

DNS: Domain Name System

People: many identifiers:

- SSN, name, Passport #

Internet hosts, routers:

- IP address (32 bit) - used for addressing datagrams
- "name", e.g., gaia.cs.umass.edu - used by humans

Q: map between IP addresses and name ?

DNS: Domain Name System

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

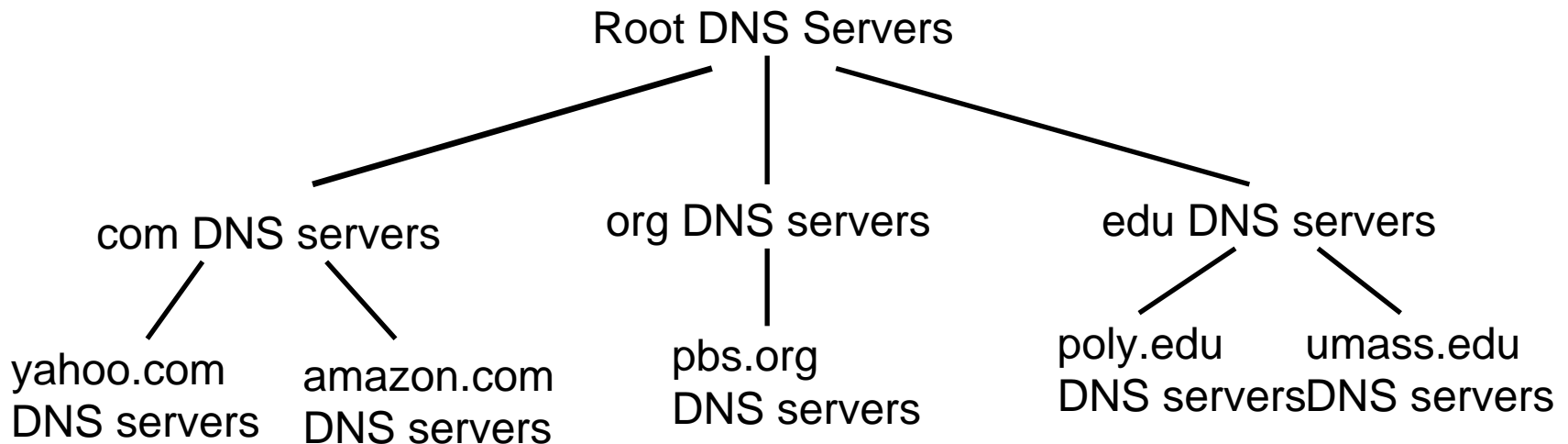
- ❑ Hostname to IP address translation
- ❑ Host aliasing
 - Canonical and alias names
- ❑ Mail server aliasing
- ❑ Load distribution
 - Replicated Web servers: set of IP addresses for one canonical name

Why not centralize DNS?

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

doesn't *scale!*

Distributed, Hierarchical Database

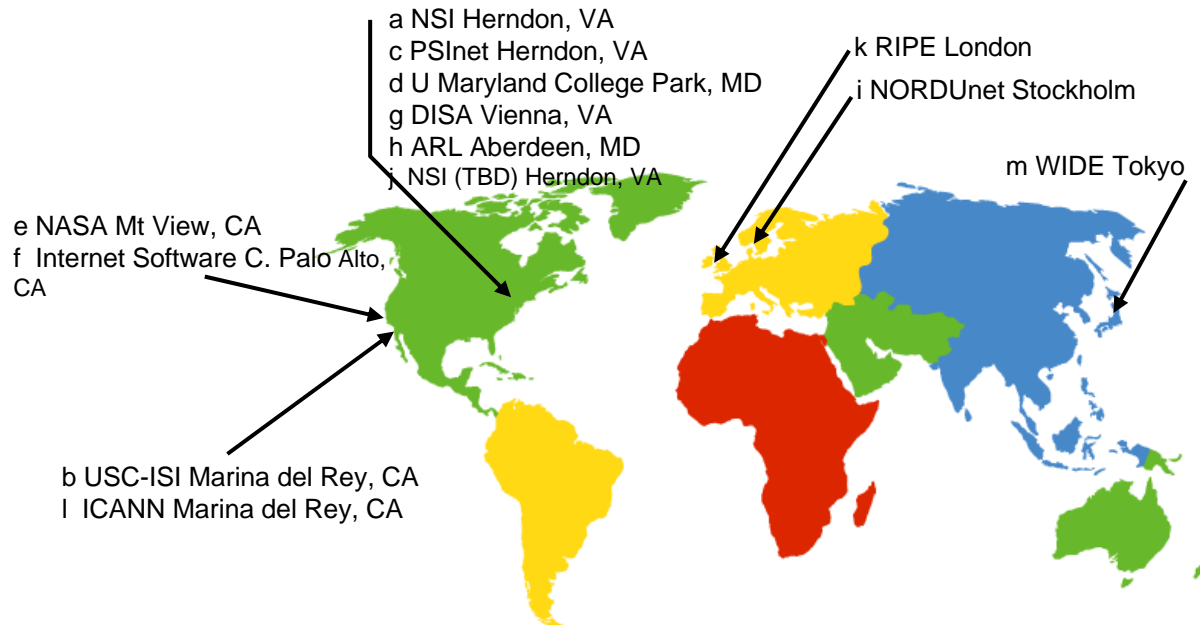


Client wants IP for www.amazon.com; 1st approx:

- ❑ 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
- ❑ 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

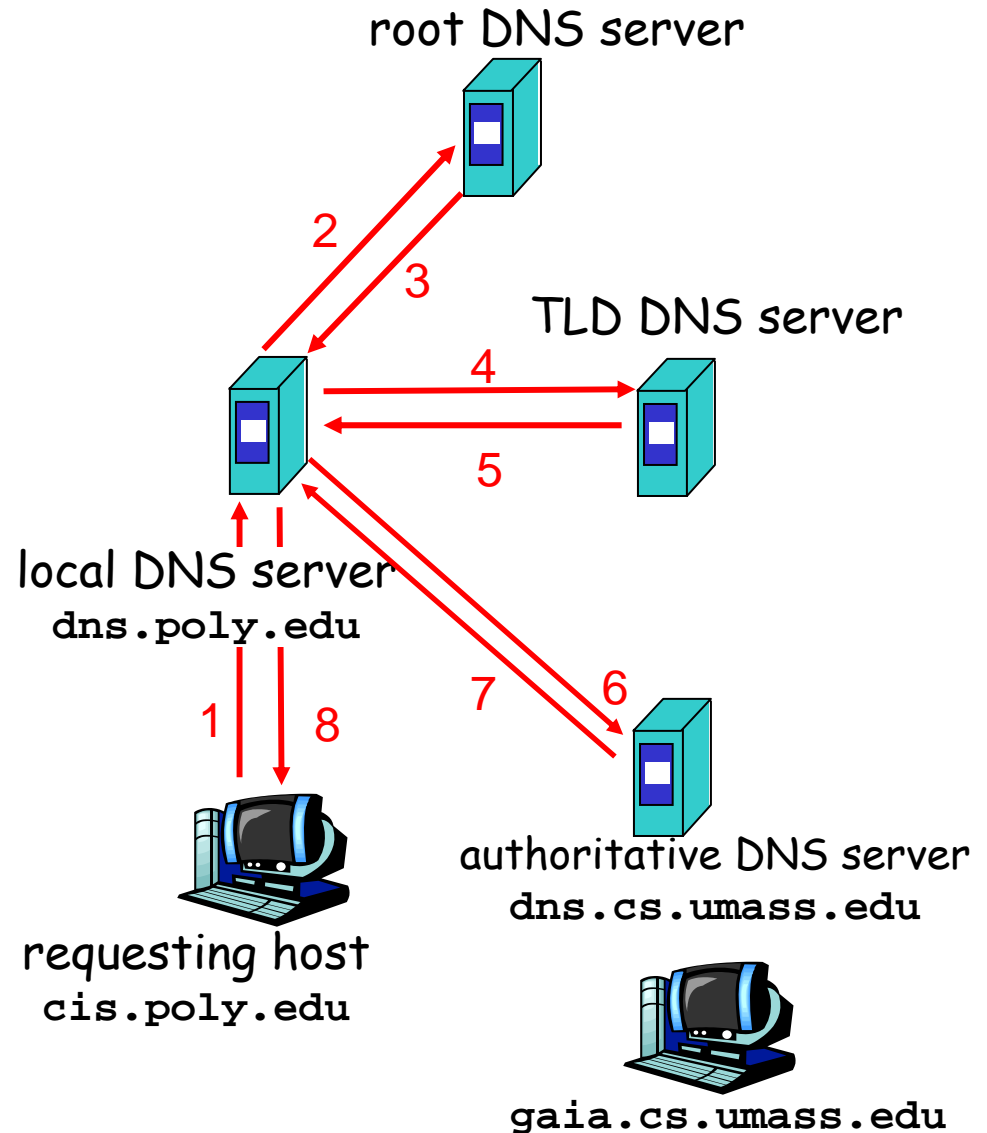
- ❑ **Top-level domain (TLD) servers:** responsible for com, org, net, edu, etc, and all top-level country domains uk, fr, ca, jp.
 - Network solutions maintains servers for com TLD
 - Educause for edu TLD
- ❑ **Authoritative DNS servers:** organization's DNS servers, providing authoritative hostname to IP mappings for organization's servers (e.g., Web and 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 a host makes a DNS query, query is sent to its local DNS server
 - Acts as a proxy, forwards query into hierarchy.

Example

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



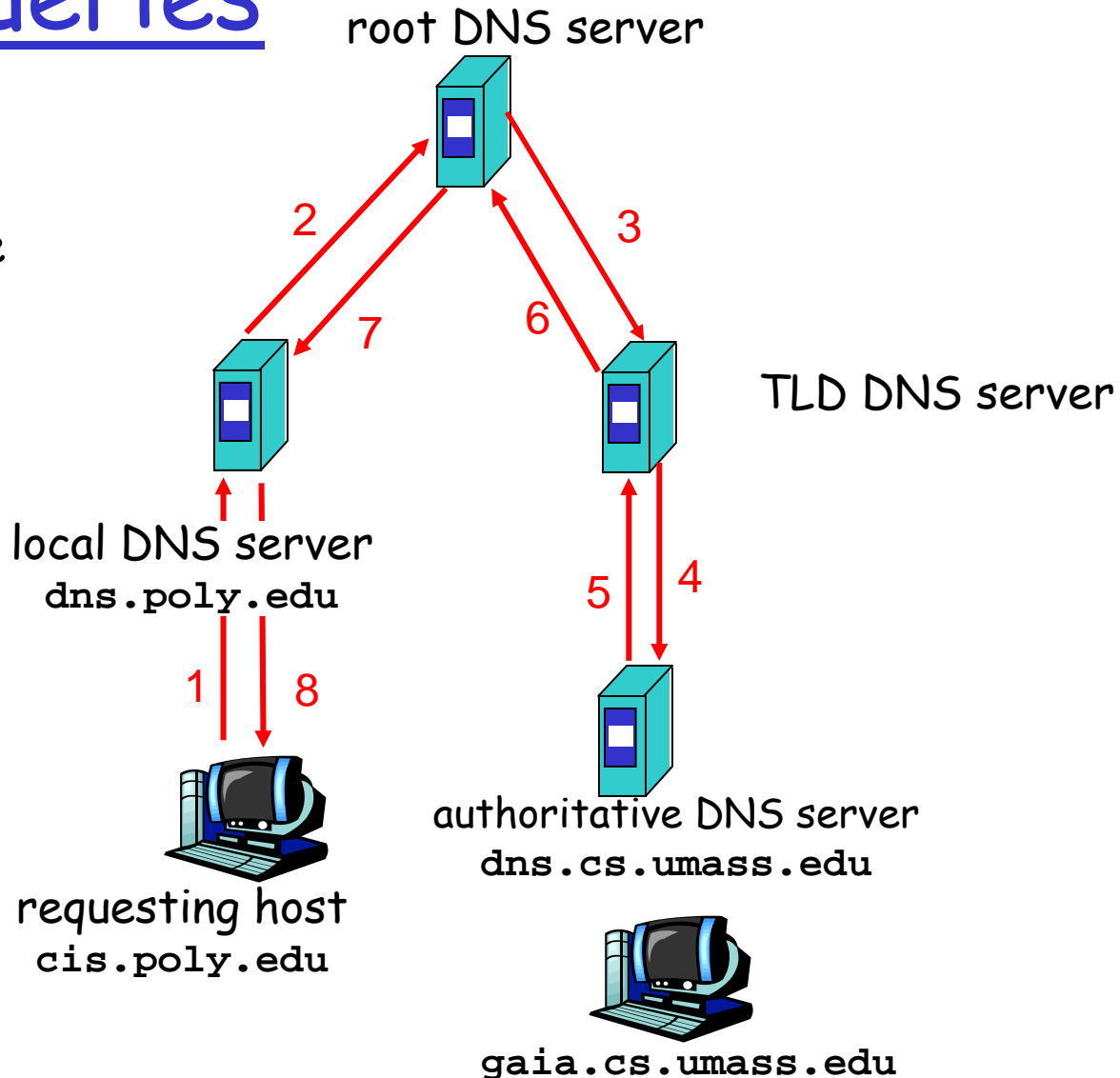
Recursive queries

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
 - TLD servers typically cached in local name servers
 - Thus root name servers not often visited
- 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 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

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

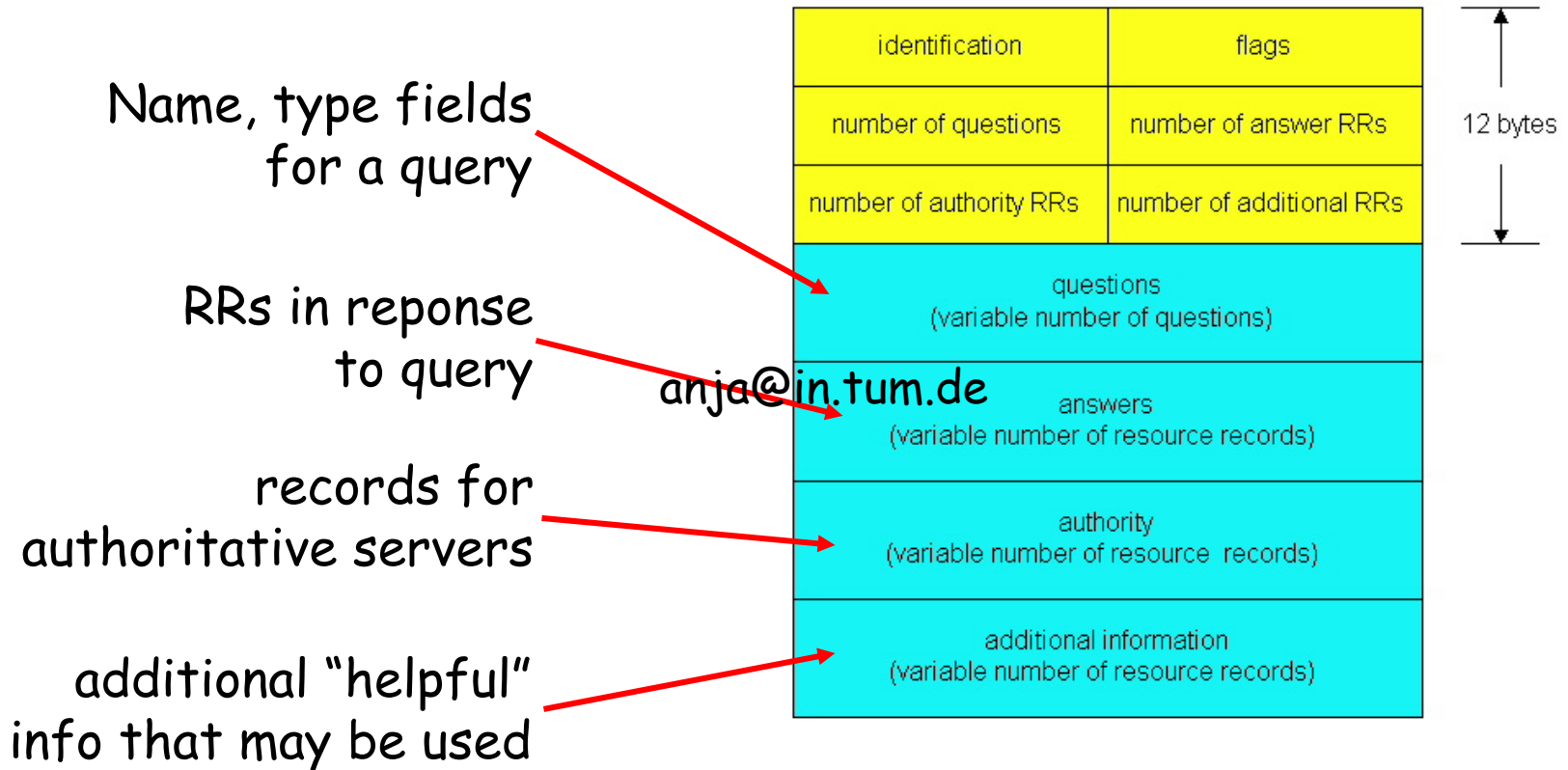
msg header

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

identification	flags
number of questions	number of answer RRs
number of authority RRs	number of additional RRs
questions (variable number of questions)	
answers (variable number of resource records)	
authority (variable number of resource records)	
additional information (variable number of resource records)	



DNS protocol, messages



Inserting records into DNS

- ❑ Example: just created startup "Network Utopia"
- ❑ Register name networkutopia.com at a registrar (e.g., Network Solutions)
 - Need to provide registrar with names and IP addresses of your authoritative name server (primary and secondary)
 - Registrar inserts two RRs into the com TLD server:


```
(networkutopia.com, dns1.networkutopia.com, NS)  
(dns1.networkutopia.com, 212.212.212.1, A)
```
- ❑ Put in authoritative server Type A record for www.networkutopia.com and Type MX record for networkutopia.com
- ❑ How do people get the IP address of your Web site?

Let's look at Applications in action

- Tcpdump examples

HTTP revisited: Uploading form input

Post method:

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

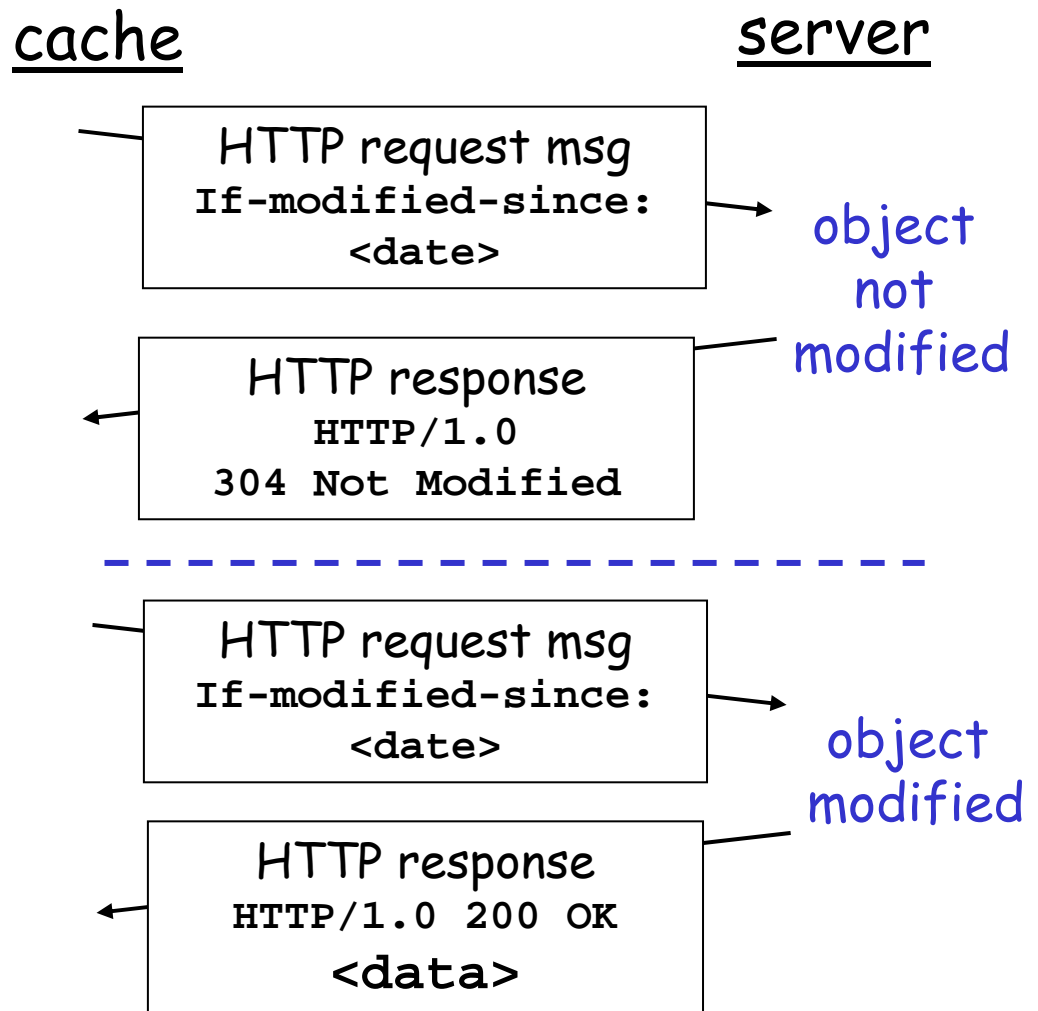
URL method:

- ❑ Uses GET method
- ❑ Input is uploaded in URL field of request line:

`www.somesite.com/animalsearch?monkeys&banana`

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



User-server state: cookies

Many major Web sites use cookies

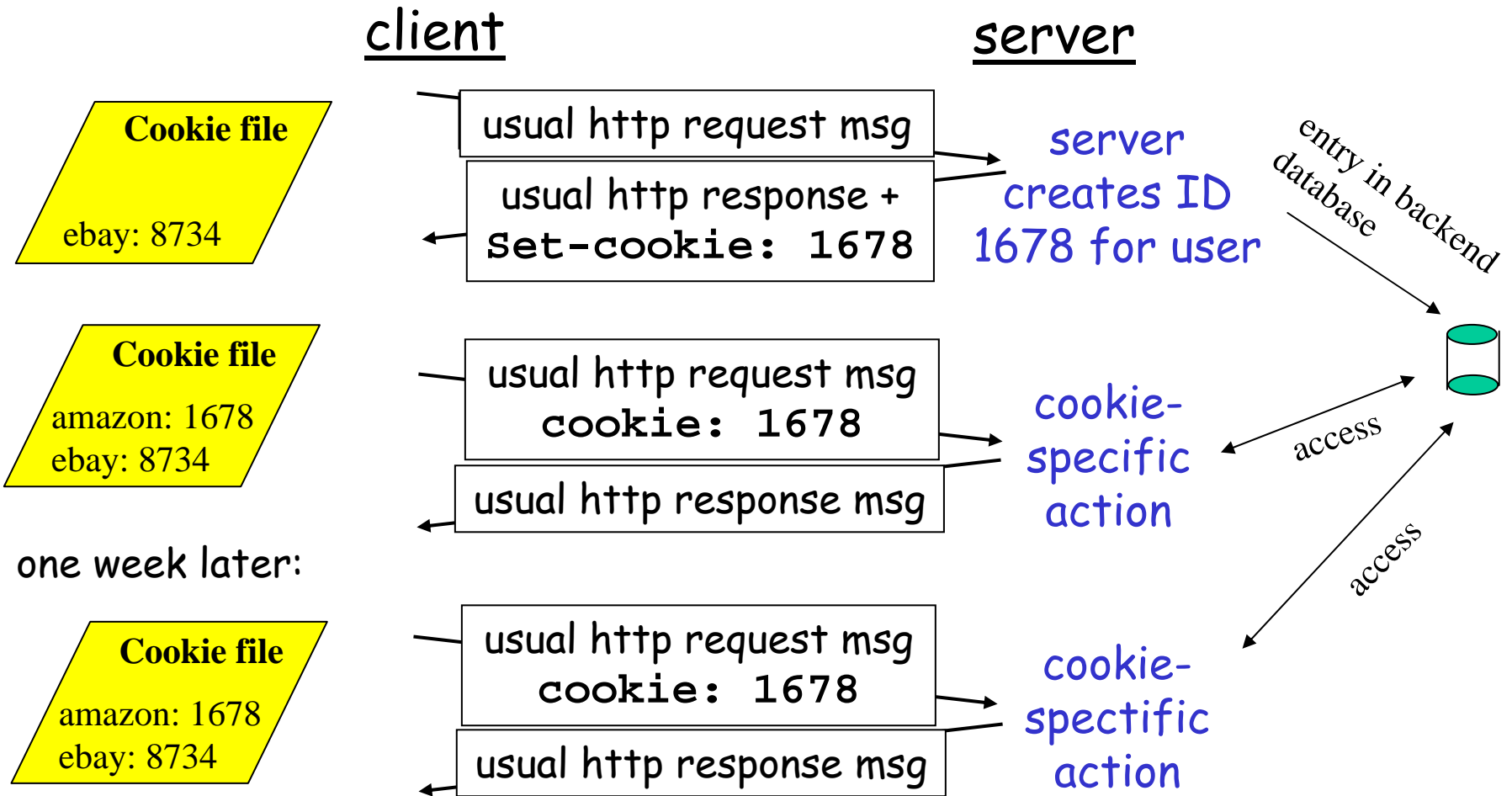
Four components:

- 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 access Internet always from same PC
- She visits a specific e-commerce site for first time
- When initial HTTP requests arrives at site, site creates a unique ID and creates an 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)

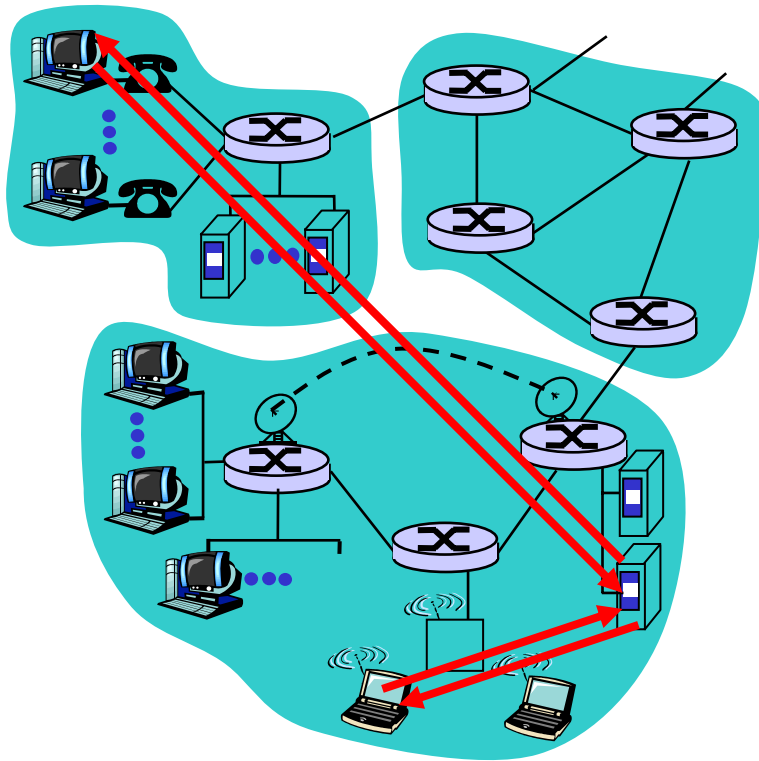
Cookies and privacy: aside

- ❑ cookies permit sites to learn a lot about you
- ❑ you may supply name and e-mail to sites
- ❑ search engines use redirection & cookies to learn yet more
- ❑ advertising companies obtain info across sites

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

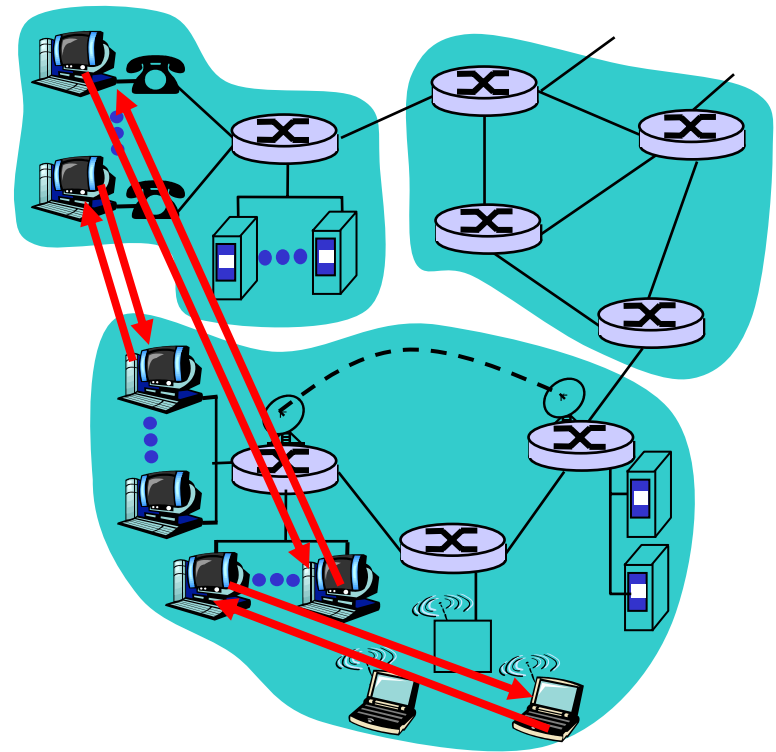
clients:

- 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
- ❑ example: Gnutella

Highly scalable but
difficult to manage



Hybrid of client-server and P2P

Skype

- Internet telephony app
- Finding address of remote party: centralized server(s)
- Client-client connection is direct (not through server)

Instant messaging

- Chatting between two users is P2P
- Presence detection/location centralized:
 - User registers its IP address with central server when it comes online
 - User contacts central server to find IP addresses of buddies

P2P file sharing

Example

- ❑ Alice runs P2P client application on her notebook computer
 - ❑ Intermittently connects to Internet; gets new IP address for each connection
 - ❑ Asks for "Hey Jude"
 - ❑ Application displays other peers that have copy of Hey Jude.
 - ❑ Alice chooses one of the peers, Bob.
 - ❑ File is copied from Bob's PC to Alice's notebook: HTTP
 - ❑ While Alice downloads, other users uploading from Alice.
 - ❑ Alice's peer is both a Web client and a transient Web server.
- All peers are servers = highly scalable!

P2P: centralized directory

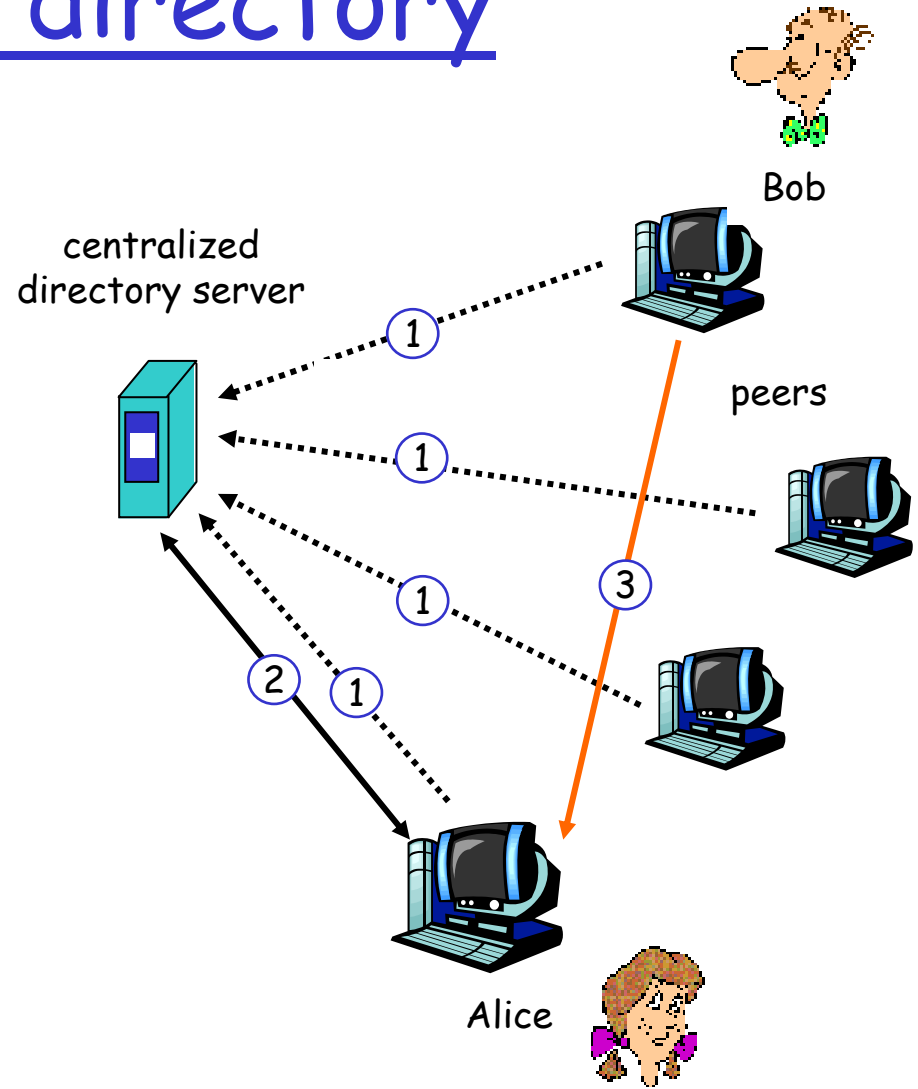
original "Napster" design

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

file transfer is decentralized, but locating content is highly centralized

Query flooding: Gnutella

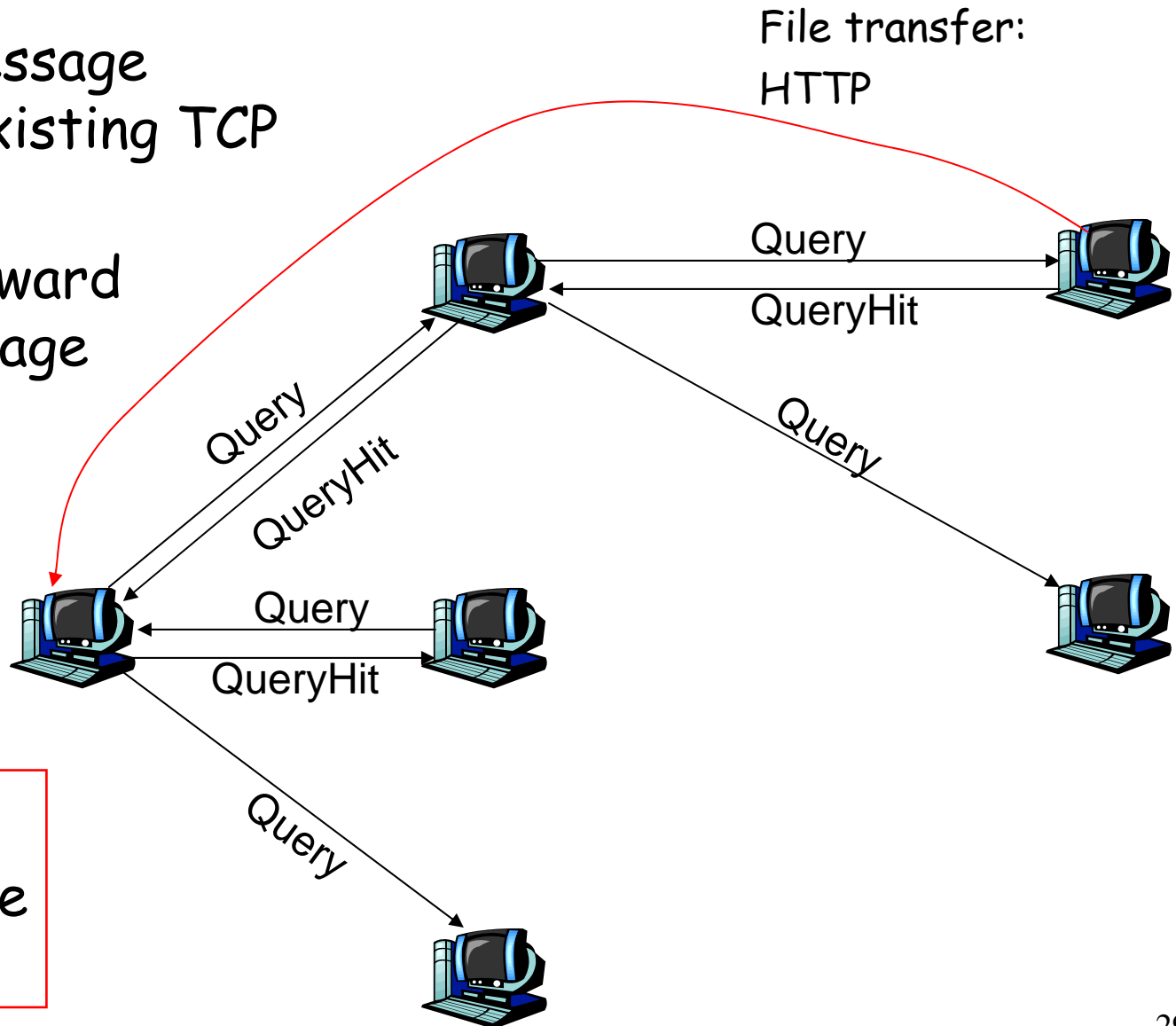
- ❑ fully distributed
 - no central server
- ❑ public domain protocol
- ❑ many Gnutella clients implementing protocol

overlay network: graph

- ❑ edge between peer X and Y if there's a TCP connection
- ❑ all active peers and edges are part of the overlay net
- ❑ edge is not a physical link
- ❑ given peer will typically be connected with < 10 overlay neighbors

Gnutella: protocol

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



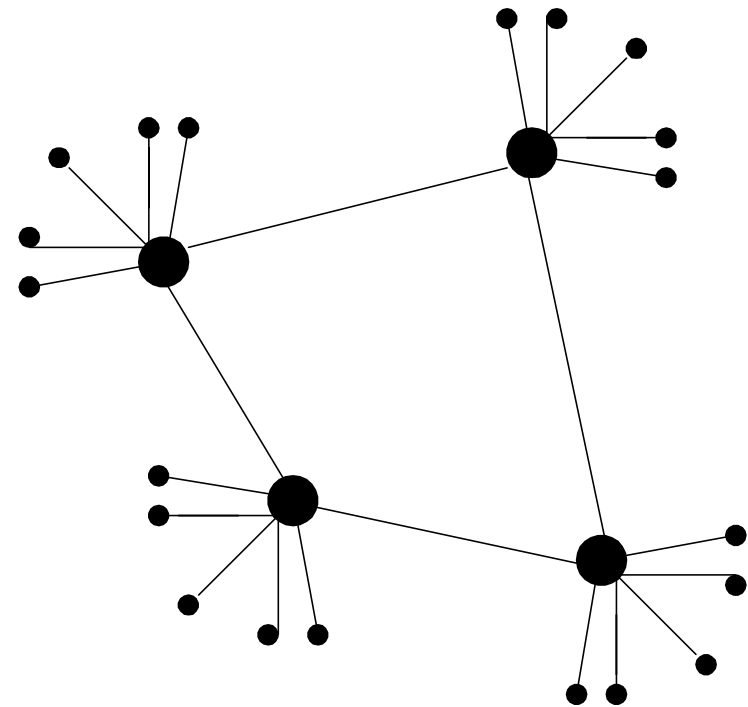
Scalability:
limited scope
flooding

Gnutella: Peer joining

1. Joining peer X must find some other peer in Gnutella network: use list of candidate peers
2. X sequentially attempts to make TCP with peers on list until connection setup with Y
3. X sends Ping message to Y; Y forwards Ping message.
4. All peers receiving Ping message respond with Pong message
5. X receives many Pong messages. It can then setup additional TCP connections

Exploiting heterogeneity: KaZaA

- Each peer is either a group leader or assigned to a group leader.
 - TCP connection between peer and its group leader.
 - TCP connections between some pairs of group leaders.
- Group leader tracks the content in all its children.



● ordinary peer

● group-leader peer

— neighboring relationships
in overlay network

KaZaA: Querying

- ❑ Each file has a hash and a descriptor
- ❑ Client sends keyword query to its group leader
- ❑ Group leader responds with matches:
 - For each match: metadata, hash, IP address
- ❑ If group leader forwards query to other group leaders, they respond with matches
- ❑ Client then selects files for downloading
 - HTTP requests using hash as identifier sent to peers holding desired file

KaZaA tricks

- ❑ Limitations on simultaneous uploads
- ❑ Request queuing
- ❑ Incentive priorities
- ❑ Parallel downloading

For more info:

- ❑ J. Liang, R. Kumar, K. Ross, "Understanding KaZaA,"
(available via cis.poly.edu/~ross)

Application Layer: Summary

Our study of network apps now complete!

- application architectures
- application service requirements:
 - reliability, bandwidth, delay
- client-server paradigm
- Internet transport service model
 - connection-oriented, reliable: TCP
 - unreliable, datagrams: UDP
- specific protocols:
 - http
 - ftp
 - smtp, pop3
 - dns
 - gnutella
- socket programming
 - client/server implementation
 - using tcp, udp sockets

Application-Layer: 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
- ❑ control vs. data msgs
 - in-band, out-of-band
- ❑ centralized vs. decentralized
- ❑ stateless vs. stateful
- ❑ reliable vs. unreliable msg transfer
- ❑ "complexity at network edge"