blindo" i telnet
   HEAD /datorkom/tomten.html HTTP/1.1 [Enter]

3. Skriv "i blindo" i telnet
   Host: basen.oru.se [Enter] [Enter]

"I blindo" eftersom
set localecho
avbryter kommunikationen.

HTTP-Exempel (2)

Telnet basen.oru.se

```
HTTP/1.1 200 OK
Vary: accept-encoding
Server: Roxen/5.2.200-r2
Accept-Ranges: bytes
Last-Modified: Fri, 13 Feb 2015 12:55:48 GMT
Date: Wed, 23 Sep 2015 12:36:00 GMT
Connection: keep-alive
Content-Type: text/html
Content-Length: 476
ETag: "32bab6fdb6d2e270c527eff196aab255"
```

Cookies (continued)

What cookies can bring:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

User-server state: cookies

Many major Web sites use cookies

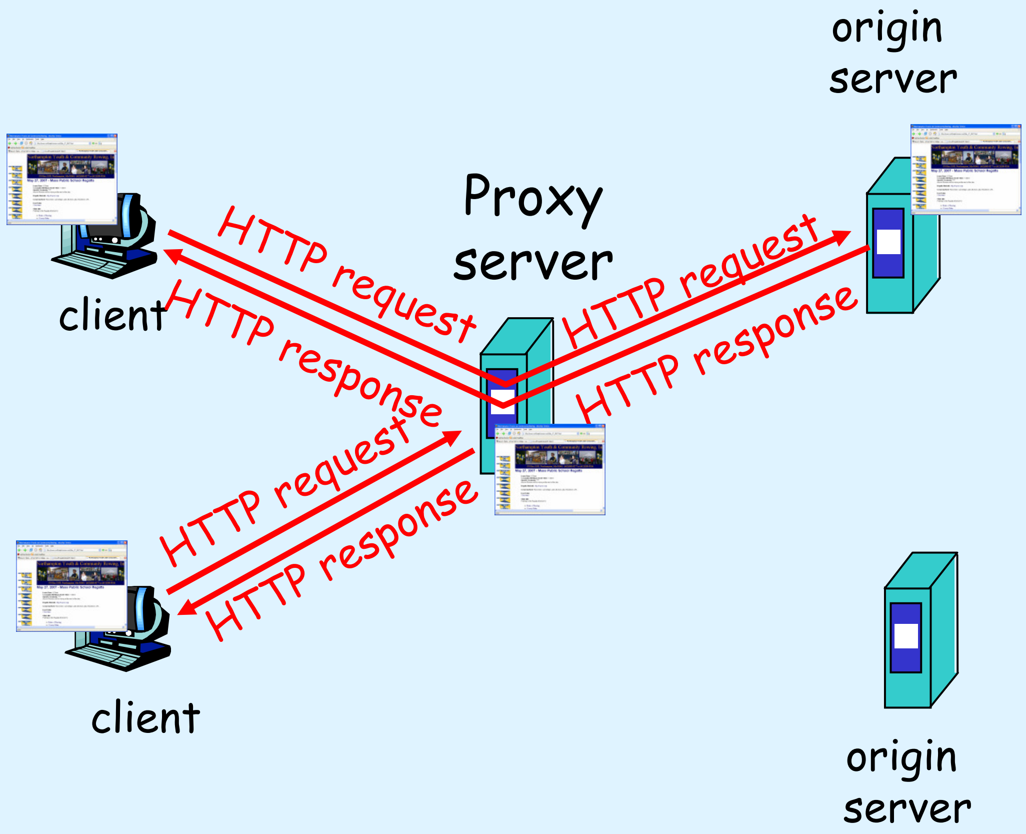
<u>Four components:</u>

1) cookie header line of HTTP *response* message
2) cookie header line in HTTP *request* message
3) cookie file kept on user's host, managed by user's browser
4) back-end database at Web site

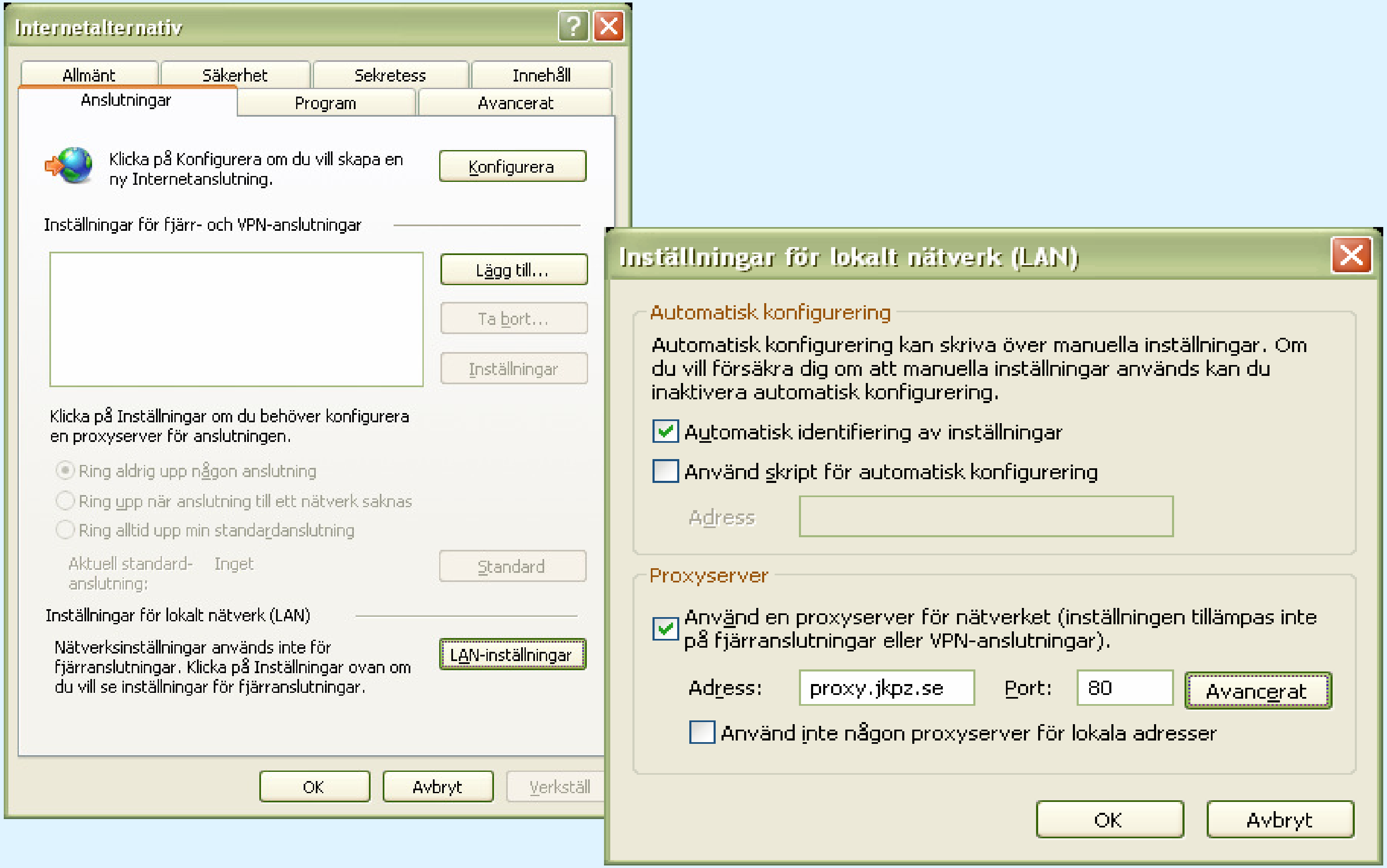
Web caches (proxy server)

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



Proxyinställning i IE



More about Web caching

- cache acts as both client and 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 (but so does P2P file sharing)

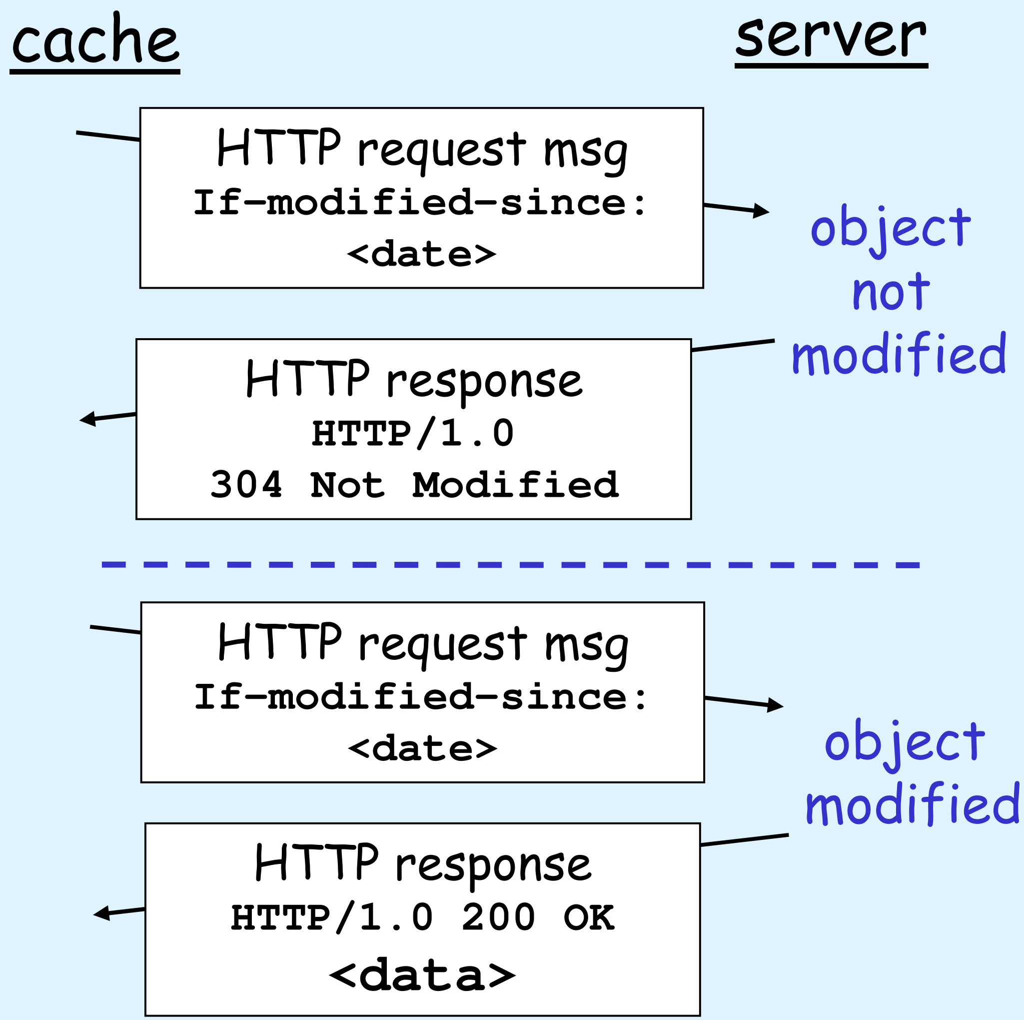
Conditional GET

- **Goal:** don't send object if cache has up-to-date cached version
- cache: specify date of cached copy in HTTP request

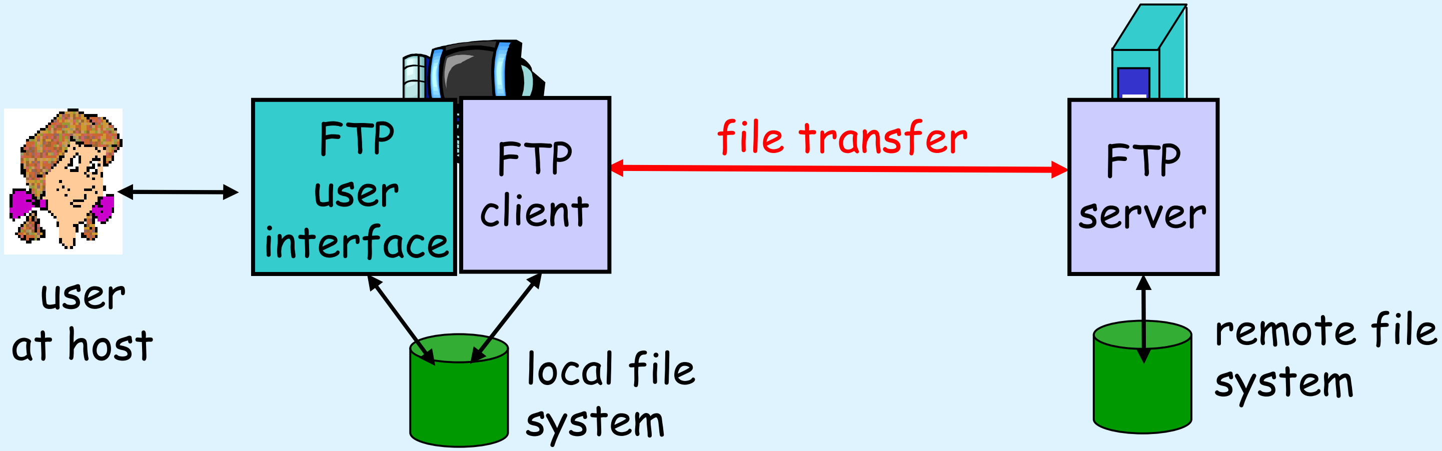
`If-modified-since: <date>`

- server: response contains no object if cached copy is up-to-date:

`HTTP/1.0 304 Not Modified`



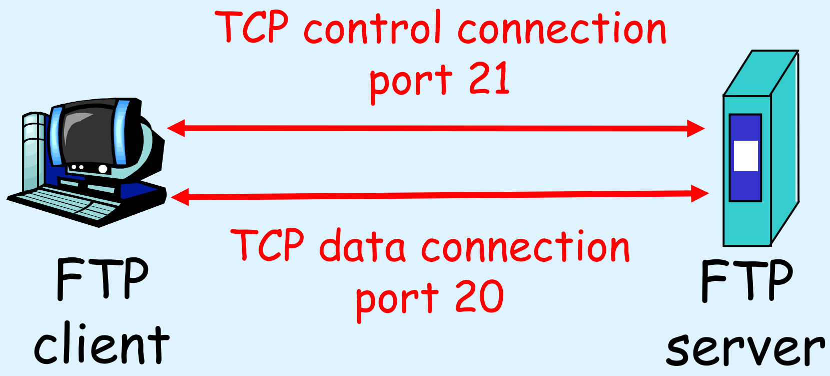
FTP: the file transfer protocol



- 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: separate control, data connections

- FTP client contacts FTP server at port 21, TCP is transport protocol
- client authorized over control connection
- client browses remote directory by sending commands over control connection.
- when server receives file transfer command, server opens *2nd* TCP 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

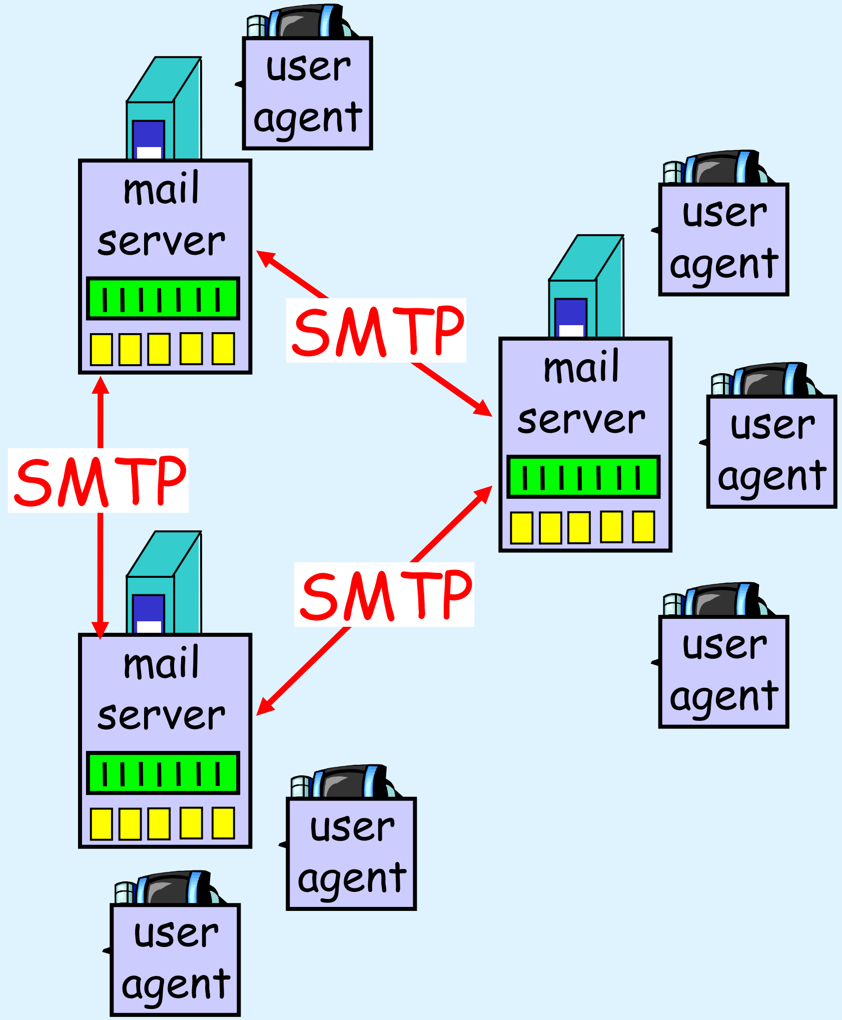
Sample return codes

- 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**

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
  - commands: ASCII text
  - response: status code and phrase
- messages must be in 7-bit ASCII

Message format: multimedia extensions

- MIME: multimedia mail extension, RFC 2045, 2056
- additional lines in msg header declare MIME content type

MIME version

method used to encode data

multimedia data type, subtype, parameter declaration

encoded data

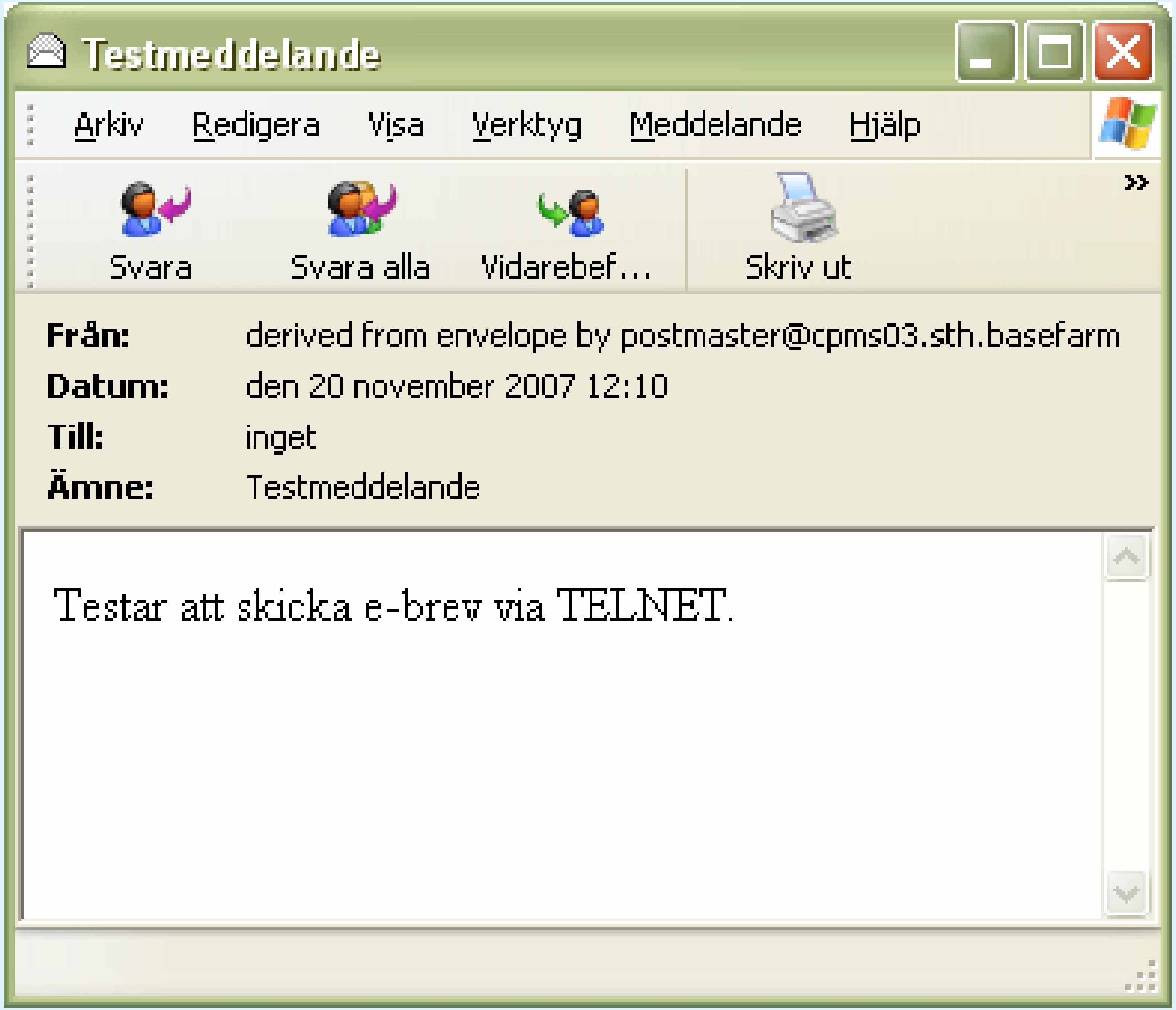
```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.........................
......base64 encoded data
```

Try SMTP interaction for yourself:

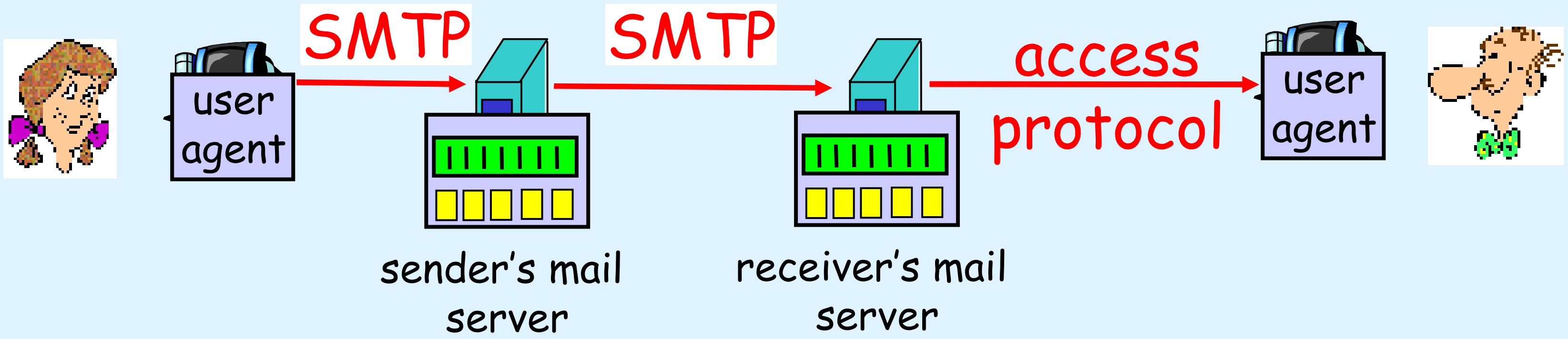
1. **`telnet servername 25`**
2. see 220 reply from server
3. enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

Kommandotolken

```
220 ch-smtp01.sth.basefarm.net ESMTP Exim 4.68 Tue, 20 Nov 2007 12:07:03 +0100
HELO comhem.se
250 ch-smtp01.sth.basefarm.net Hello c83-253-64-85.bredband.comhem.se [83.253.64
.85]
MAIL FROM: jack.pencz@comhem.se
250 OK
RCPT TO: jack.pencz@comhem.se
250 Accepted
DATA
354 Please start mail input.
Subject: Testmeddelande

Testar att skicka e-brev via TELNET.
.
250 Mail queued for delivery.
QUIT
221 Closing connection. Good bye.


Anslutningen till värddatorn bröts.

C:\Documents and Settings\Ägaren>
```


Testmeddelande
Arkiv Redigera Visa Verktyg Meddelande Hjälp
Svara
Svara alla
Vidarebef...
Skriv ut
Från: derived from envelope by postmaster@cpms03.sth.basefarm
Datum: den 20 november 2007 12:10
Till: inget
Ämne: Testmeddelande
Testar att skicka e-brev via TELNET.

Mail access protocols



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

DNS

DNS services

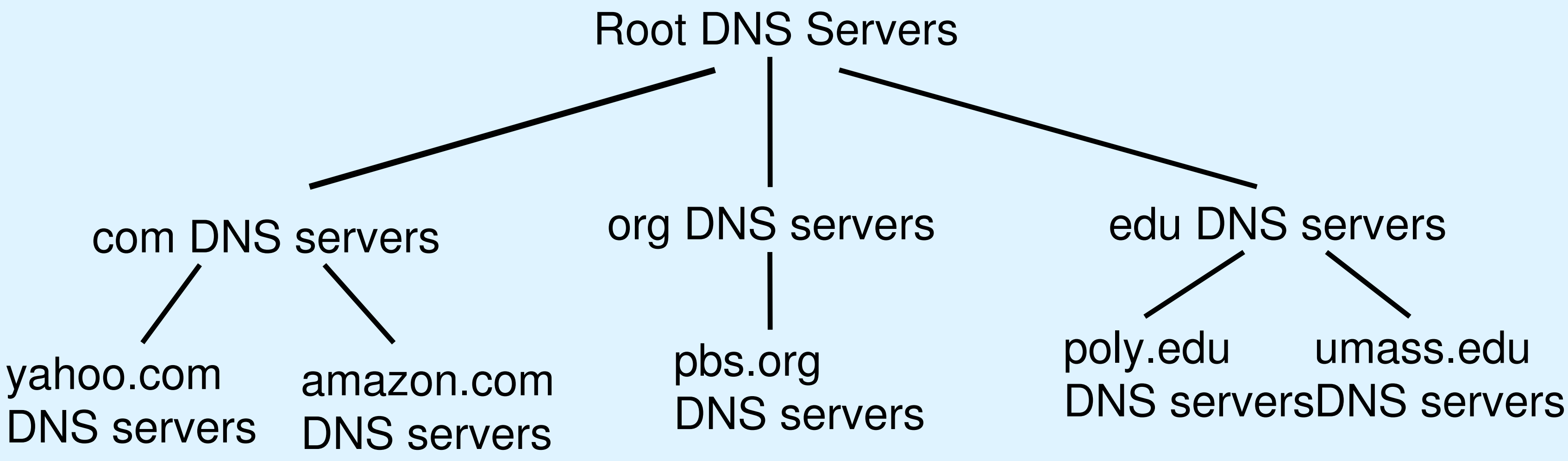
- hostname to IP address translation
- host aliasing
  - Canonical, alias names
- mail server aliasing
- load distribution
  - replicated Web servers: set of IP addresses for one canonical name

Why not centralize DNS?

- single point of failure
- traffic volume
- distant centralized database
- maintenance

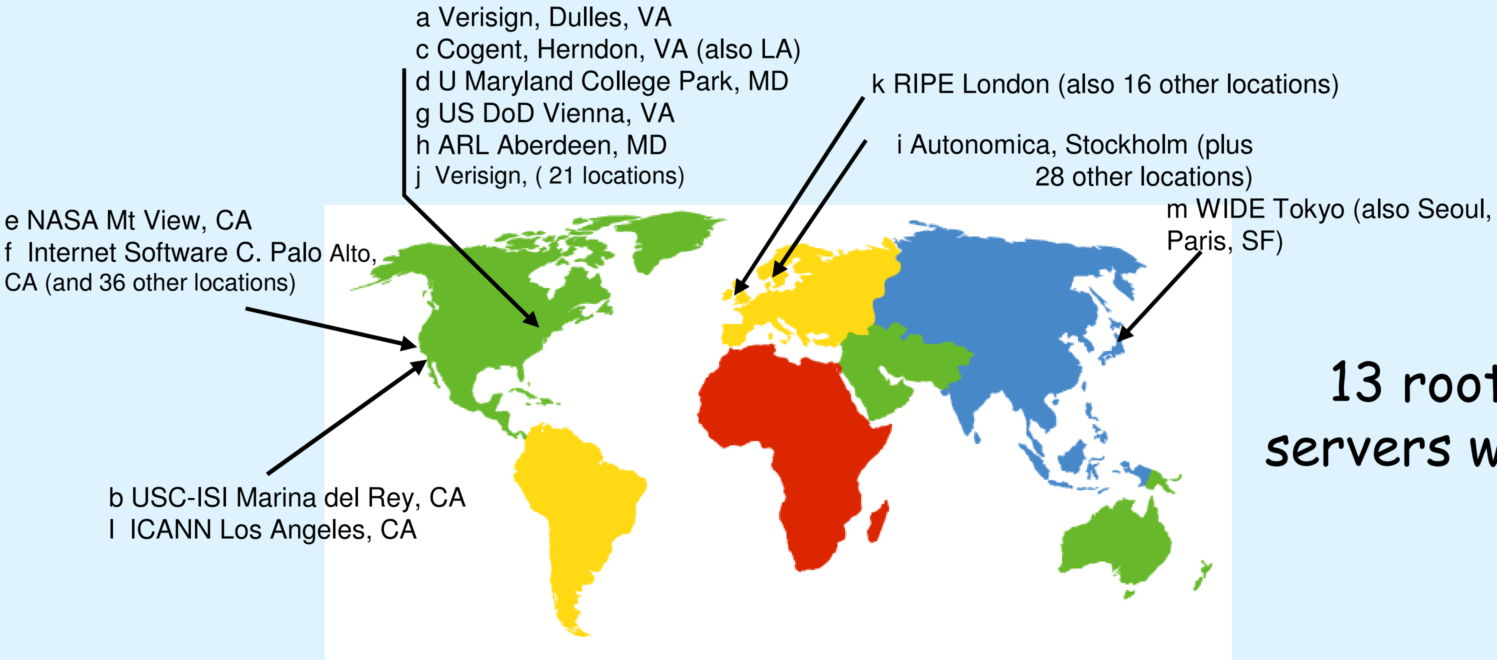
doesn't *scale!*

Distributed, Hierarchical Database



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 and Authoritative Servers

- **Top-level domain (TLD) servers:**
  - responsible for com, org, net, edu, etc, and all top-level country domains uk, fr, ca, jp.
- **Authoritative DNS servers:**
  - organization's DNS servers, providing authoritative hostname to IP mappings for organization's servers (e.g., Web, mail).
  - can be maintained by organization or service provider

Local Name Server

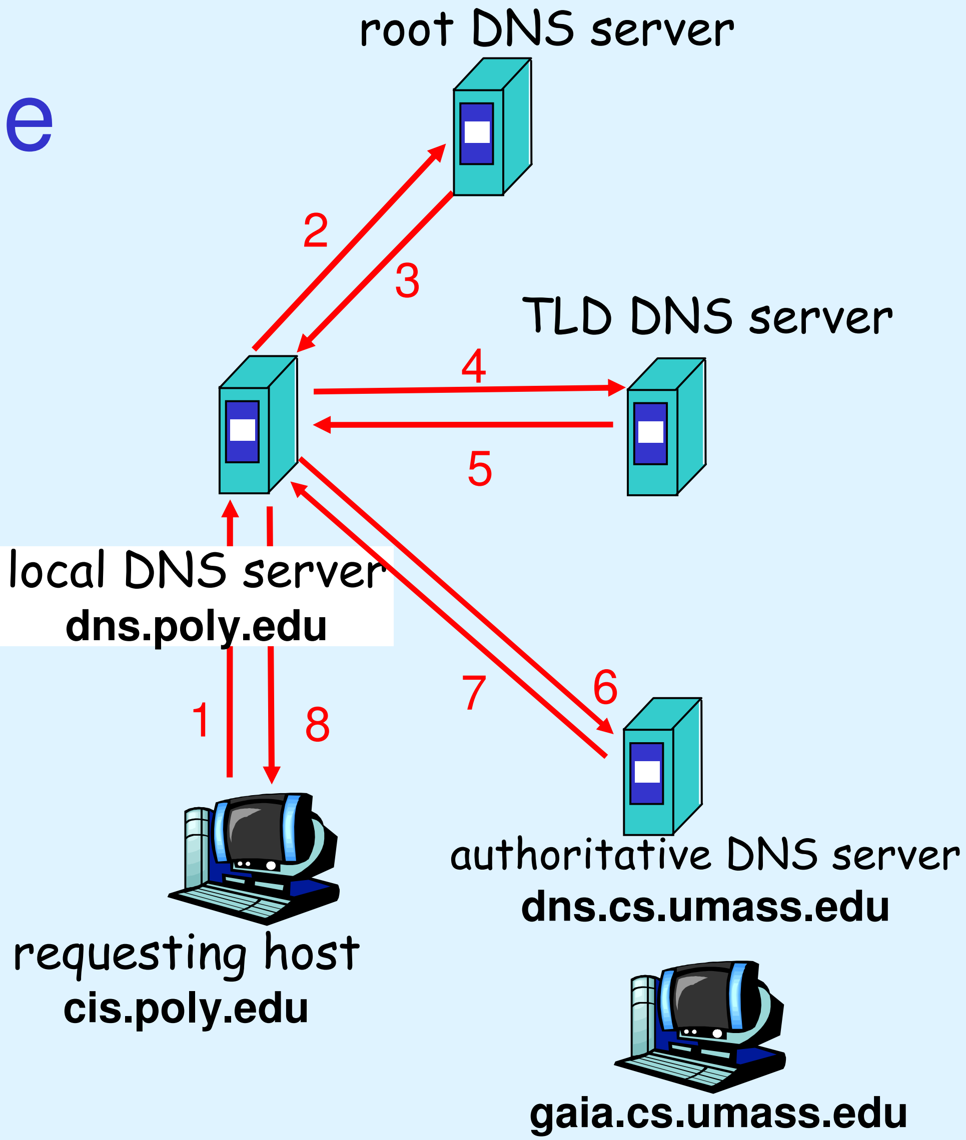
- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one.
  - also called "default name server"
- when host makes DNS query, query is sent to its local DNS server
  - acts as proxy, forwards query into hierarchy

DNS name resolution example

- Host at cis.poly.edu wants IP address for gaia.cs.umass.edu

<u>iterated query:</u>

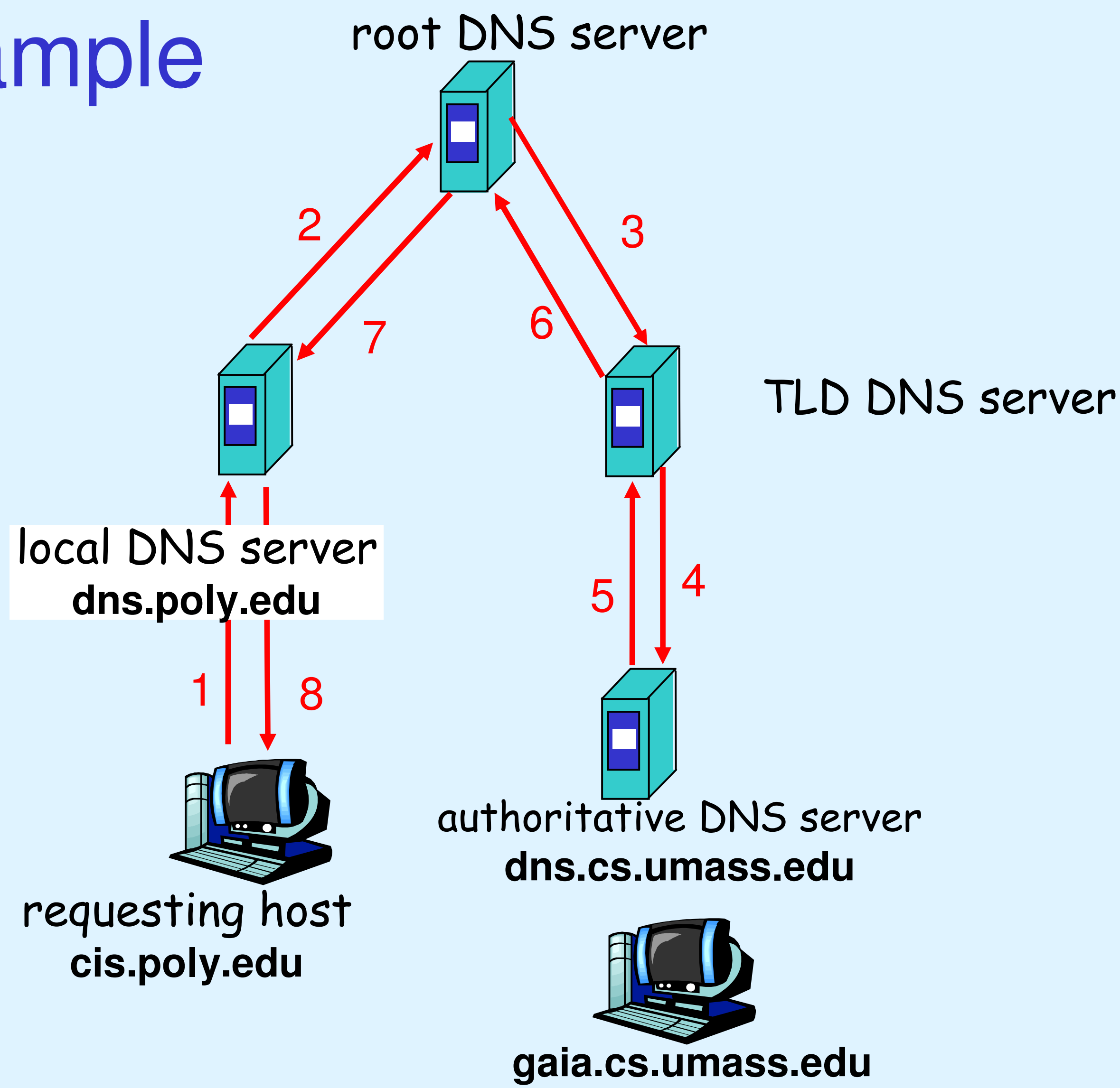
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



DNS name resolution example

<u>recursive query:</u>

- puts burden of name resolution on contacted name server
- heavy load?

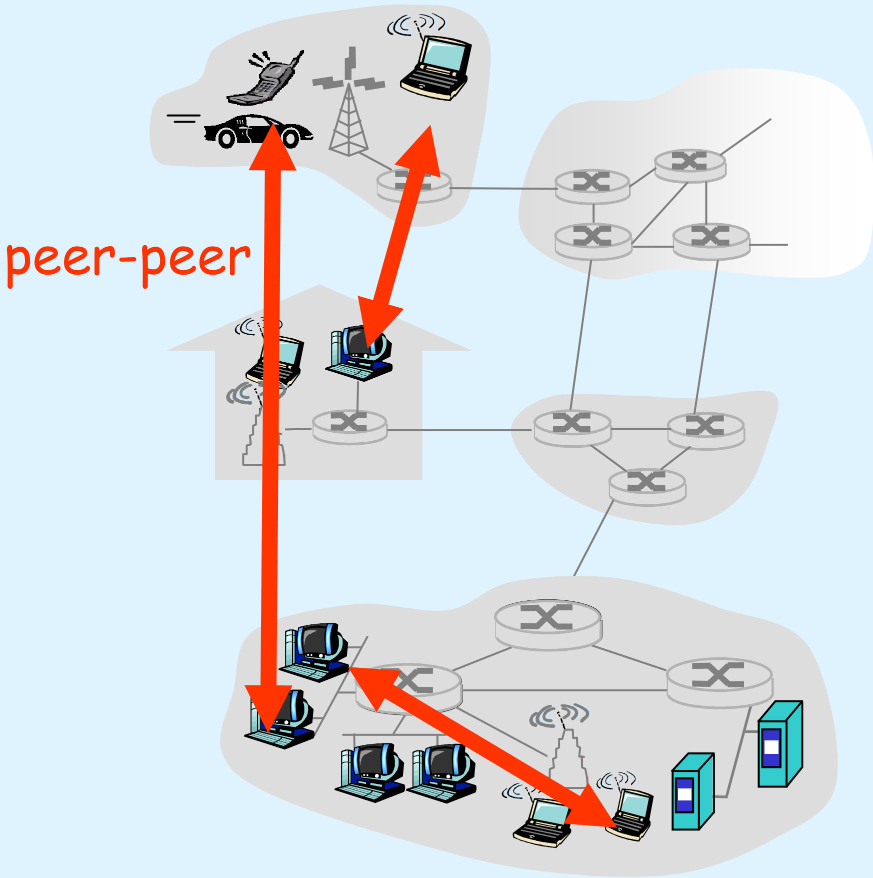


DNS: caching and updating records

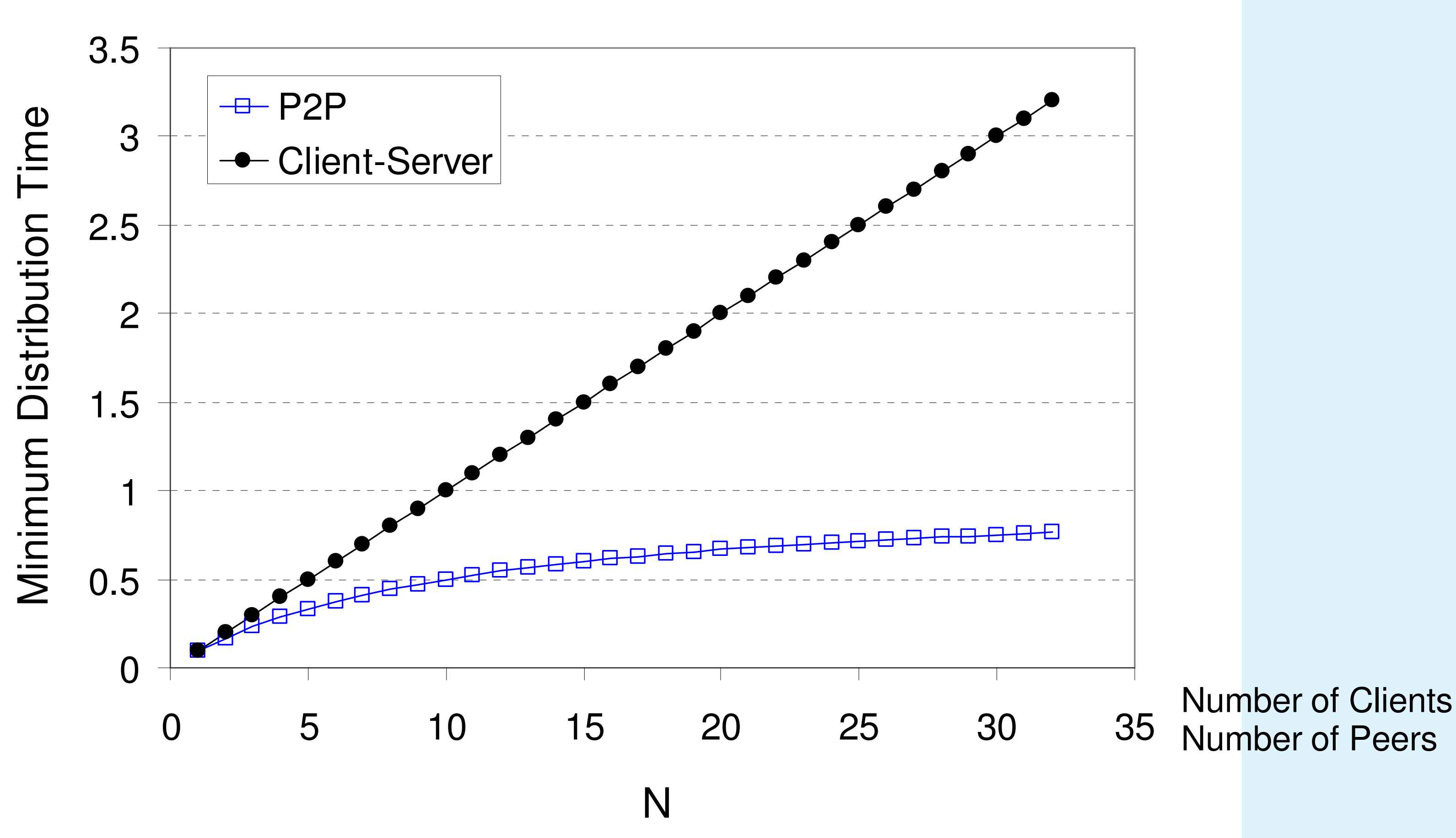
- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time
  - TLD servers typically cached in local name servers
    - Thus root name servers not often visited

Pure P2P architecture

- *No* always-on server
- Arbitrary end-systems directly communicate
- Peers are intermittently connected and change IP addresses

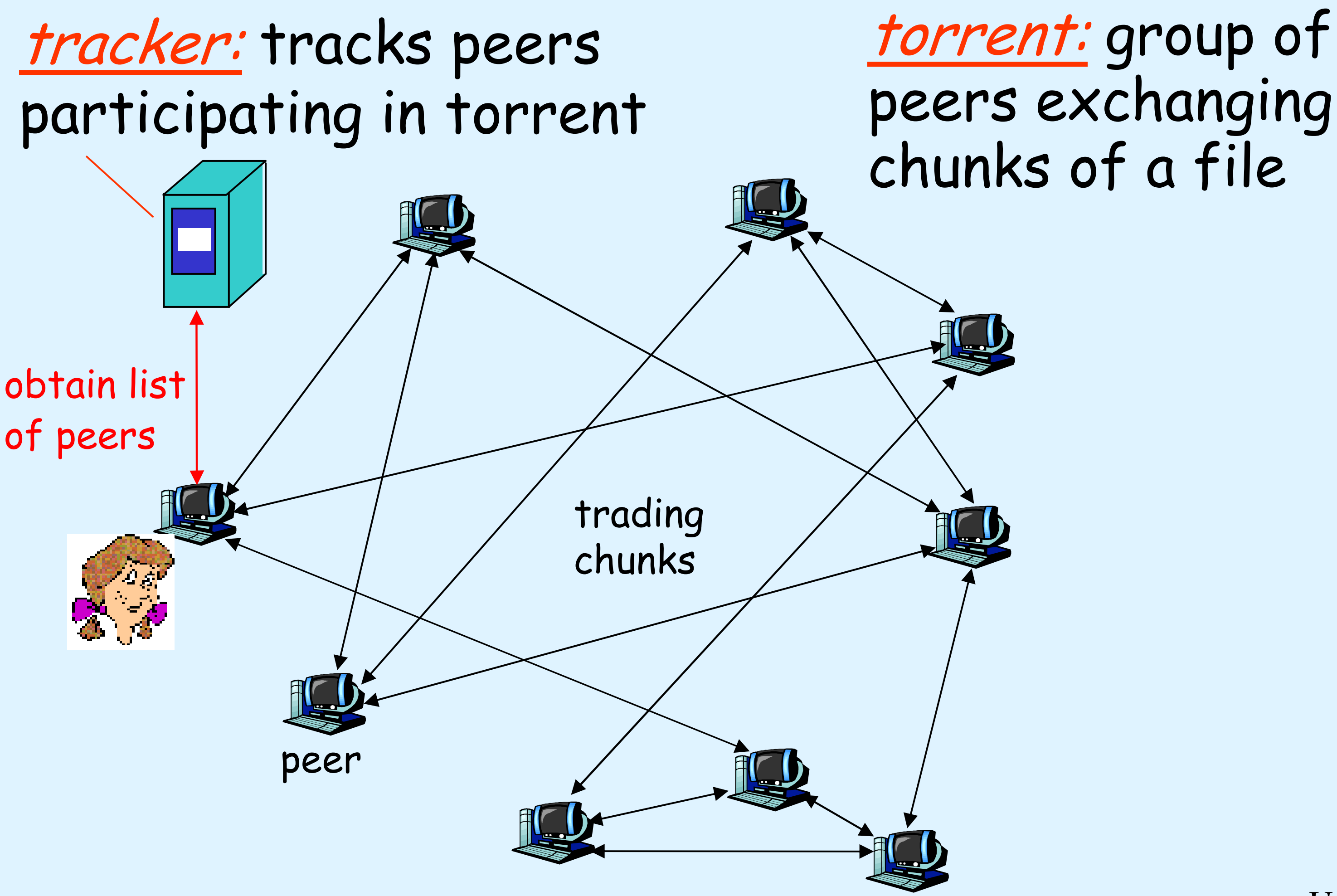


Server-client vs. P2P: example

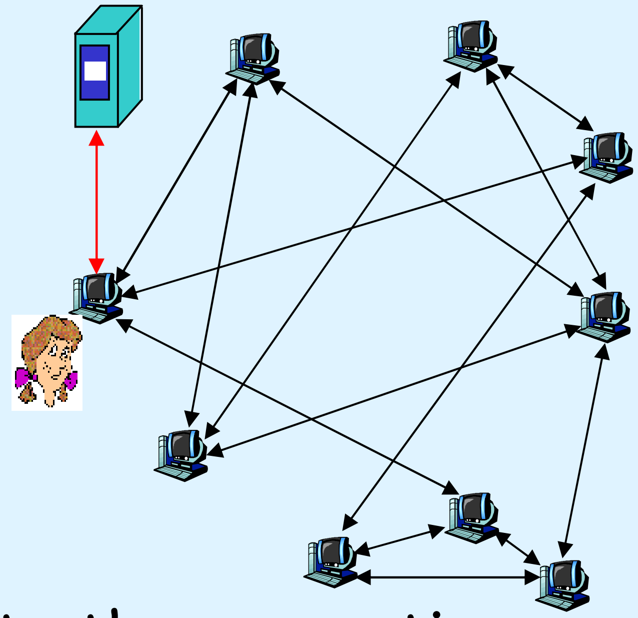


File distribution: BitTorrent

- P2P file distribution



BitTorrent (1)



- Peer joining torrent:
- File divided into 256 KB *chunks*.
  - –has no chunks, but will accumulate them over time
  - –registers with tracker to get list of peers, connects to subset of peers (“neighbors”)
- While downloading, peer uploads chunks to other peers.
- Peers may come and go
- Once peer has entire file, it may (selfishly) leave or (altruistically) remain

BitTorrent (2)

Pulling Chunks

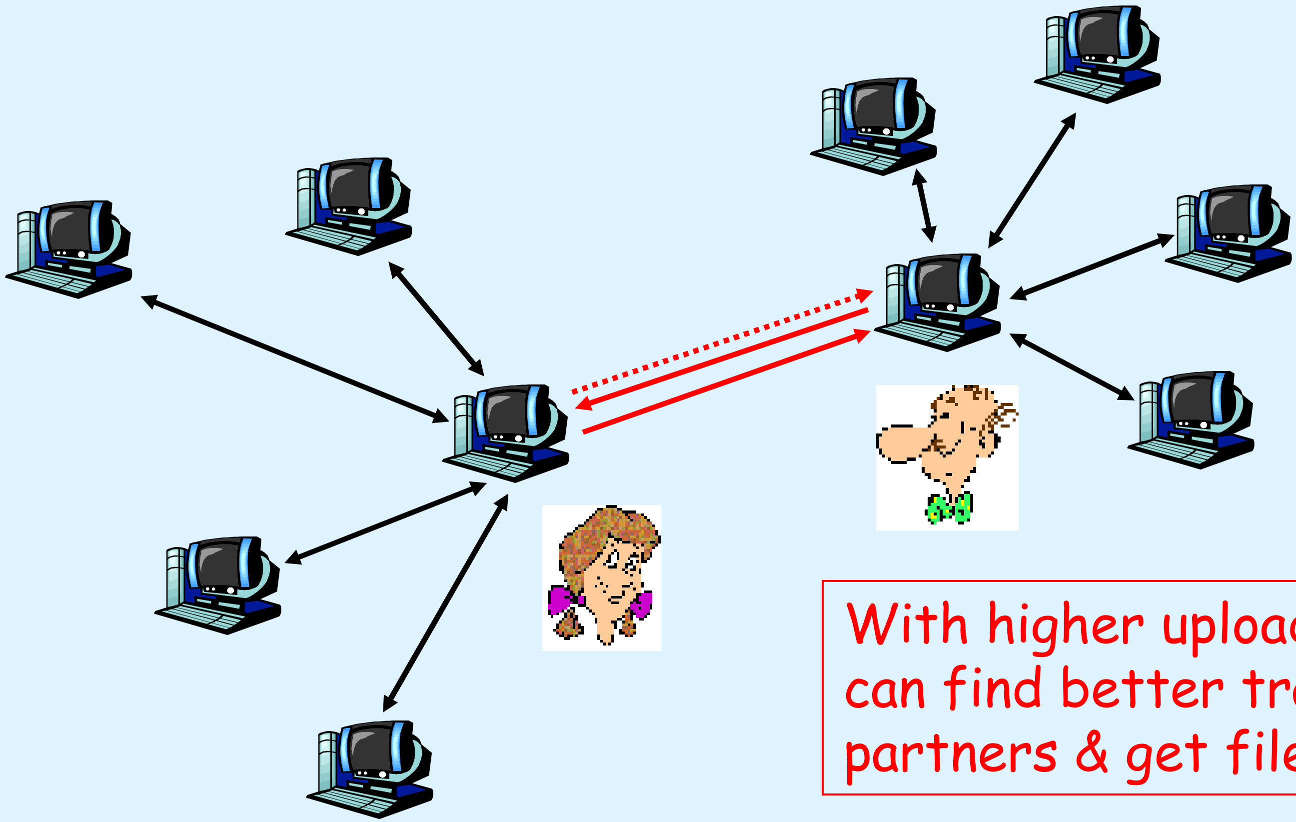
- At any given time, different peers have different subsets of file chunks
- Periodically, a peer (Alice) asks each neighbor for list of chunks that they have.
- Alice sends requests for her missing chunks
  - –rarest first

Sending Chunks: tit-for-tat

- Alice sends chunks to four neighbors currently sending her chunks *at the highest rate*
  - re-evaluate top 4 every 10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - newly chosen peer may join top 4
  - "optimistically unchoke"

BitTorrent: Tit-for-tat

(1) Alice “optimistically unchokes” Bob
(2) Alice becomes one of Bob’s top-four providers; Bob reciprocates
(3) Bob becomes one of Alice’s top-four providers



With higher upload rate, can find better trading partners & get file faster!

Distributed Hash Table (DHT)

- DHT = distributed P2P database
- Database has (key, value) pairs;
  key: ss number; value: human name
  key: content type; value: IP address
- Peers query DB with key
  DB returns values that match the key
- Peers can also insert (key, value) peers

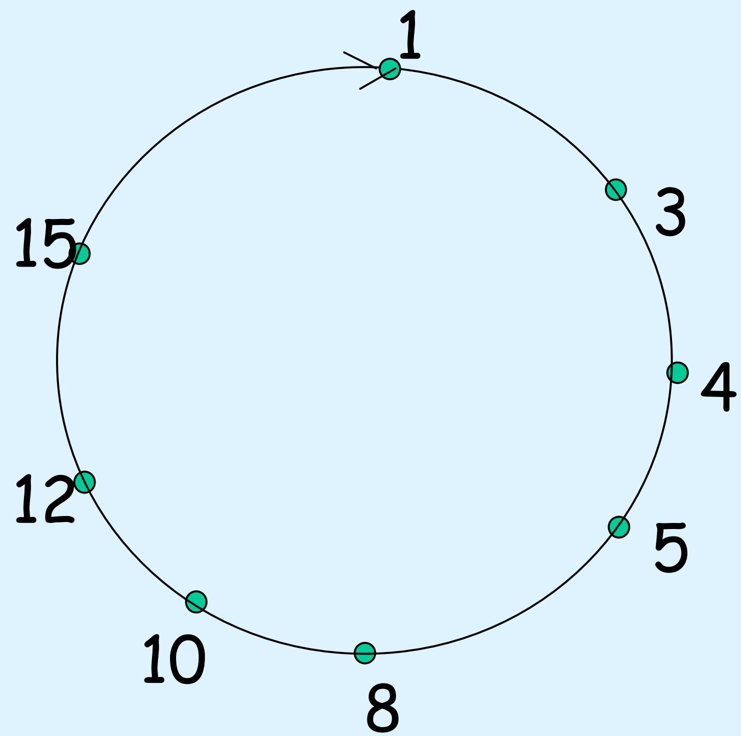
DHT Identifiers

- Assign integer identifier to each peer in range $[0, 2^n-1]$.
- Each identifier can be represented by n bits.
- Require each key to be an integer in same range.
- To get integer keys, hash original key, eg, key = H("Led Zeppelin IV")
- This is why they call it a distributed "hash" table.

How to assign keys to peers?

- Central issue:
  - Assigning (key, value) pairs to peers.
- Rule: assign key to the peer that has the closest ID.
- Convention in lecture: closest is the immediate successor of the key.
- Ex: $n = 4$; peers: 1,3,4,5,8,10,12,15;
  key = 13, then successor peer = 15
  key = 15, then successor peer = 1

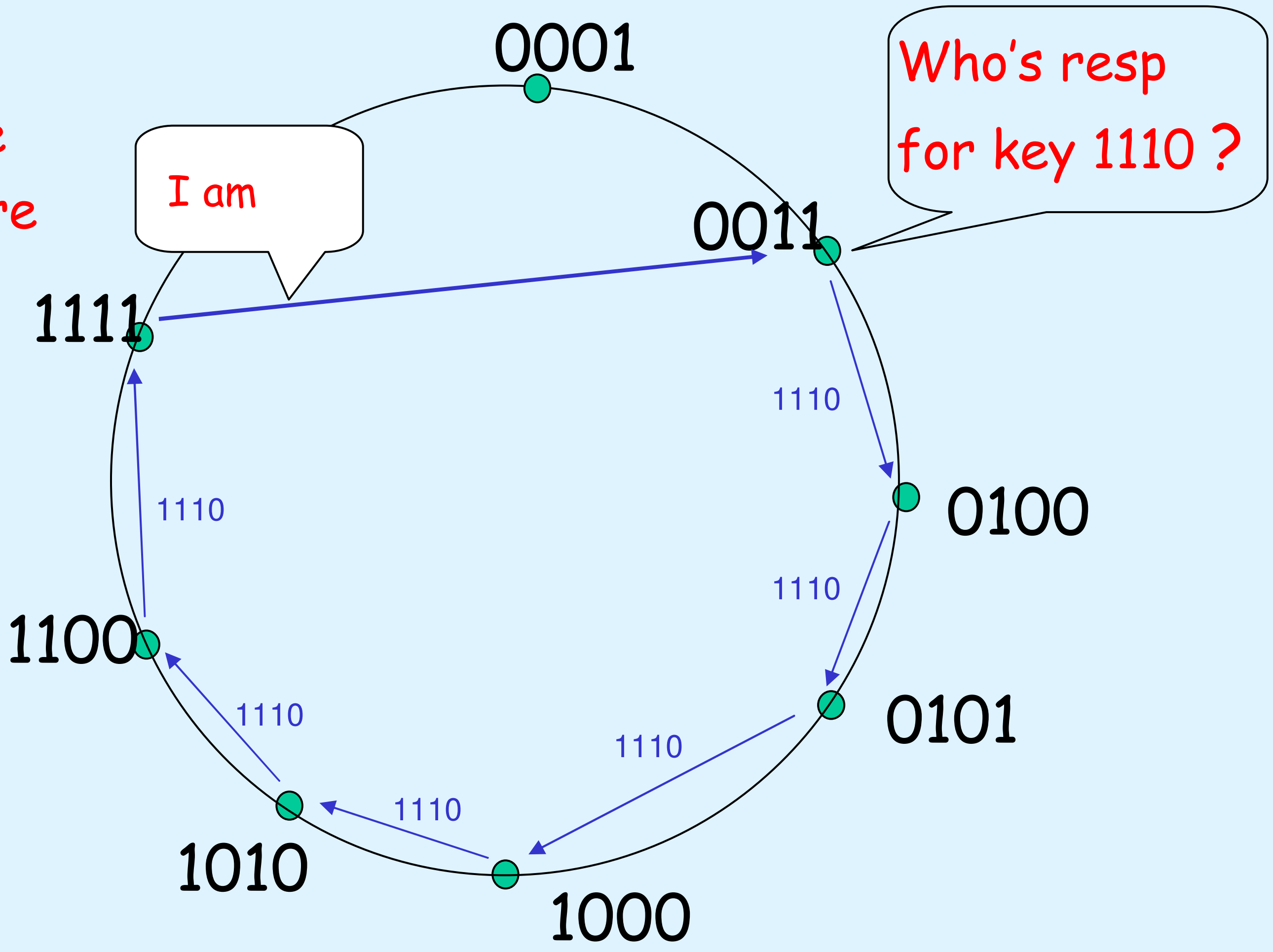
Circular DHT (1)



- Each peer *only* aware of immediate successor and predecessor.
- "Overlay network"

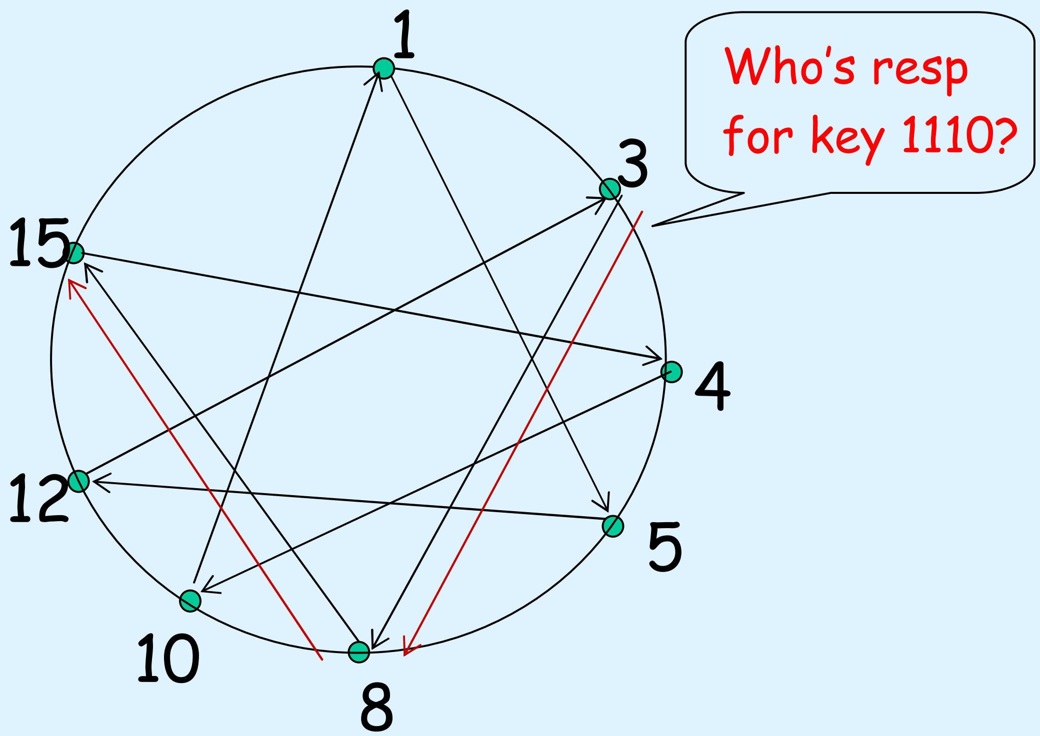
Circle DHT (2)

$O(N)$ messages on avg to resolve query, when there are N peers



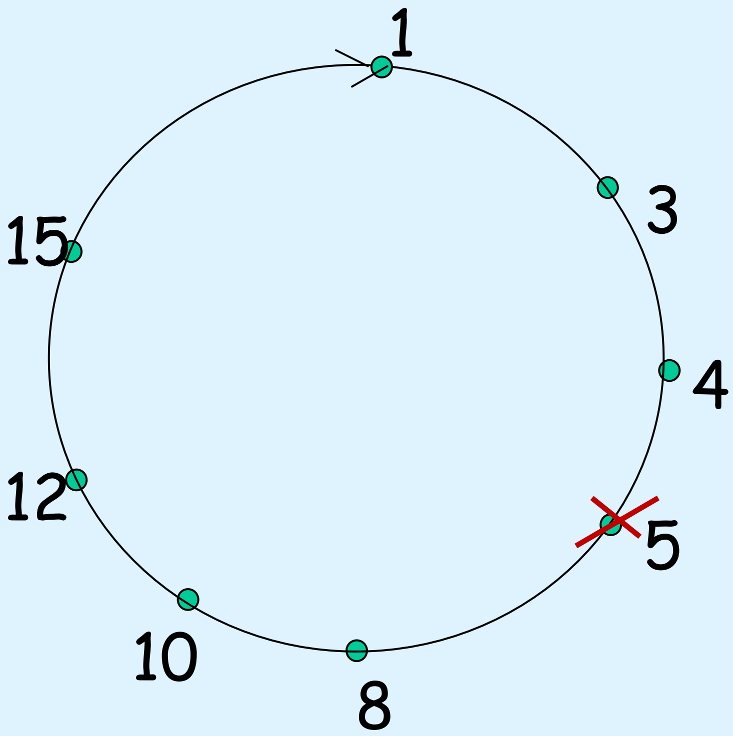
Define closest as closest successor

Circular DHT with Shortcuts



- Each peer keeps track of IP addresses of predecessor, successor, short cuts.
- Reduced from 6 to 2 messages.
- Possible to design shortcuts so $O(\log N)$ neighbors, $O(\log N)$ messages in query

Peer Churn

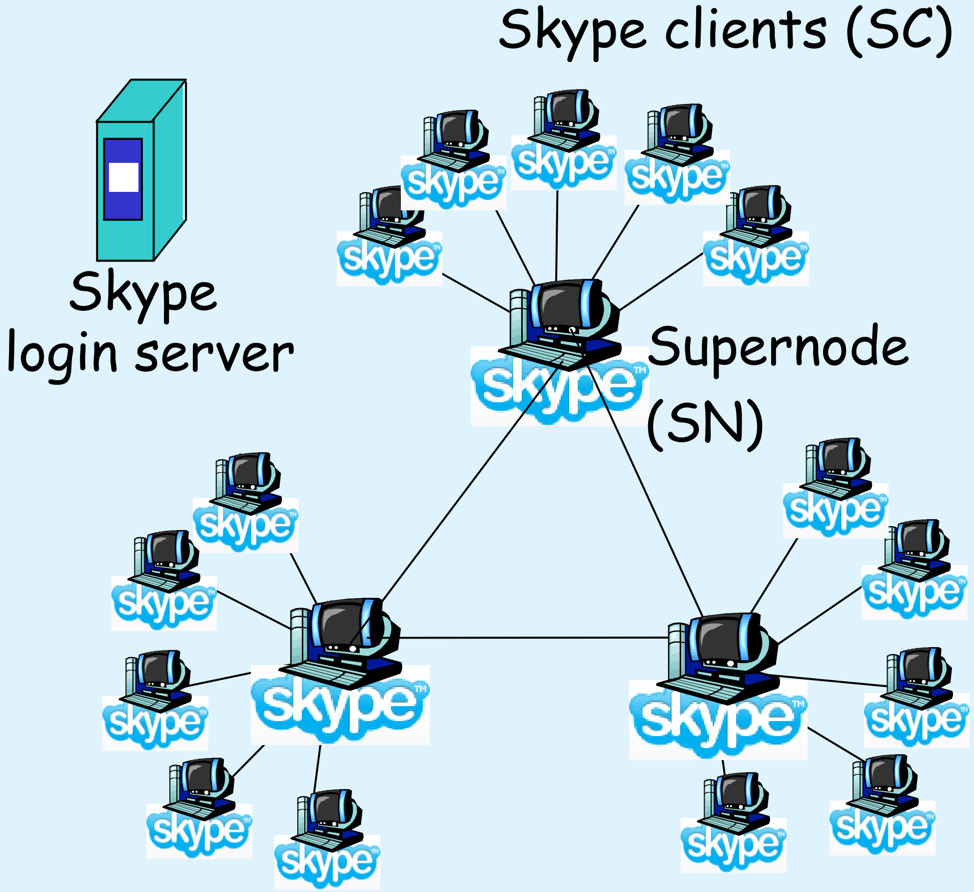


- To handle peer churn, require each peer to know the IP address of its two successors.
- Each peer periodically pings its two successors to see if they are still alive.

1. Peer 5 abruptly leaves
2. Peer 4 detects; makes 8 its immediate successor; asks 8 who its immediate successor is; makes 8's immediate successor its second successor.
3. What if peer 13 wants to join?

P2P Case study: Skype

- Inherently P2P: pairs of users communicate.
- Proprietary application-layer protocol (inferred via reverse engineering)
- Hierarchical overlay with SNs
- Index maps usernames to IP addresses; distributed over SNs

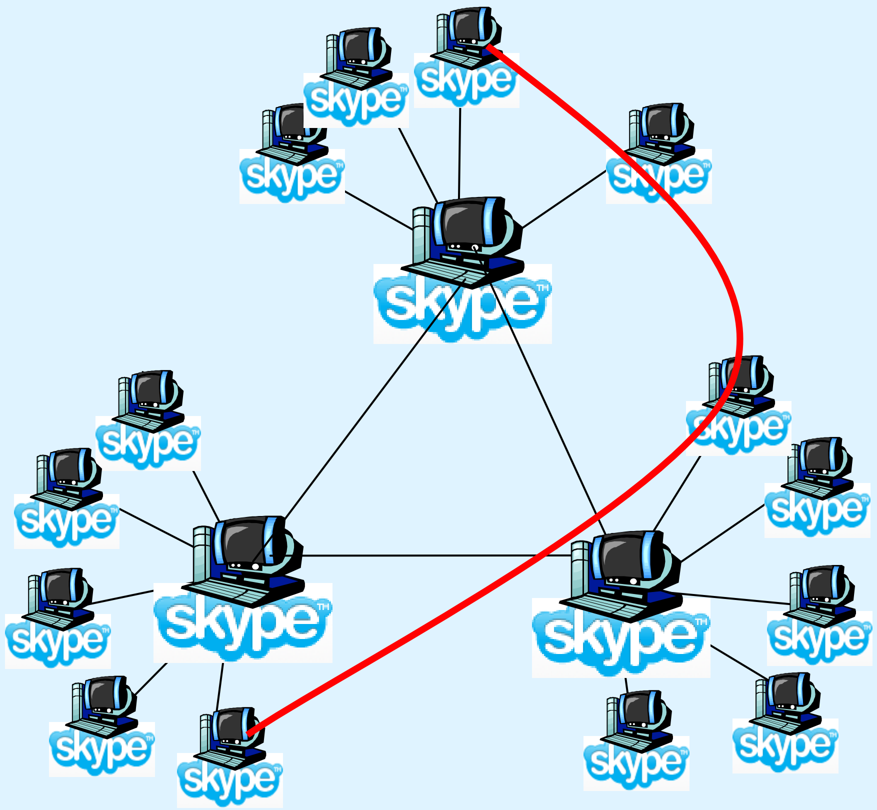


Peers as relays

- Problem when both Alice and Bob are behind "NATs".
- NAT prevents an outside peer from initiating a call to insider peer

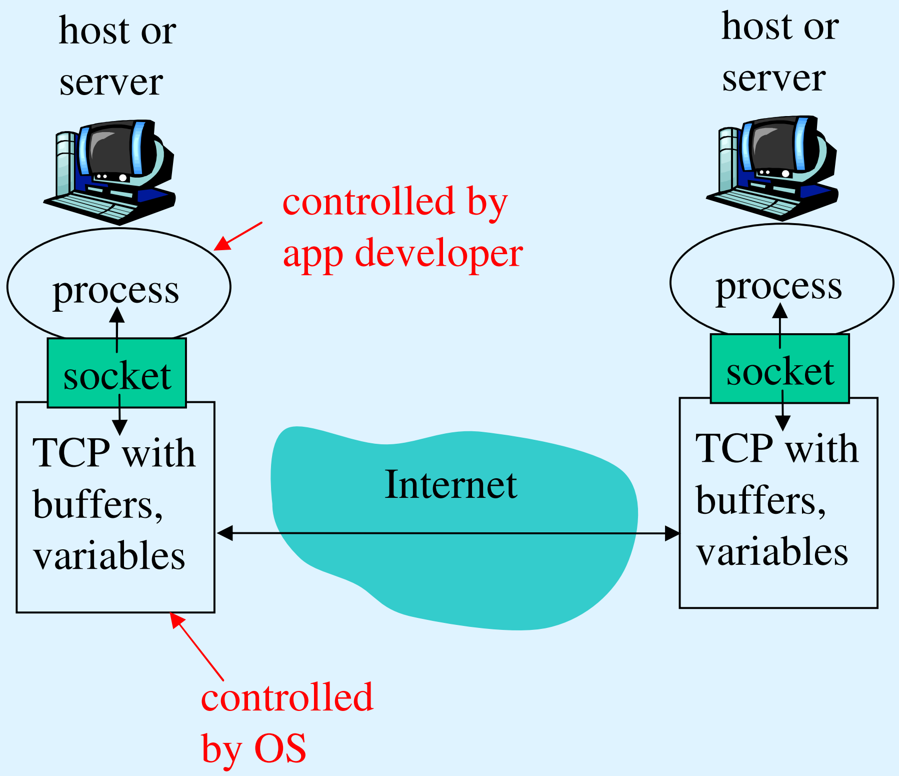
Solution:

1. Using Alice's and Bob's SNs, Relay is chosen
2. Each peer initiates session with relay.
3. Peers can now communicate through NATs via relay



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 which brings message to socket at receiving process



- API: (1) choice of transport protocol: TCP or UDP
  (2) ability to fix a few parameters, e.g. address