Chapter 2
Application Layer

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
- programming network applications
  - socket API

Some network apps

- e-mail
- web
- instant messaging
- remote login
- P2P file sharing
- multi-user network games
- streaming stored video clips
- voice over IP
- real-time video conferencing
- grid computing
-
-
-

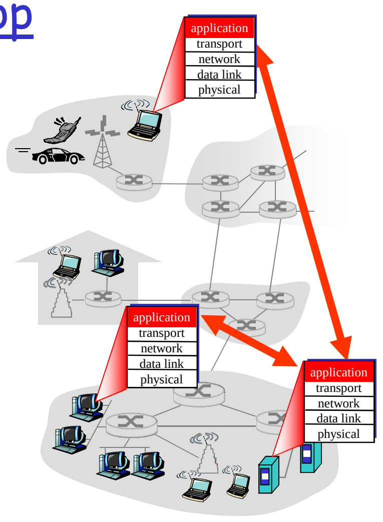
Creating a network app

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

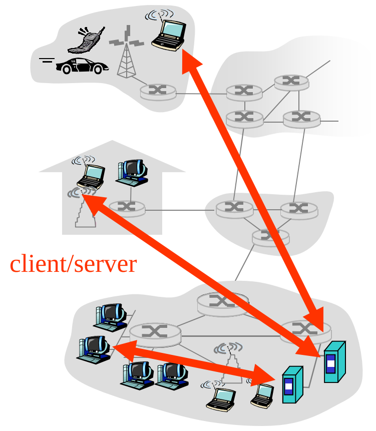
- Network-core devices do not run user applications
- applications on end systems allows for rapid app development, propagation



Application architectures

- Client-server
- Peer-to-peer (P2P)
- Hybrid of client-server and P2P

Client-server architecture

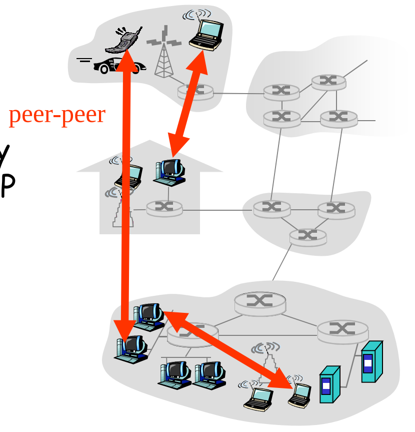


server:
- always-on host
- permanent IP address
- server farms for scaling

clients:
- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

Pure P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses



Highly scalable but
difficult to manage

Hybrid of client-server and P2P

Skype
- voice-over-IP P2P application
- centralized server: finding address of remote party:
- client-client connection: direct (not through server)

Instant messaging
- chatting between two users is P2P
- centralized service: client presence detection/location
  - user registers its IP address with central server when it comes online
  - user contacts central server to find IP addresses of buddies

Communicating Process

Process: program running within a host.

- within same host, two processes communicate using interprocess communication (defined by OS).
- processes running in different hosts communicate with an application-layer protocol

Client process: process that initiates communication

Server process: process that waits to be contacted

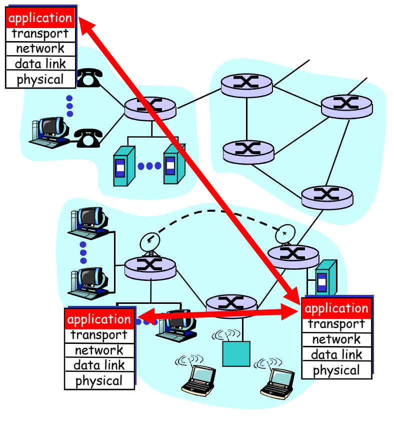
User agent (UA): interfaces with user "above" and network "below"

- Implements user interface & application-level protocol
  - Web: browser
  - E-mail: mail reader

Applications and application-layer protocols

Application: communicating, distributed processes
- e.g., e-mail, Web, P2P file sharing, instant messaging
- running in end systems (hosts)
- exchange messages to implement application

Application-layer protocols
- one "piece" of an app
- define messages exchanged by apps and actions taken
- use communication services provided by lower layer protocols (TCP, UDP)



App-layer protocol defines

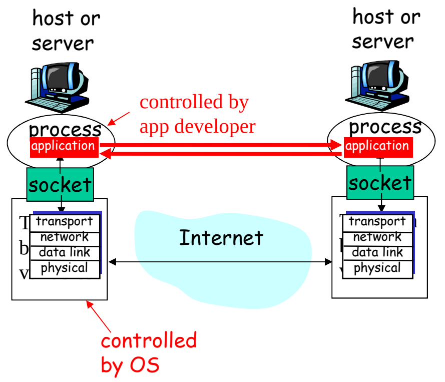
- Types of messages exchanged, eg, request & response messages
- Syntax of message types: what fields in messages & how fields are delineated
- Semantics of the fields, ie, meaning of information in fields
- Rules for when and how processes send & respond to messages

Public-domain protocols:
- defined in RFCs
- allows for interoperability
- eg, HTTP, SMTP

Proprietary protocols:
- eg, KaZaA

Processes communicating across network

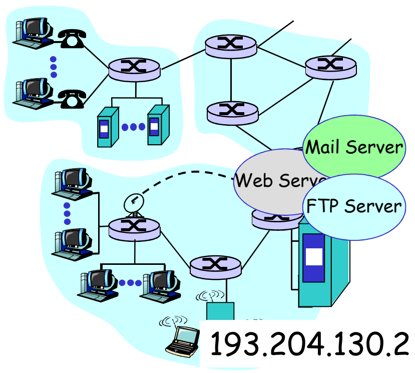
- process sends/receives messages to/from its socket
- Socket: interface between application and transport layers
- socket analogous to door
  - sending process shoves message out door
  - sending process asssumes transport infrastructure on other side of door which brings message to socket at receiving process



- Network API: (1) choice of transport protocol; (2) ability to fix a few parameters

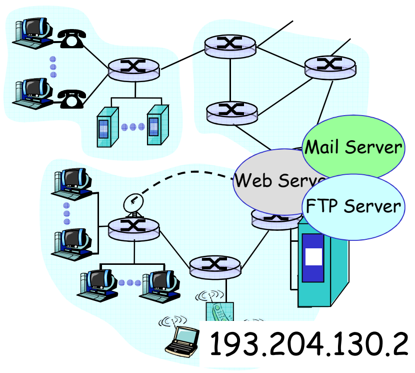
Addressing processes:

- For a process to receive messages, it must have an identifier
- Every host has a unique 32-bit IP address
- Q: does the IP address of the host on which the process runs suffice for identifying the process?
- Answer: No, many processes can be running on same host



Addressing processes:

- Identifier includes both the IP address and port numbers associated with the process on the host.
- Example port numbers:
  - HTTP server: 80
  - Mail server: 25



Port numbers

- 16 bit address (0-65535)
- well known port numbers for common servers
  - FTP 20, TELNET 23, SMTP 25, HTTP 80, POP3 110, ... (full list: RFC 1700)
- number assignment (by IANA)
  - 0 not used
  - 1-255 reserved for well known processes
  - 256-1023 reserved for other processes
  - 1024-65535 dedicated to user apps

What transport service does an app need?

Data loss

- some apps (e.g., audio) can tolerate some loss
- other apps (e.g., file transfer, telnet) require 100% reliable data transfer

Timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

Bandwidth

- some apps (e.g., multimedia) require minimum amount of bandwidth to be "effective"
- other apps ("elastic apps") make use of whatever bandwidth they get

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

Q: why bother? Why is there a UDP?

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	proprietary (e.g. RealNetworks)	TCP or UDP
Internet telephony	proprietary (e.g., Dialpad)	typically UDP

World Wide Web

- CERN (European Center of Nuclear Research)
  - Need to have large teams of dispersed researches collaborate using a constantly changing collection of reports, drawings, photos, etc.
- March '89 Tim Berners-Lee proposes a network application to access a "web" of linked documents.

  "I just had to take te hypertext idea and connect it to TCP and DNS ideas and –ta-da!- the World Wide Web"

World Wide Web

- 1945: Vannevar Bush in 'As we may think' proposes Memex to extend human memory via mechanical means. V. Bush wanted to make the existing store of knowledge (fastly growing) more accessible to mankind.
  - 'A memex is a device in which an individual stores all his books, records, and communications, and which is mechanized so that it maybe consulted with exceeding speed and flexibility. It is an enlarged supplement to his memory. …Wholly new forms of encyclopedias will appear ready made with a mesh of associative trails running through them'
- 1965 term 'hypertext' introduced by Ted Nelson to describe non sequential writing that presents information as a collection of linked nodes. 'Readers can pursue the information in a variety of ways by navigating from one node to another'

World Wide Web

- 1991 First browser and server introduced
- 1993 The Web consisted of around 50 servers
- 1993 First release of Mosaic browser. The web accounted for 1% of the traffic of the Internet
- Late 1990s Web responsible for over 75% of the Internet traffic!! Hundreds millions of users; millions of web sites. Reasons for success: graphical user interface, ease of publishing new content.

Web Content

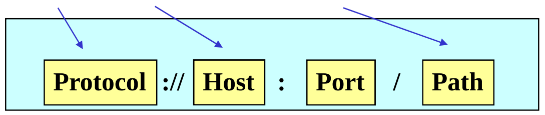
- The Web is a collection of resources/objects distributed throughout the Internet. Each object/resource maybe a static file on a machine or maybe dynamically generated upon request.
- A web page consists of a container resource such as HTML file which may include links to one or more embedded resources such as images and animations.

Basic 'bricks' of the Web

- Uniform Resource Location (URL) –allows to identify a web resource
- Hypertext Markup Language (HTML) – provides a standard representation for hypertext documents in ASCII format. Allows authors to format text, reference images, embed hypertext links.
- Hypertext Transfer Protocol (HTTP)—it is the protocol web components use to communicate.

The three components of an URL

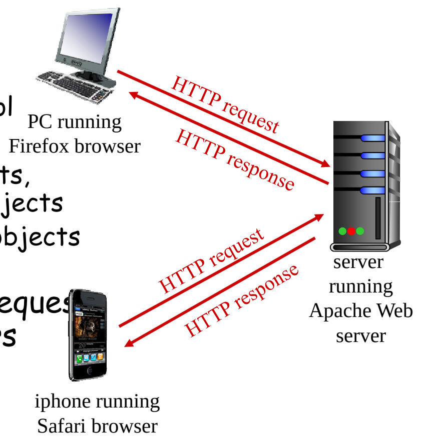
1. *Protocol (also called "scheme")*
   - *how can a page be accessed?* **(application protocol used)**
     - **http**://www.di.univaq.it/lopresti/index.html
2. *Host name*
   - *Where is the page located?* **(symbolic or IP address)**
     - **http://www.di.univaq.it/lopresti/index.html**
3. *File (resource) name*
   - *What is the page called?* **(with full path)**
     - http://www.di.univaq.it**/lopresti/index.html**



HTTP overview

HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, "displays" Web objects
  - *server:* Web server sends objects in response to requests
- defines how web clients reques web pages and how servers transfer pages to clients
- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2068



PC running Firefox browser
HTTP request
HTTP response
server running Apache Web server
HTTP request
HTTP response
iphone running Safari browser

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

HTTP connections

Nonpersistent HTTP

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

Persistent HTTP

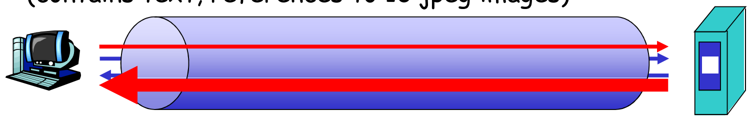
- Multiple objects can be sent over single TCP connection between client and server.
- HTTP/1.1 uses persistent connections in default mode

Nonpersistent HTTP

Suppose user enters URL

www.someSchool.edu/someDepartment/home.index

(contains text, references to 10 jpeg images)



[illegible]

someDepartment/home.index

Nonpersistent 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

Nonpersistent HTTP (cont.)

4. HTTP server closes TCP connection.

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

time

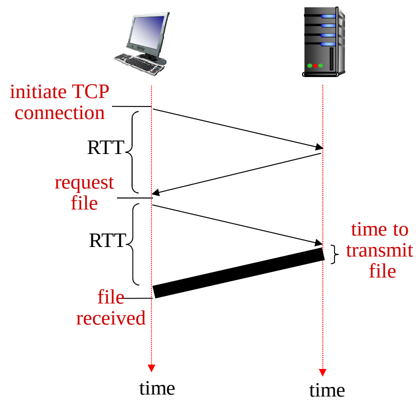
Response time modeling

Definition of RTT: time to send a small packet to travel from client to server and back.

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

total = 2RTT+transmit time



Persistent HTTP

Nonpersistent HTTP issues:
- requires 2 RTTs per object
- OS must work and allocate host resources for each TCP connection
- but browsers often open parallel TCP connections to fetch referenced objects

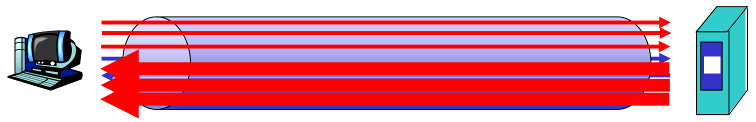
Persistent HTTP
- server leaves connection open after sending response
- subsequent HTTP messages between same client/server are sent over connection

Persistent without pipelining:
- client issues new request only when previous response has been received
- one RTT for each referenced object

Persistent with pipelining:
- 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

Persistent HTTP without pipelining

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



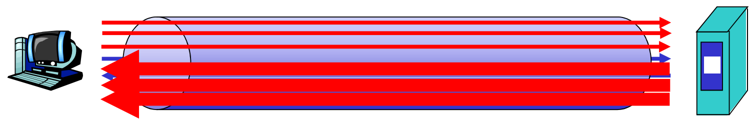
[illegible] HTTP client [illegible] [illegible] someDepartment/home.index

[illegible] HTTP server [illegible] socket

...and so on until all objects are retrieved from the server

Persistent HTTP with pipelining

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



[illegible] HTTP client [illegible] someDepartment/home.index

[illegible] HTTP server [illegible] socket

...and so on until all objects are retrieved from the server

HTTP request message

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

request line (GET, POST, HEAD commands)

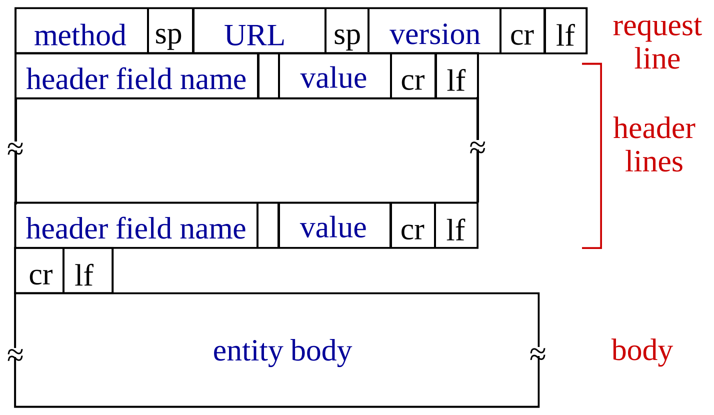
```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

header lines

(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

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

HTTP response message

status line (protocol status code status phrase)

```
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 ...
```

header lines

data, e.g., requested HTML file

HTTP response status codes

In first line in server->client response message.

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

Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet www.eurecom.fr 80
```

Opens TCP connection to port 80 (default HTTP server port) at www.eurecom.fr. Anything typed in sent to port 80 at www.eurecom.fr

2. Type in a GET HTTP request:

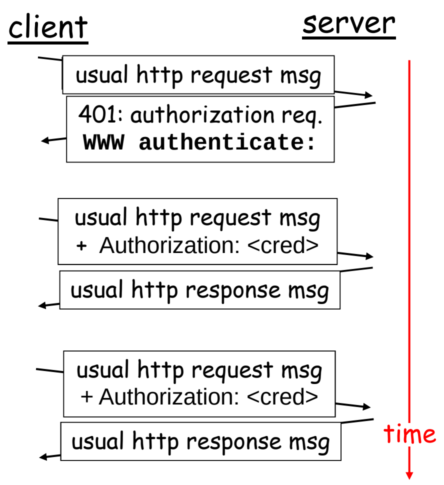
```
GET /~ross/index.html HTTP/1.0
```

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

3. Look at response message sent by HTTP server!

User-server interaction: authorization

Authorization : control access to server content
- authorization credentials: typically name, password
- stateless: client must present authorization in *each* request
  - authorization: header line in each request
  - if no authorization: header, server refuses access, sends
    WWW authenticate:
    header line in response



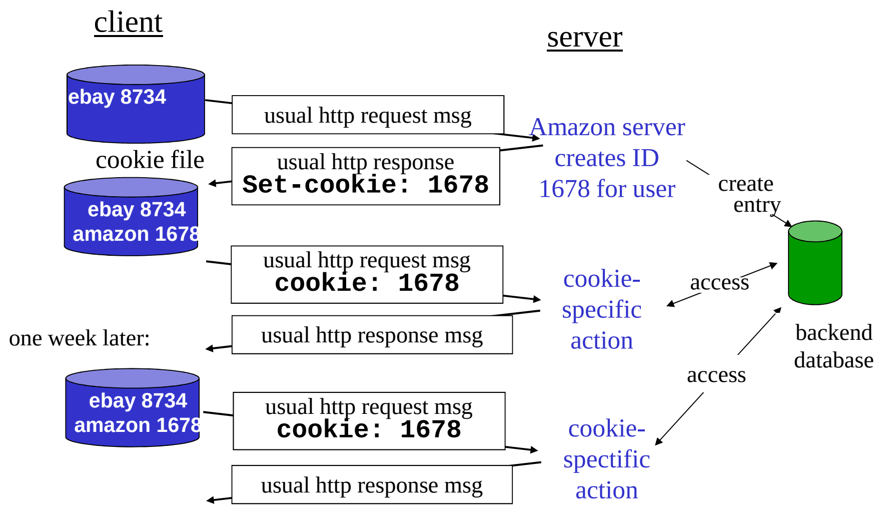
User-server state: cookies

Many major Web sites use cookies

Four components:
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

Example:
- Susan always access Internet always from 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 bring:
- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

aside

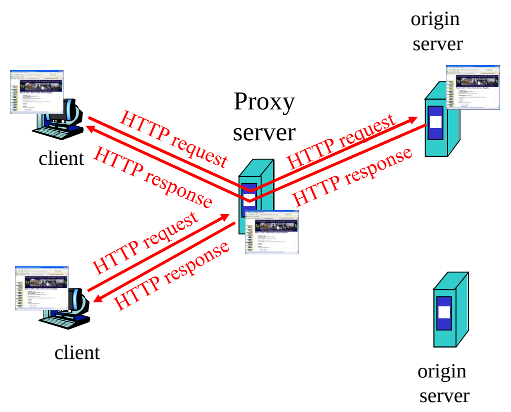
Cookies and privacy:
- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites

How to keep "state":
- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state

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
- 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)

Caching example

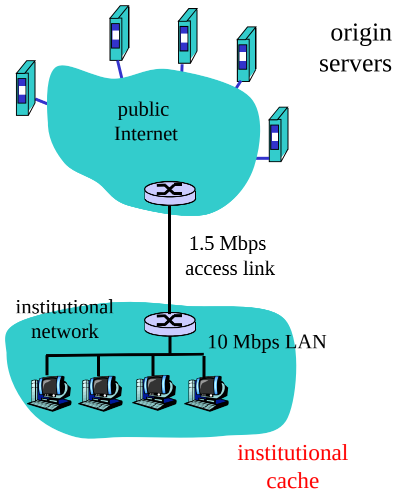
Assumptions

- average object size = 100,000 bits
- avg. request rate from institution's browsers to origin servers = 15/sec
- delay from institutional router to any origin server and back to router = 2 sec

Consequences

- utilization on LAN = 15%
- utilization on access link = 100%
- total delay = Internet delay + access delay + LAN delay

= 2 sec + minutes + milliseconds



Caching example (cont)

possible solution

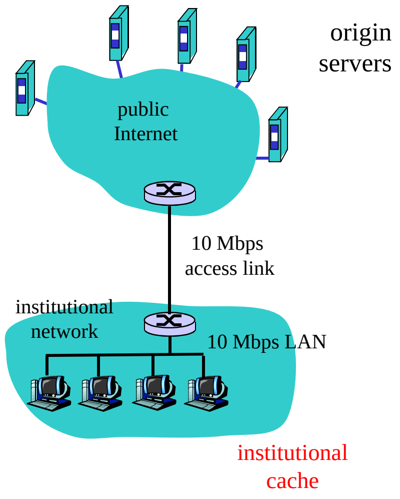
- increase bandwidth of access link to, say, 10 Mbps

consequence

- utilization on LAN = 15%
- utilization on access link = 15%
- Total delay = Internet delay + access delay + LAN delay

= 2 sec + msecs + msecs

- often a costly upgrade



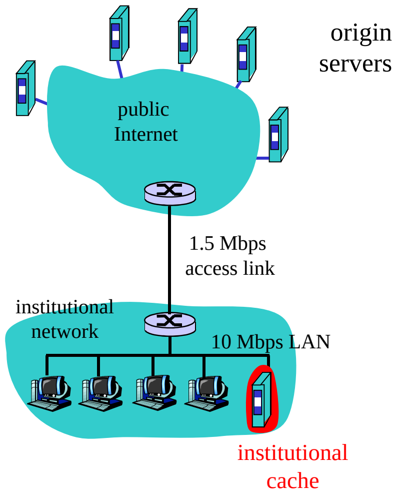
Caching example (cont)

possible solution: install cache

- suppose hit rate is 0.4

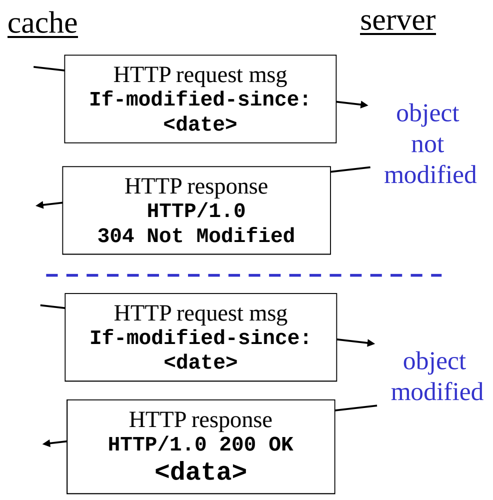
consequence

- 40% requests will be satisfied almost immediately
- 60% requests satisfied by origin server
- utilization of access link reduced to 60%, resulting in negligible delays (say 10 msec)
- total avg delay = Internet delay + access delay + LAN delay = .6*(2.01) secs + .4*milliseconds < 1.4 secs

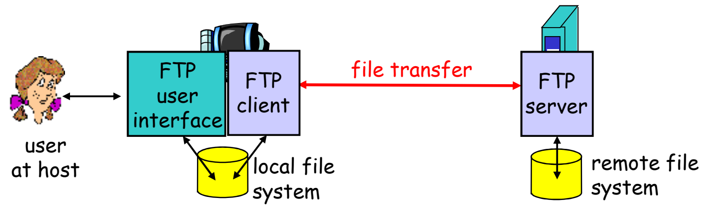


Conditional GET

- 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`



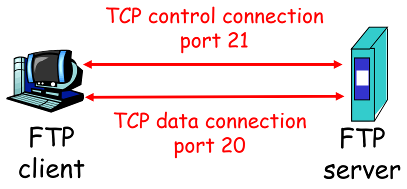
FTP: the file transfer protocol



- 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 server: port 21

FTP: separate control, data connections

- FTP client contacts FTP server at port 21, specifying TCP as transport protocol
- Client obtains authorization over control connection
- Client browses remote directory by sending commands over control connection.
- When server receives a command for a file transfer, the server opens a TCP data connection to client
- After transferring one file, server closes connection.



- Server opens a second 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`

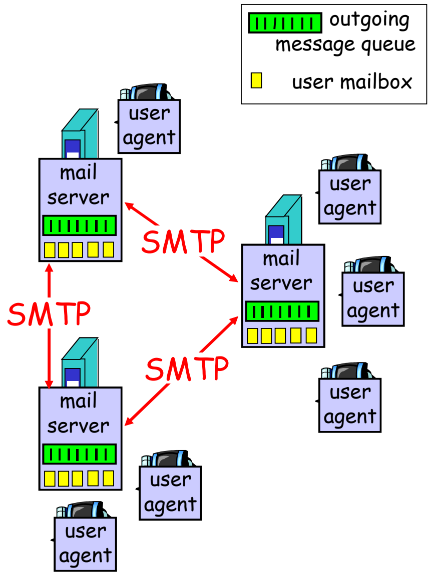
Electronic Mail

Three major components:

- user agents
- mail servers
- simple mail transfer protocol: SMTP

User Agent

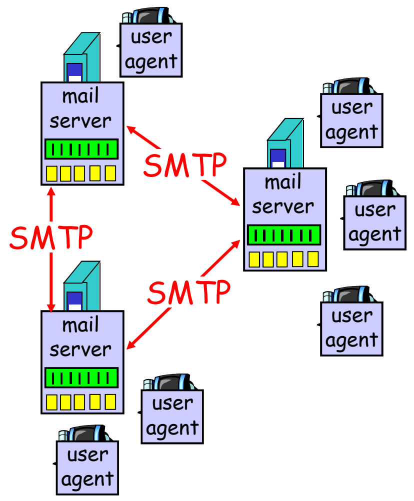
- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Eudora, Outlook, elm, Netscape Messenger
- outgoing, incoming messages stored on server



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

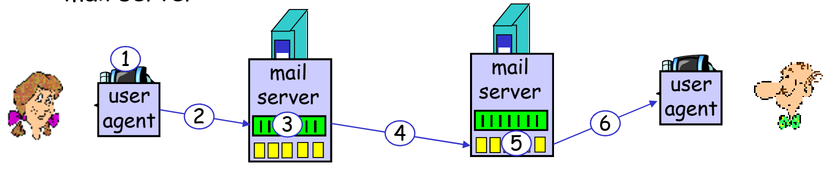


Electronic Mail: SMTP [RFC 2821]

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

Scenario: Alice sends message to Bob

1) Alice uses UA to compose message and "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
```

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

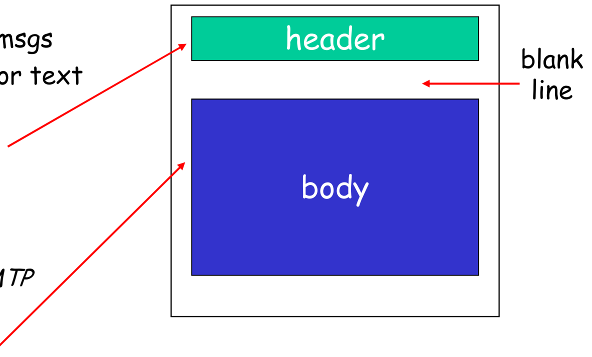
Mail message format

SMTP: protocol for exchanging email msgs

RFC 822: standard for text message format:

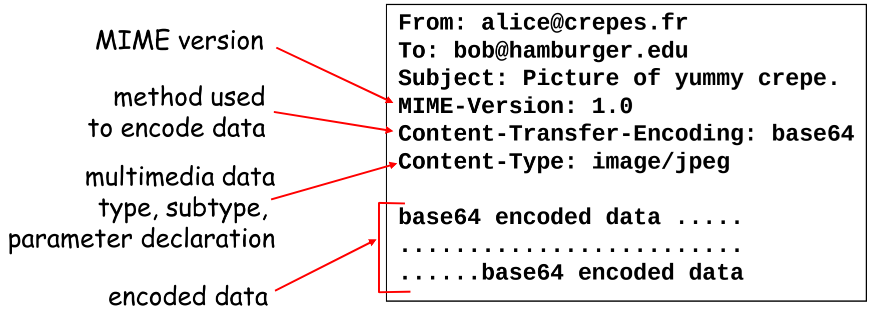
- header lines, e.g.,
  - To:
  - From:
  - Subject:

  different from SMTP commands!
- body
  - the "message", ASCII characters only



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
```

MIME types

Content-Type: type/subtype; parameters

Text

- example subtypes: **plain, html**

Image

- example subtypes: **jpeg, gif**

Audio

- exampe subtypes: **basic** (8-bit mu-law encoded), **32kadpcm** (32 kbps coding)

Video

- example subtypes: **mpeg, quicktime**

Application

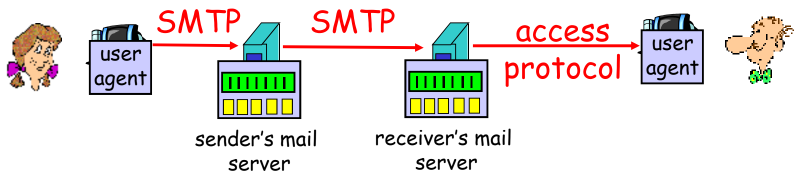
- other data that must be processed by reader before "viewable"
- example subtypes: **msword, octet-stream**

Multipart Type

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=StartOfNextPart

--StartOfNextPart
Dear Bob, Please find a picture of a crepe.
--StartOfNextPart
Content-Transfer-Encoding: base64
Content-Type: image/jpeg
base64 encoded data .....
.........................
......base64 encoded data
--StartOfNextPart
Do you want the reciple?
```

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: 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**

update phase
- Server deletes messages marked for deletion

```
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 "download and delete" mode.
- Bob cannot re-read e-mail if he changes client
- "Download-and-keep": copies of messages on different clients
- POP3 is stateless across sessions

IMAP
- Keep all messages in one place: the server
- Allows user to organize messages in folders
- IMAP keeps user state across sessions:
  - names of folders and mappings between message IDs and folder name

DNS: Domain Name System

People: many identifiers:
- SSN, name, passport #

Internet hosts, routers:
- IP address (32 bit) - used for addressing datagrams
- "name", e.g., ww.yahoo.com - used by humans

Q: map between IP addresses and name ?

Domain Name System:
- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol* host, routers, name servers to communicate to *resolve* names (address/name translation)
  - note: core Internet function, implemented as application-layer protocol
  - complexity at network's "edge"

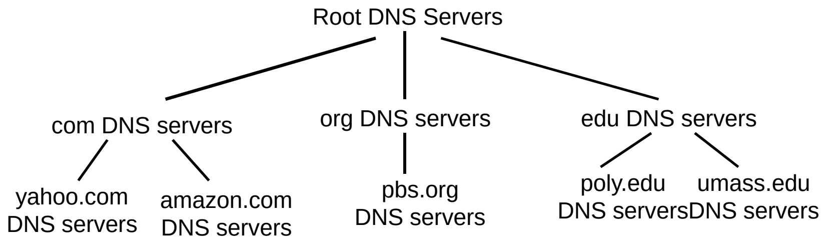
DNS

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!*

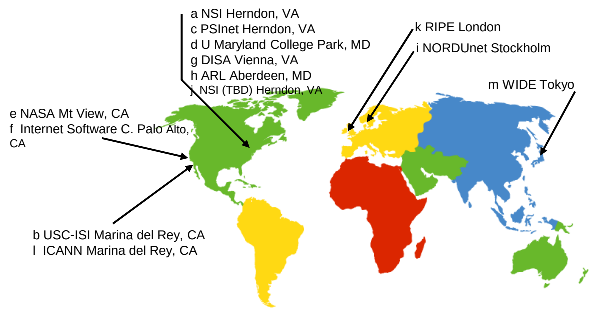
Distributed, Hierarchical Database



Client wants IP for www.amazon.com; 1st approx:
- client queries a 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

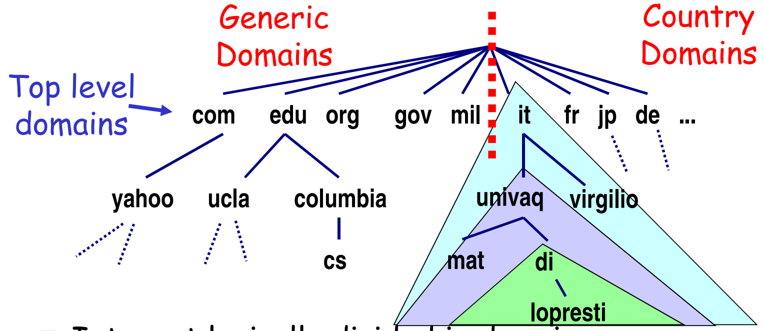
DNS: Root name servers

- contacted by local name server that can not resolve name:
  1. contacts authoritative name server if name mapping not known
  2. gets mapping
  3. returns mapping to local name server



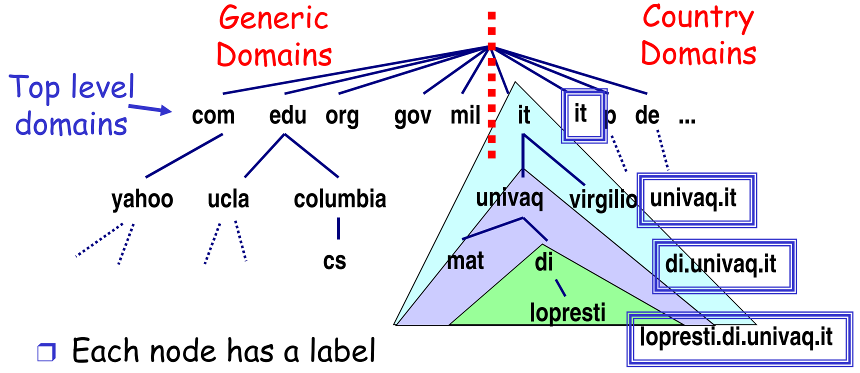
13 root name servers worldwide

DNS: Name Space



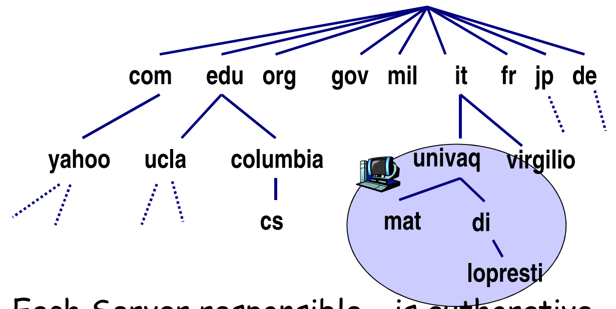
- Internet logically divided in domains
- Domain set of “related” host
  - Same type, country, organization
  - Can be divided in subdomains
- Domains structure can be represented by a tree
  - Each subtree corresponds to a domain

DNS: Name Space



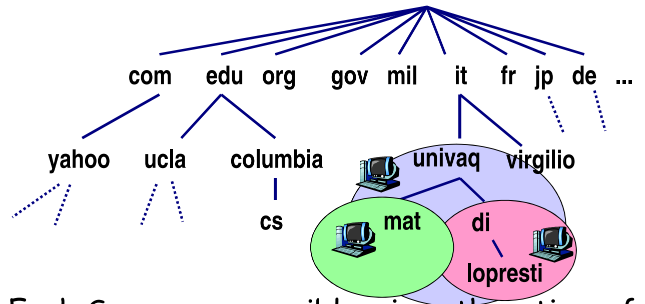
- Each node has a label
- Domain name
  - Sequence of labels from a node up to the root, separated by dots
- Each domain has authority for names in its domain
  - Typically delegated to subdomains

DNS: Name Space



- Each Server responsible – is authorative - for a “zone”
  - Zone – connected subgraph of the tree
  1. Zone=subtree
     - for a host in the zone: stores that host’s IP address, name in the zone file
     - can perform name/address translation for that host’s name

DNS: Name Space



- Each Server responsible – is authorative - for a “zone”
  2. Zone ≠ subtree
     - server delegates authority for (some of) its subdomains to lower level servers
     - Node information in the lower level servers
     - Server keeps reference to these lower level server

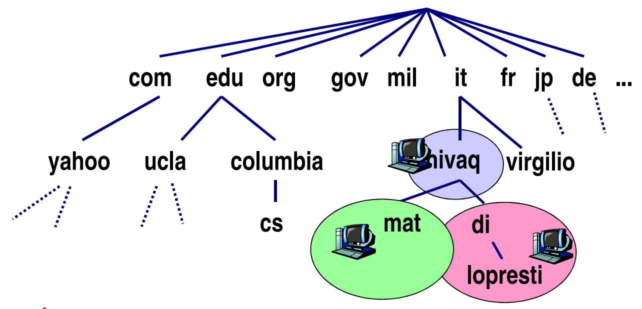
TLD and Authoritative Servers

- Top-level domain (TLD) servers:
  - responsible for com, org, net, edu, etc, and all top-level country domains uk, fr, ca, jp.
  - Network Solutions maintains servers for com TLD
  - Educause for edu TLD
- 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 Space

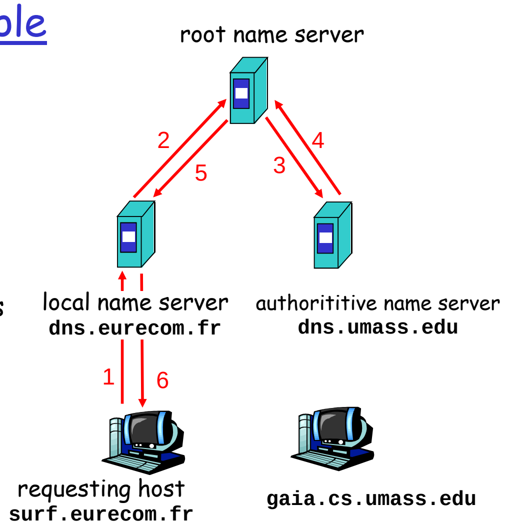


local name servers:
- each ISP, company (domain) has *local (default) name server*
- host DNS query first goes to local name server

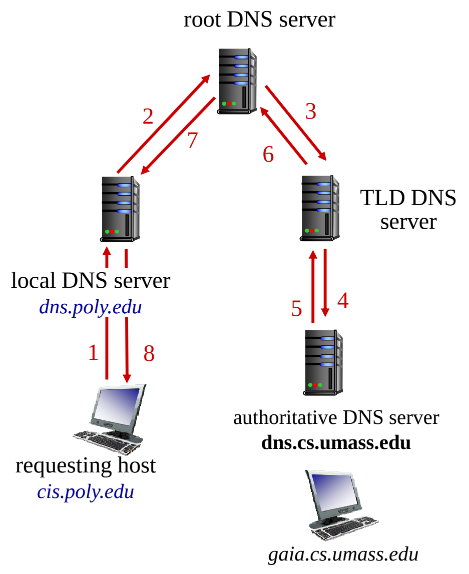
Simple DNS example

host **surf.eurecom.fr** wants IP address of **gaia.cs.umass.edu**

1. contacts its local DNS server, **dns.eurecom.fr**
2. **dns.eurecom.fr** contacts root name server, if necessary
3. root name server contacts authoritative name server, **dns.umass.edu**, if necessary



DNS example



Root name server:

- may not know authoritative name server
- may know *intermediate name server:* who to contact to find authoritative name server

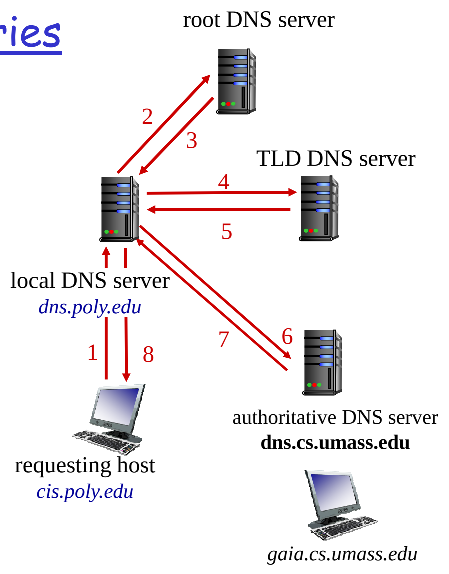
DNS: iterated queries

recursive query:

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

iterated query:

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



DNS: caching and updating records

- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time
- update/notify mechanisms under design by IETF
  - RFC 2136
  - http://www.ietf.org/html.charters/dnsind-charter.html

DNS records

DNS: distributed db storing resource records (RR)

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

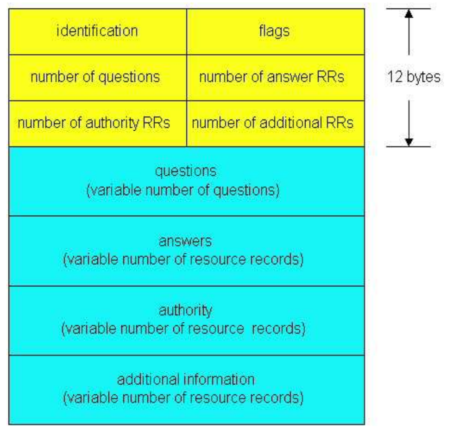
- Type=A
  - **name** is hostname
  - **value** is IP address
- Type=NS
  - **name** is domain (e.g. foo.com)
  - **value** is IP address of authoritative name server for this domain
- Type=CNAME
  - **name** is alias name for some "canonical" (the real) name
    `www.ibm.com` is really `servereast.backup2.ibm.com`
  - **value** is canonical name
- Type=MX
  - **value** is name of mailserver associated with **name**

DNS protocol, messages

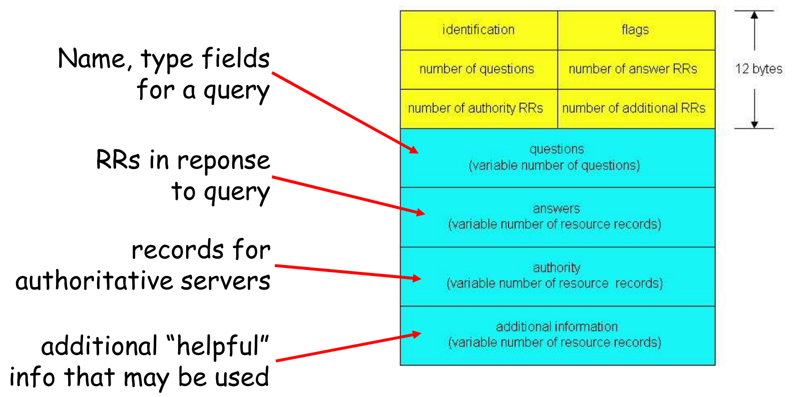
DNS protocol : *query* and *reply* messages, both with same *message format*

msg header

- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative



DNS protocol, messages


Name, type fields for a query
RRs in reponse to query
records for authoritative servers
additional "helpful" info that may be used
identification
flags
number of questions
number of answer RRs
number of authority RRs
number of additional RRs
12 bytes
questions
(variable number of questions)
answers
(variable number of resource records)
authority
(variable number of resource records)
additional information
(variable number of resource records)

Inserting records into DNS

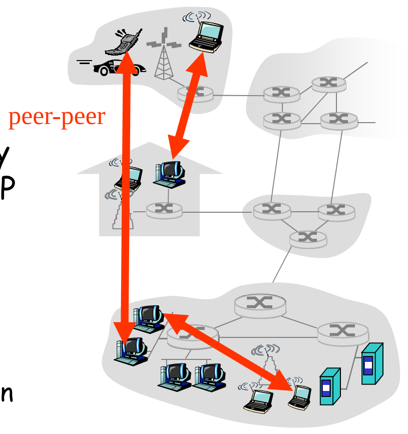
- example: new startup "Network Utopia"
- register name networkuptopia.com at *DNS registrar* (e.g., Network Solutions)
  - provide names, IP addresses of authoritative name server (primary and secondary)
  - registrar inserts two RRs into com TLD server:

```
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)
```

- create authoritative server Type A record for www.networkuptopia.com; Type MX record for networkutopia.com
- How do people get IP address of your Web site?

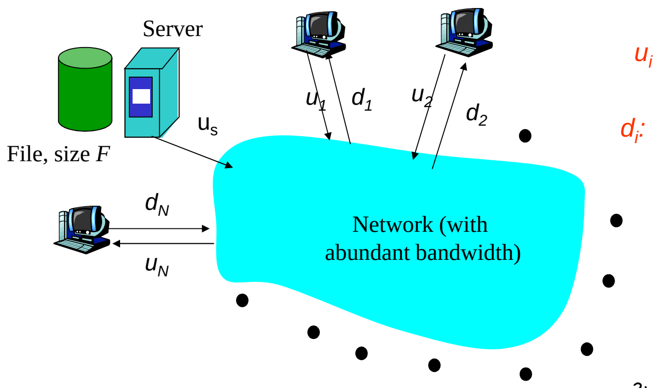
Pure P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses
- Three topics:
  - File distribution
  - Searching for information
  - Case Study: Skype



File Distribution: Server-Client vs P2P

Question : How much time to distribute file from one server to *N* *peers*?



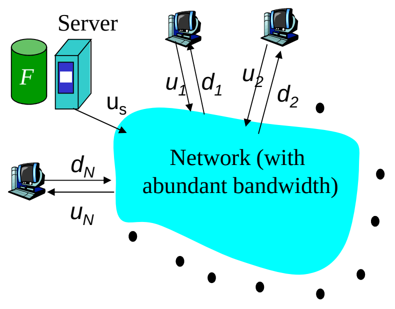
u_s: server upload bandwidth

u_i: peer i upload bandwidth

d_i: peer i download bandwidth

File distribution time: server-client

- server sequentially sends N copies:
  - NF/u_s time
- client i takes F/d_i time to download

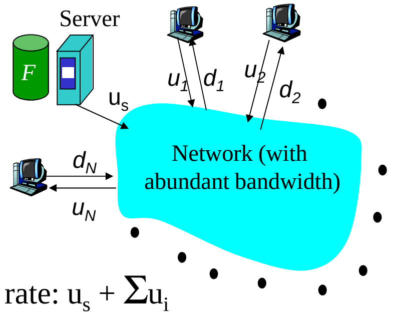


Time to distribute *F* to *N* clients using client/server approach $= d_{cs} = \max \{ NF/u_s, F/\min_i(d_i) \}$

increases linearly in N (for large N)

File distribution time: P2P

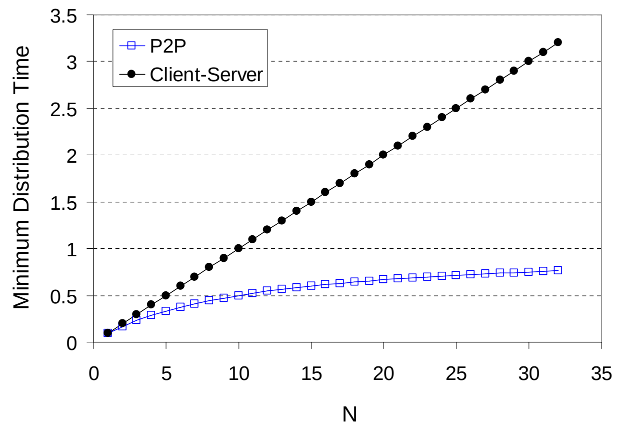
- server must send one copy: F/u_s time
- client i takes F/d_i time to download
- NF bits must be downloaded (aggregate)
  - fastest possible upload rate: $u_s + \Sigma u_i$



$$d_{P2P} = \max \{ F/u_s, F/\min_i(d_i) , NF/(u_s + \Sigma u_i) \}$$

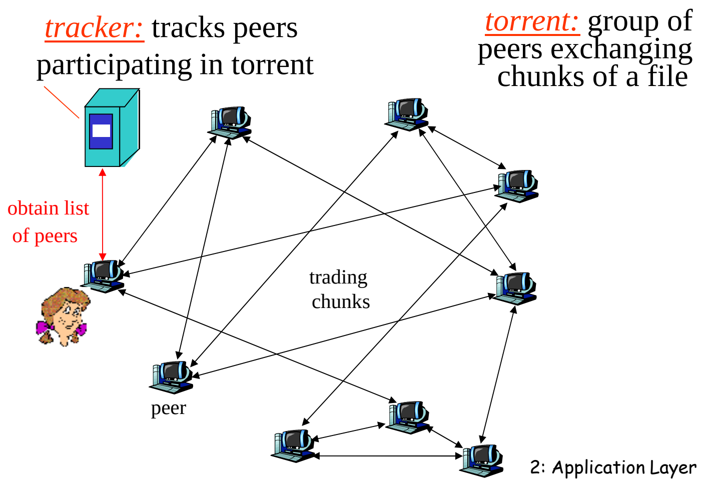
Server-client vs. P2P: example

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

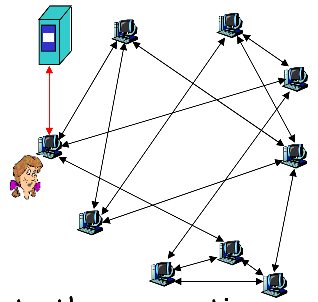


File distribution: BitTorrent

❒ P2P file distribution


tracker: tracks peers participating in torrent
torrent: group of peers exchanging chunks of a file
obtain list of peers
trading chunks
peer

BitTorrent (1)



- ❒ file divided into 256KB *chunks*.
- ❒ peer joining torrent:
  - ❍ 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

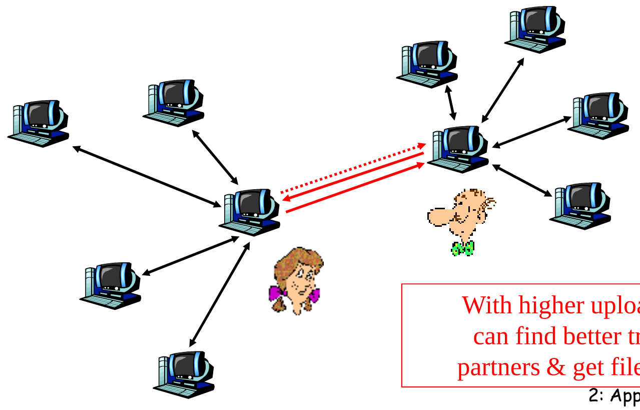
- ❒ at any given time, different peers have different subsets of file chunks
- ❒ periodically, a peer (Alice) asks each neighbor for list of chunks that they have.
- ❒ Alice sends requests for her missing chunks
  - ❍ rarest first

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!

P2P: searching for information

Index in P2P system: maps information to peer location (location = IP address & port number)

File sharing (eg e-mule)

- Index dynamically tracks the locations of files that peers share.
- Peers need to tell index what they have.
- Peers search index to determine where files can be found

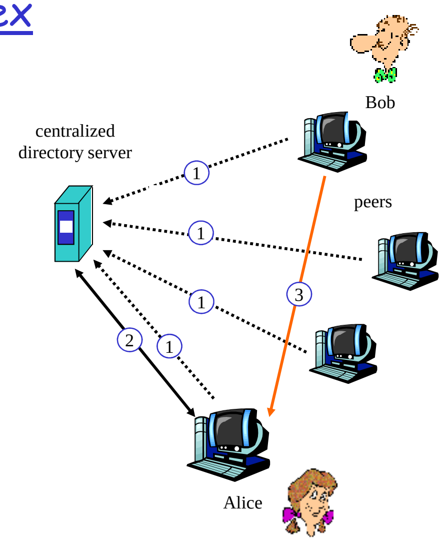
Instant messaging

- Index maps user names to locations.
- When user starts IM application, it needs to inform index of its location
- Peers search index to determine IP address of user.

P2P: centralized index

original "Napster" design

1) when peer connects, it informs central server:
   - IP address
   - content
2) Alice queries for "Hey Jude"
3) Alice requests file from Bob



P2P: problems with centralized directory

- single point of failure
- performance bottleneck
- copyright infringement: "target" of lawsuit is obvious

file transfer is decentralized, but locating content is highly centralized

Query flooding

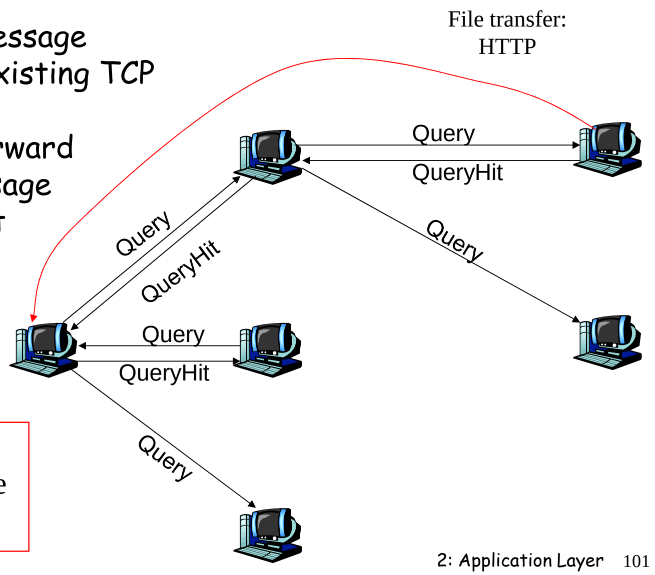
- fully distributed
  - no central server
- used by Gnutella
- Each peer indexes the files it makes available for sharing (and no other files)

overlay network: graph

- edge between peer X and Y if there's a TCP connection
- all active peers and edges form overlay net
- edge: virtual (*not* physical) link
- given peer typically connected with < 10 overlay neighbors

Query flooding

- Query message sent over existing TCP connections
- peers forward Query message
- QueryHit sent over reverse path



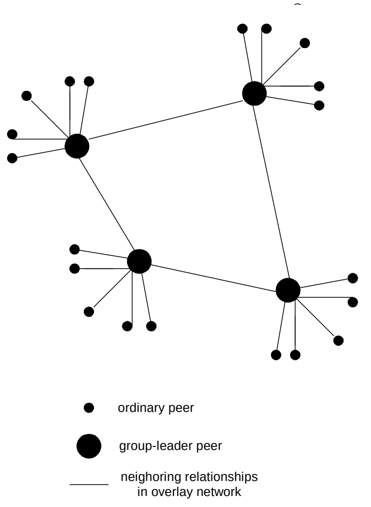
Scalability: limited scope flooding

Gnutella: Peer joining

1. joining peer Alice must find another peer in Gnutella network: use list of candidate peers
2. Alice sequentially attempts TCP connections with candidate peers until connection setup with Bob
3. *Flooding:* Alice sends Ping message to Bob; Bob forwards Ping message to his overlay neighbors (who then forward to their neighbors….)
   - peers receiving Ping message respond to Alice with Pong message
4. Alice receives many Pong messages, and can then setup additional TCP connections

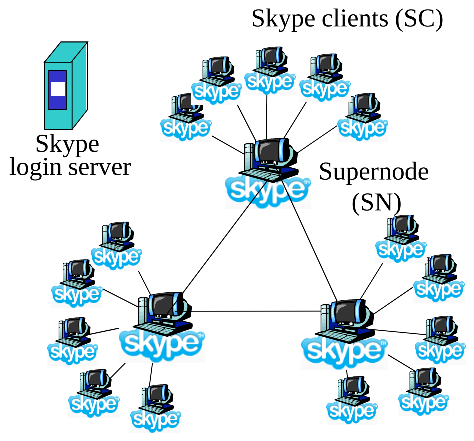
Hierarchical Overlay

- between centralized index, query flooding approaches
- each peer is either a *super node* or assigned to a super node
  - TCP connection between peer and its super node.
  - TCP connections between some pairs of super nodes.
- Super node tracks content in its children



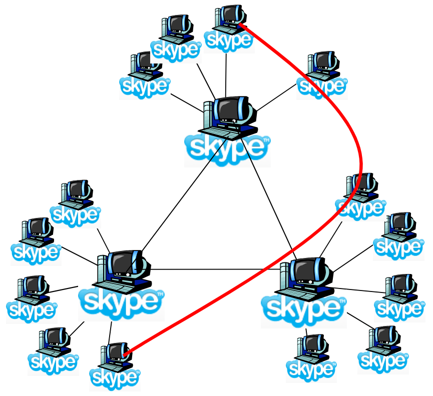
P2P Case study: Skype

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



Distributed Hash Table (DHT)

- Hash table
- DHT paradigm
- Circular DHT and overlay networks
- Peer churn

Simple Database

Simple database with(key, value) pairs:

- key: human name; value: social security #

Key	Value
John Washington	132-54-3570
Diana Louise Jones	761-55-3791
Xiaoming Liu	385-41-0902
Rakesh Gopal	441-89-1956
Linda Cohen	217-66-5609
…….	………
Lisa Kobayashi	177-23-0199

- key: movie title; value: IP address

Hash Table

- More convenient to store and search on numerical representation of key
- key = hash(original key)

Original Key	Key	Value
John Washington	8962458	132-54-3570
Diana Louise Jones	7800356	761-55-3791
Xiaoming Liu	1567109	385-41-0902
Rakesh Gopal	2360012	441-89-1956
Linda Cohen	5430938	217-66-5609
…….		………
Lisa Kobayashi	9290124	177-23-0199

Distributed Hash Table (DHT)

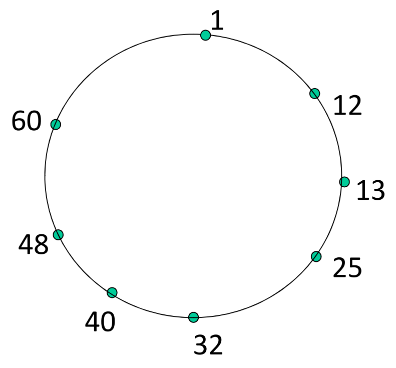
- Distribute (key, value) pairs over millions of peers
  - pairs are evenly distributed over peers
- Any peer can query database with a key
  - database returns value for the key
  - To resolve query, small number of messages exchanged among peers
- Each peer only knows about a small number of other peers
- Robust to peers coming and going (churn)

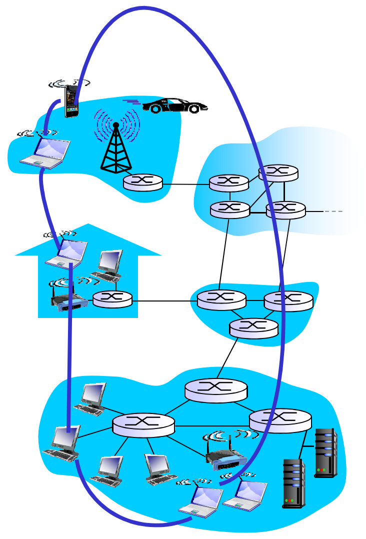
Assign key-value pairs to peers

- rule: assign key-value pair to the peer that has the *closest* ID.
- convention: closest is the *immediate successor* of the key.
- e.g., ID space {0,1,2,3,…,63}
- suppose 8 peers: 1,12,13,25,32,40,48,60
  - If key = 51, then assigned to peer 60
  - If key = 60, then assigned to peer 60
  - If key = 61, then assigned to peer 1

Circular DHT

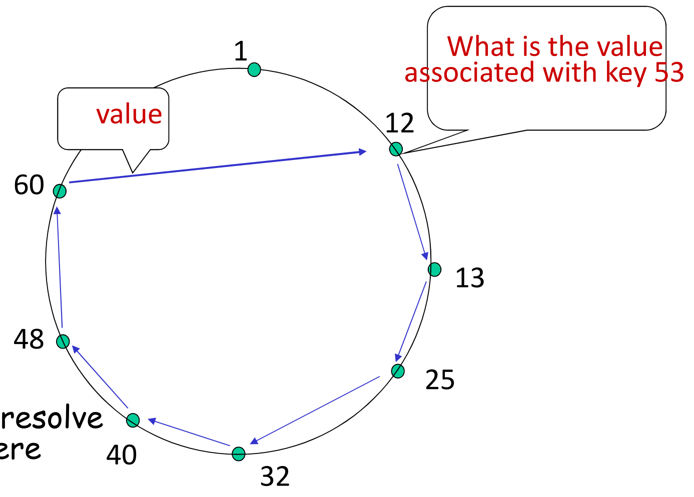
- each peer *only* aware of immediate successor and predecessor.





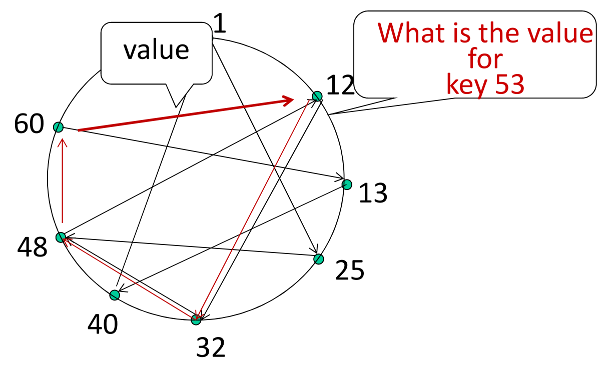
"overlay network"

Resolving a query



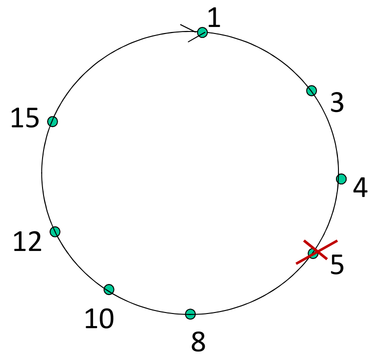
O(N) messages on avgerage to resolve query, when there are *N* peers

Circular DHT with shortcuts



- each peer keeps track of IP addresses of predecessor, successor, short cuts.
- reduced from 6 to 3 messages.
- possible to design shortcuts with *O(log N)* neighbors, *O(log N)* messages in query

Peer churn

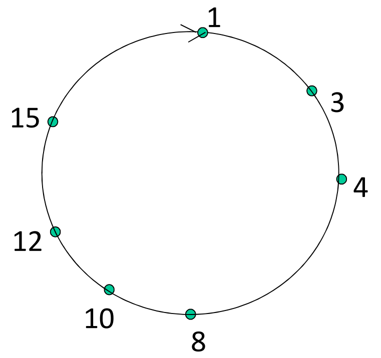


handling peer churn:

- peers may come and go (churn)
- each peer knows address of its two successors
- each peer periodically pings its two successors to check aliveness
- if immediate successor leaves, choose next successor as new immediate successor

example: peer 5 abruptly leaves

Peer churn



handling peer churn:

- peers may come and go (churn)
- each peer knows address of its two successors
- each peer periodically pings its two successors to check aliveness
- if immediate successor leaves, choose next successor as new immediate successor

example: peer 5 abruptly leaves

- peer 4 detects peer 5's departure; makes 8 its immediate successor
- 4 asks 8 who its immediate successor is; makes 8's immediate successor its second successor.