CS450 Computer Networks

The slides used in class are derived from the slides available on our text book companion website:
http://wps.pearsoned.com/ecs_kurose_compnetw_6/
copyright 1996-2012 J.F Kurose and K.W. Ross

CS450 Computer Networks

Lesson 6

Application Layer – Peer-to-Peer and DNS

The nature of life is to grow

CS450 - Lesson 6: Application layer –Peer-to-Peer

Our goal:

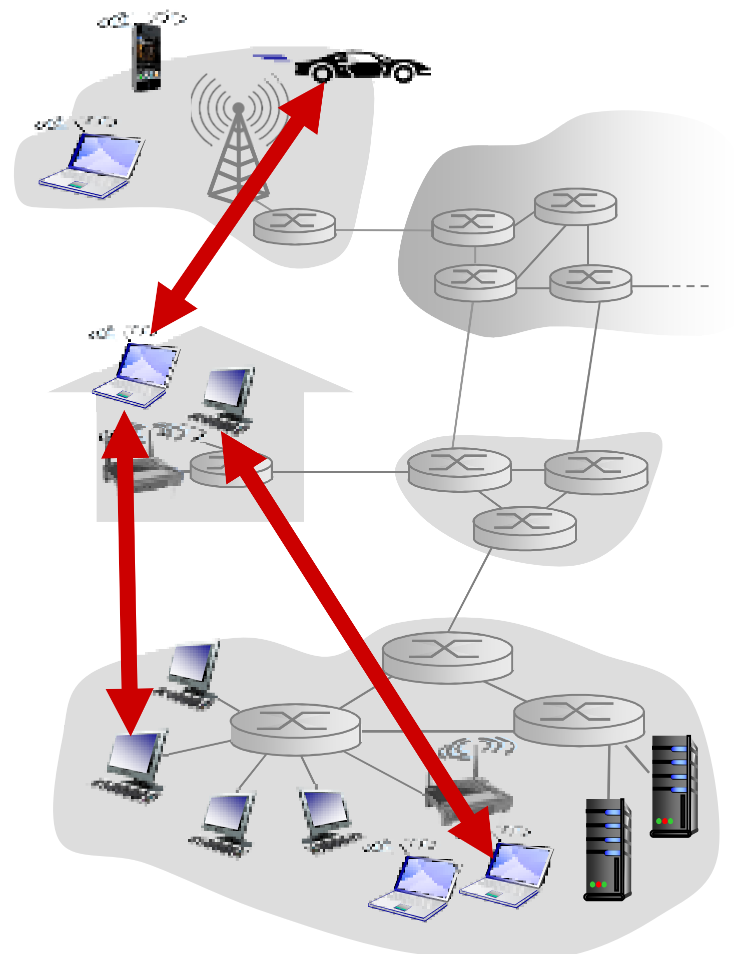
- Understand the conceptual and implementation aspects of network application protocols using the peer-to-peer paradigm

Pure P2P architecture

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

examples:

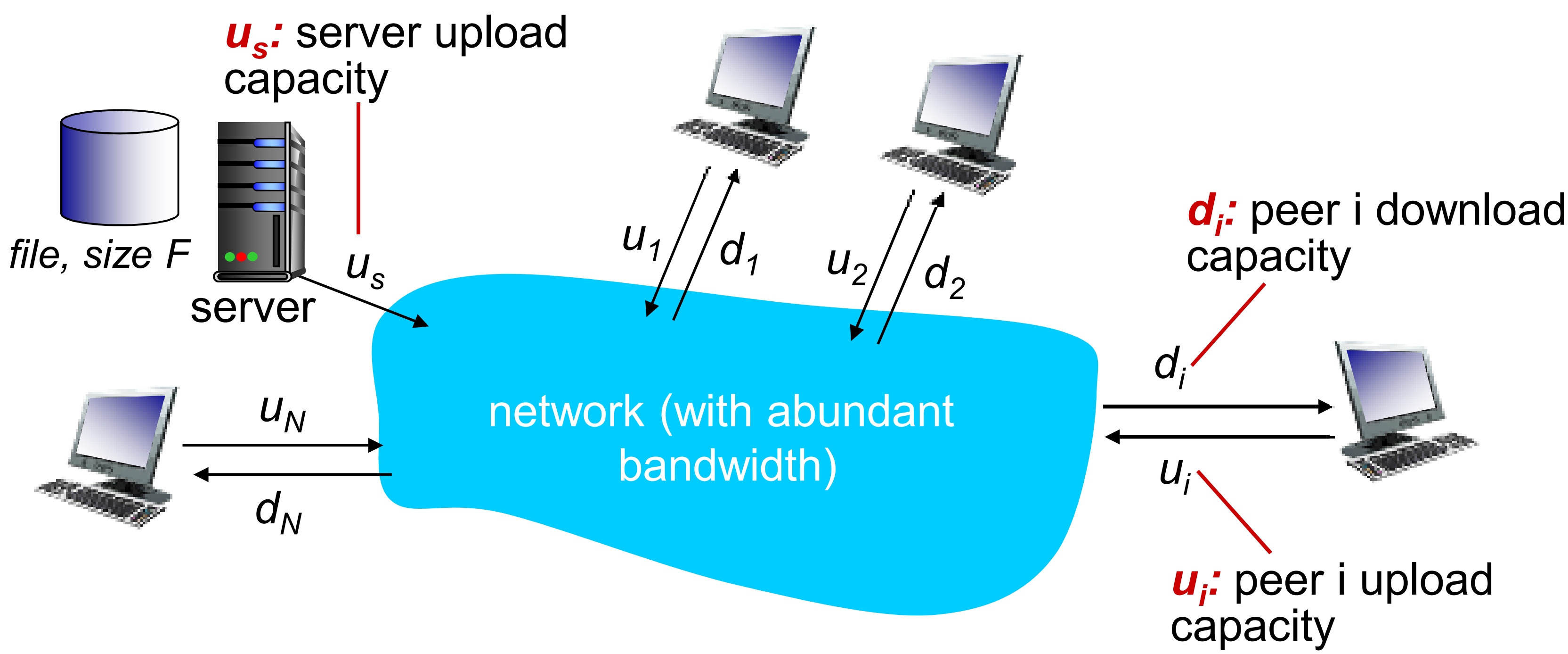
- file distribution (BitTorrent)
- Streaming (KanKan)
- VoIP (Skype)



File distribution: client-server vs P2P

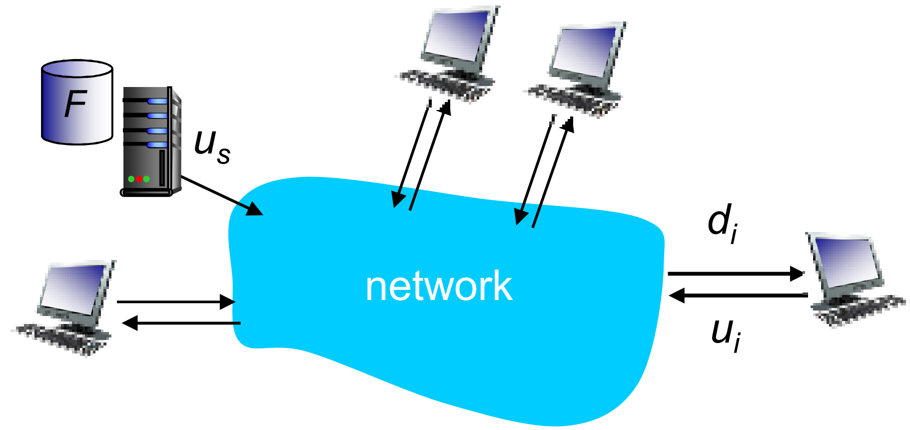
Question: how much time to distribute file (size *F*) from one server to *N* *peers*?

- peer upload/download capacity is limited resource



File distribution time: client-server

- *server transmission:* must sequentially send (upload) *N* file copies:
  - 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}

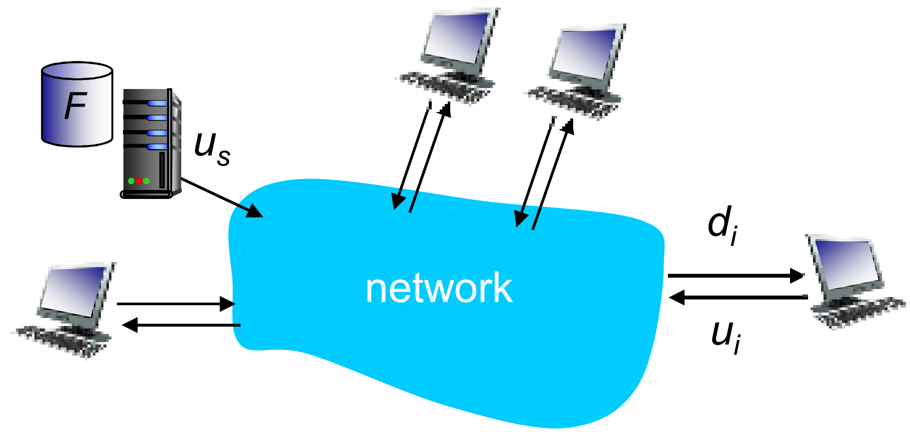
time to distribute F to N clients using client-server approach

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

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 (limting max download rate) is $u_s + \Sigma u_i$

time to distribute F to N clients using P2P approach

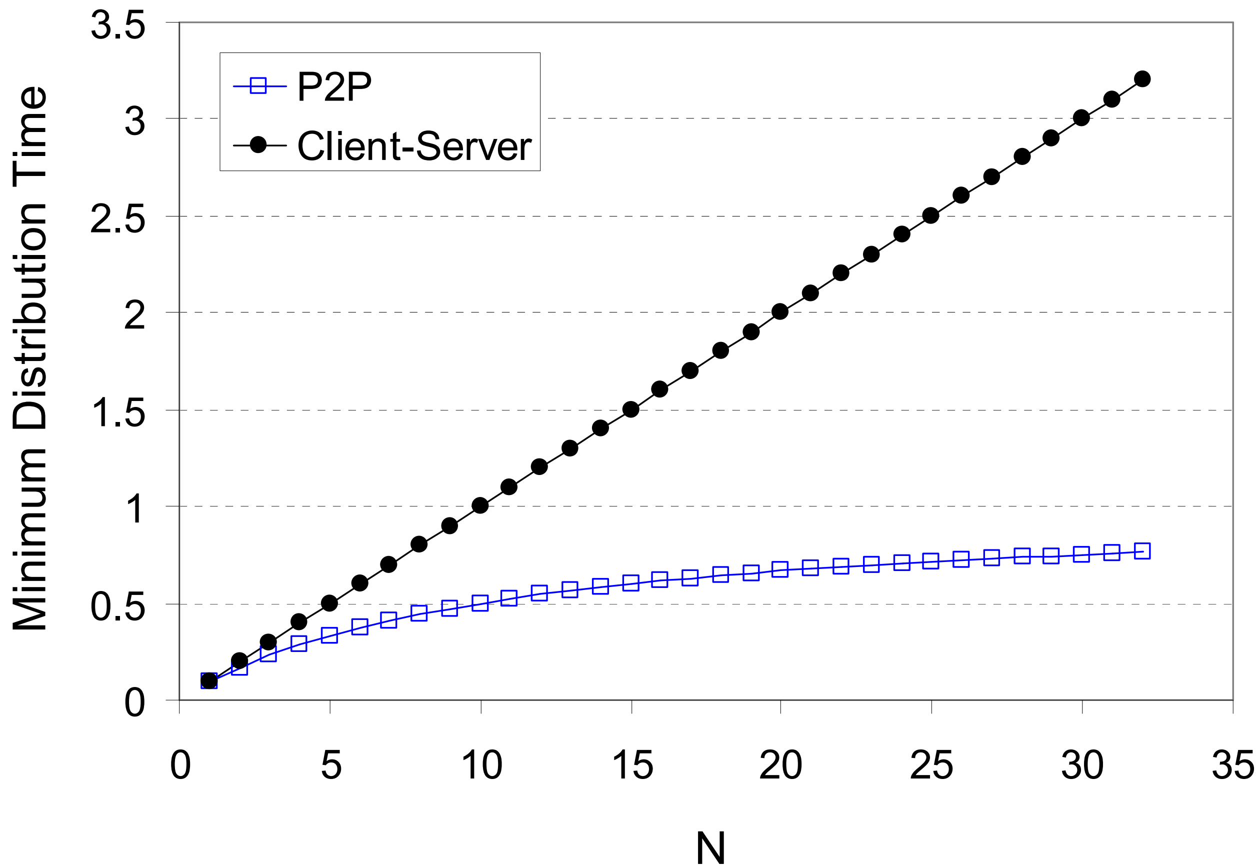
$$D_{P2P} \geq \max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$$

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, $u_s = 10u$, $d_{min} \geq u_s$



P2P Architecture Issues

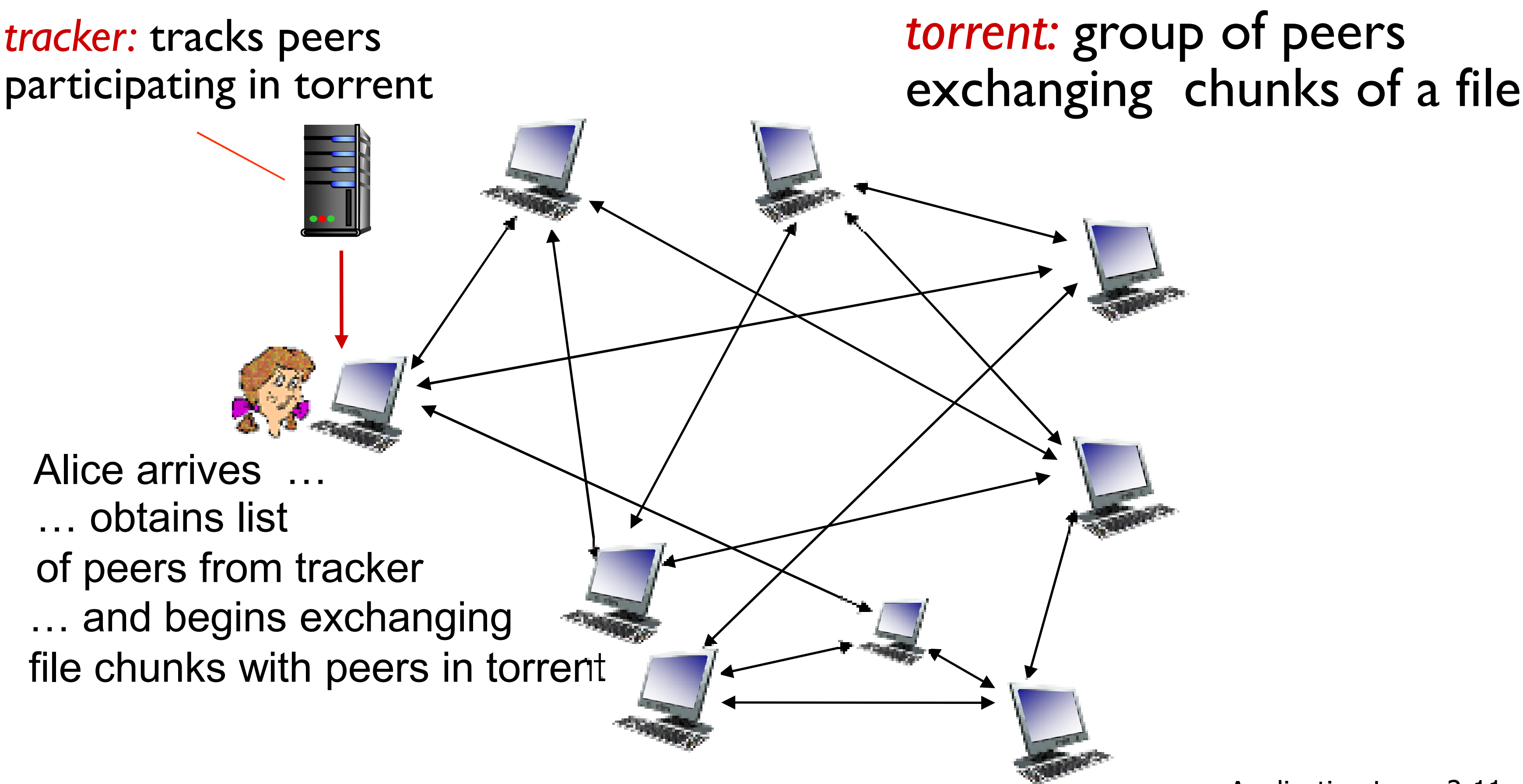
- The price for the faster file distribution of P2P over client-server is added complexity.
- What are some of the issues a successful P2P architecture must solve?

P2P Architecture Issues

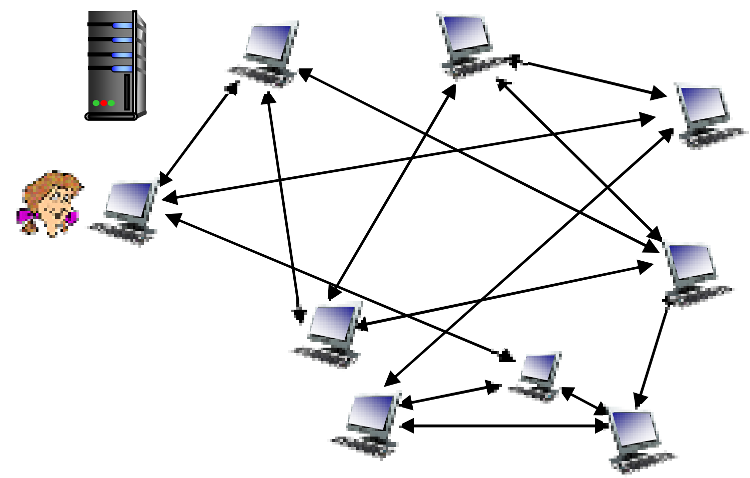
- A successful P2P architecture must solve:
  - How to create a peer group working to download a file
  - How to track peers as they join or leave the peer group
  - How to figure out which peers have the part of the file that you need
  - How to optimize the peers you need for your file
  - How to reward peers that share their files and discourage freeloaders
- Let's look at how BitTorrent solves these problems

P2P file distribution: BitTorrent

- file divided into 256Kb chunks
- peers in torrent send/receive file chunks



P2P file distribution: BitTorrent



- peer joining torrent:
  - has no chunks, but will accumulate them over time from other peers
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")
- while downloading, peer uploads chunks to other peers
- peer may change peers with whom it exchanges chunks
- *churn:* peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent

BitTorrent: requesting, sending file chunks

requesting chunks:

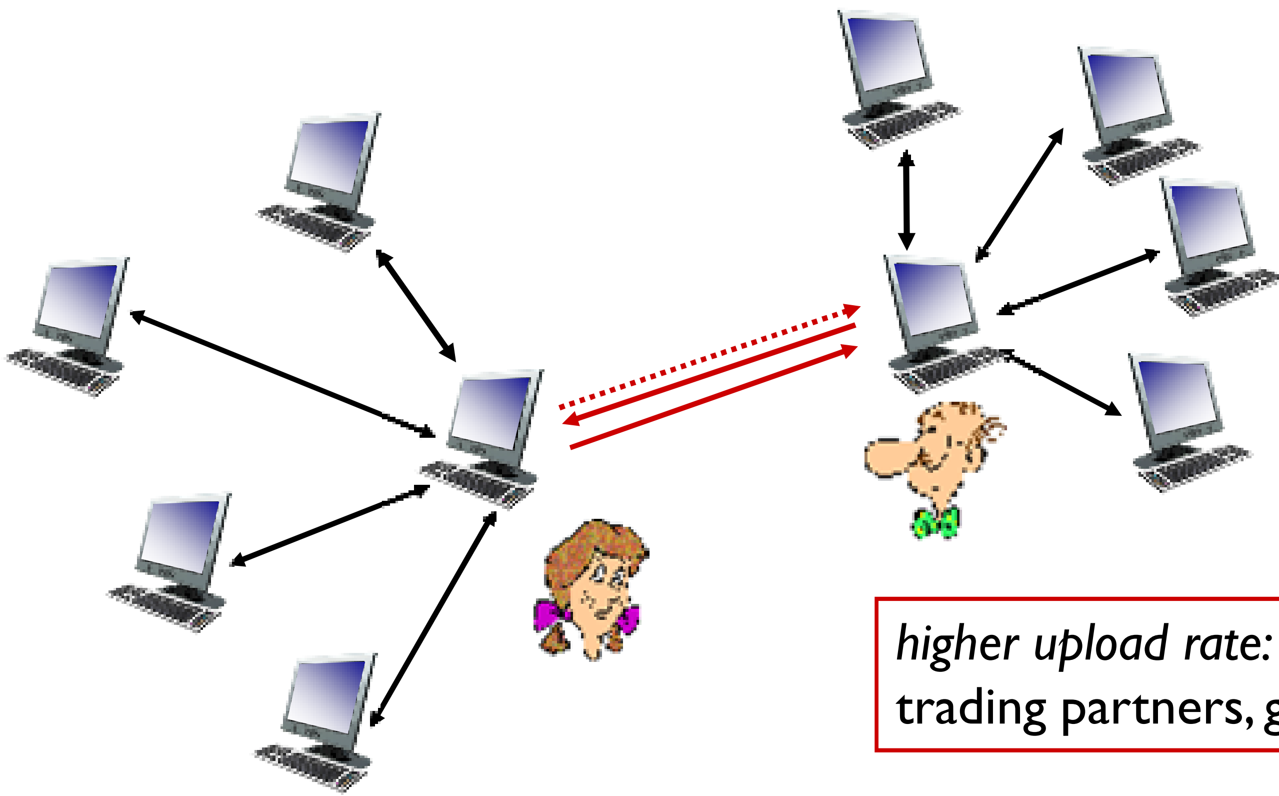
- at any given time, different peers have different subsets of file chunks
- periodically, Alice asks each peer for list of chunks that they have
- Alice requests missing chunks from peers, rarest first

sending chunks: incent higher sharing

- Alice sends chunks to those four peers currently sending her chunks *at highest rate*
  - other peers are choked by Alice (do not receive chunks from her)
  - re-evaluate top 4 every10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - "optimistically unchoke" this peer
  - newly chosen peer may join top 4

BitTorrent: incentive scheme

(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



higher upload rate: find better trading partners, get file faster !

Distributing a Database using the P2P Paradigm

- A distributed hash table.
- Database has (key, value) pairs; examples:
  - key: ss number; value: human name
  - key: movie title; value: IP address
- Distribute the (key, value) pairs over all peers (perhaps millions of peers)

Distributed Hash Table (DHT)

- All Peers can:

1) query DHT with any key
   - DHT returns values that match the key

2) insert (key, value) pairs

Q: how to assign keys to peers?

- central issue:
  - assigning (key, value) pairs to peers.
- basic idea:
  - convert each key to an integer
  - Assign integer to each peer
  - put (key,value) pair in the peer that is closest to the key

DHT identifiers

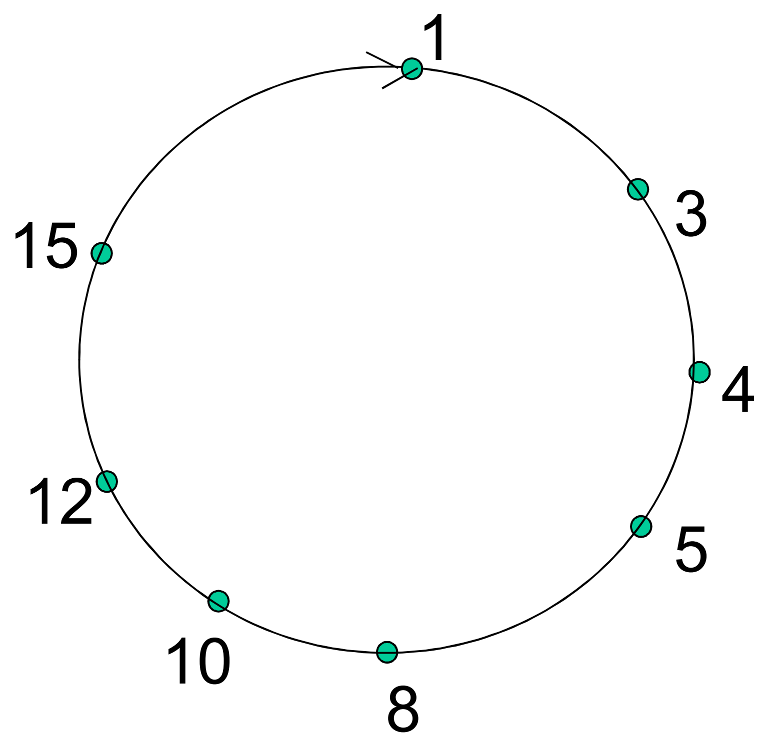
- assign integer identifier to each peer in range $[0,2^n-1]$ for some n.
  - each identifier represented by n bits.

- require each key to be an integer in same range
- to get integer key, hash original key
  - e.g., key = hash("Led Zeppelin IV")
  - this is why its is referred to as a *distributed "hash" table*

Assign keys to peers

- rule: assign key to the peer that has the *closest* ID.
- convention in lecture: closest is the *immediate successor* of the key.
- e.g., n=4; peers: 1,3,4,5,8,10,12,14;
  - key = 13, then successor peer = 14
  - key = 15, then successor peer = 1

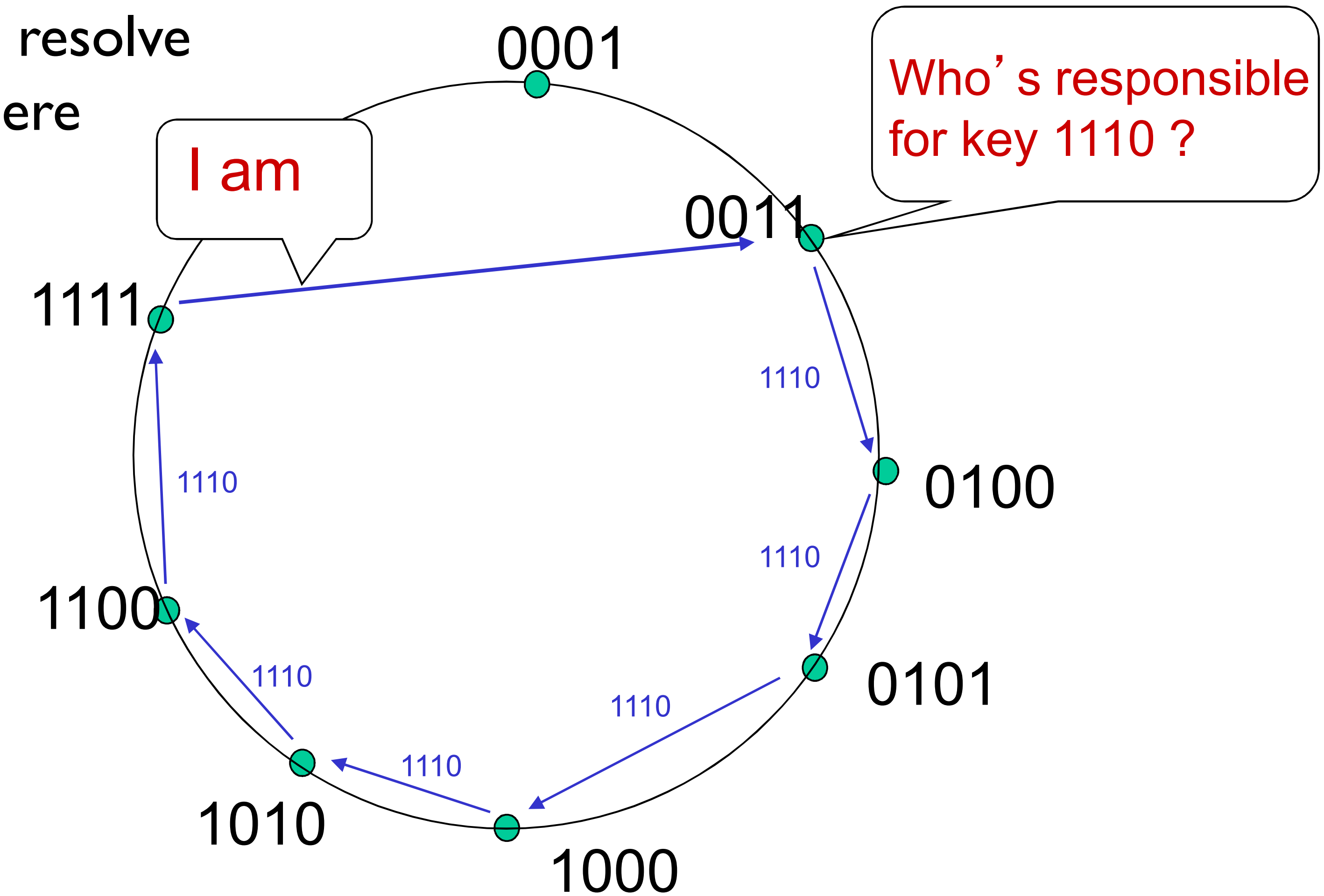
Circular DHT (1)



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

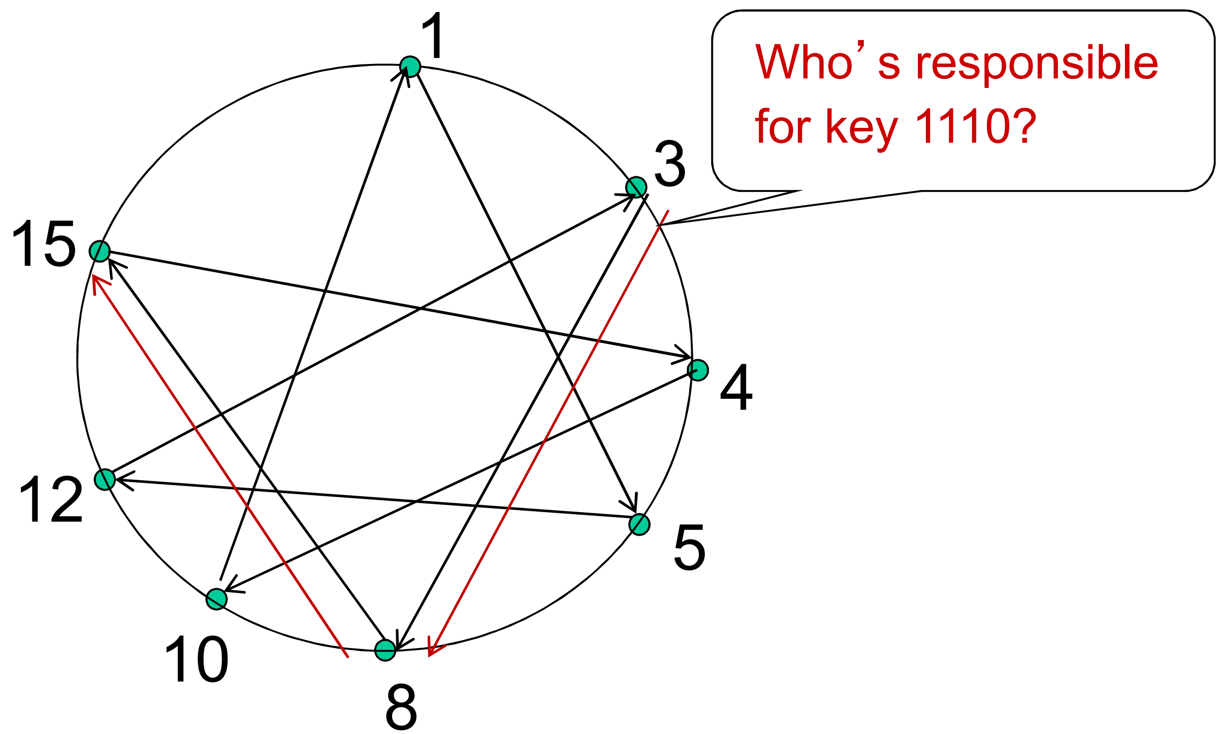
Circular DHT (1)

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



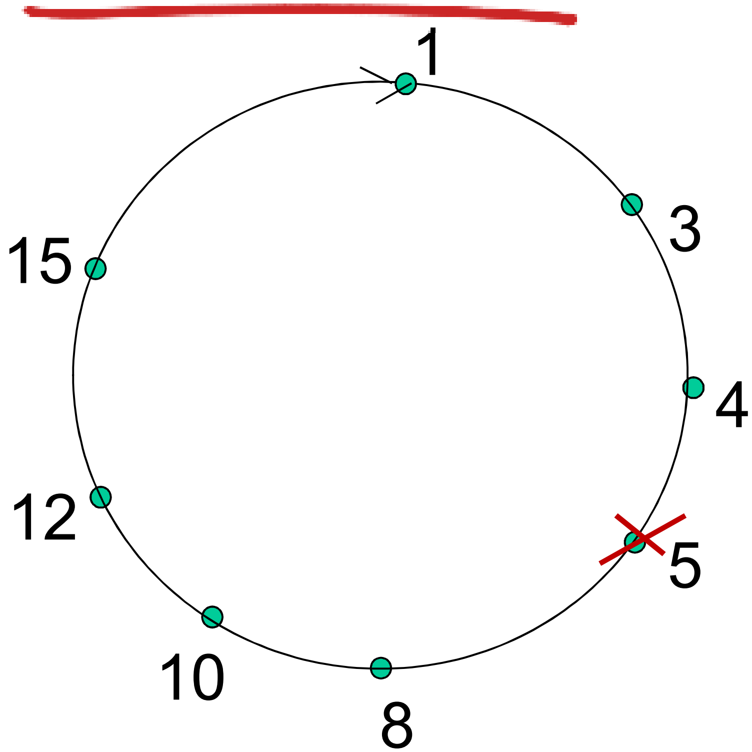
Define closest
as closest
successor

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



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 departure; makes 8 its immediate successor; asks 8 who its immediate successor is; makes 8's immediate successor its second successor.
- what if peer 13 wants to join?

Lesson 6: DNS

- Our goal: complete our study of application layer protocols by reviewing DNS. An example of core intelligence of the internet distributed throughout the network.
  - DNS – distributed control, scaleable

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: map between IP address and name, and vice versa ?

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:
    - relay1.west-coast.hotmail.com
  - alias names
    - hotmail.com
- mail server aliasing
- load distribution
  - replicated Web servers: set of IP addresses for one canonical name

Why not centralize DNS?

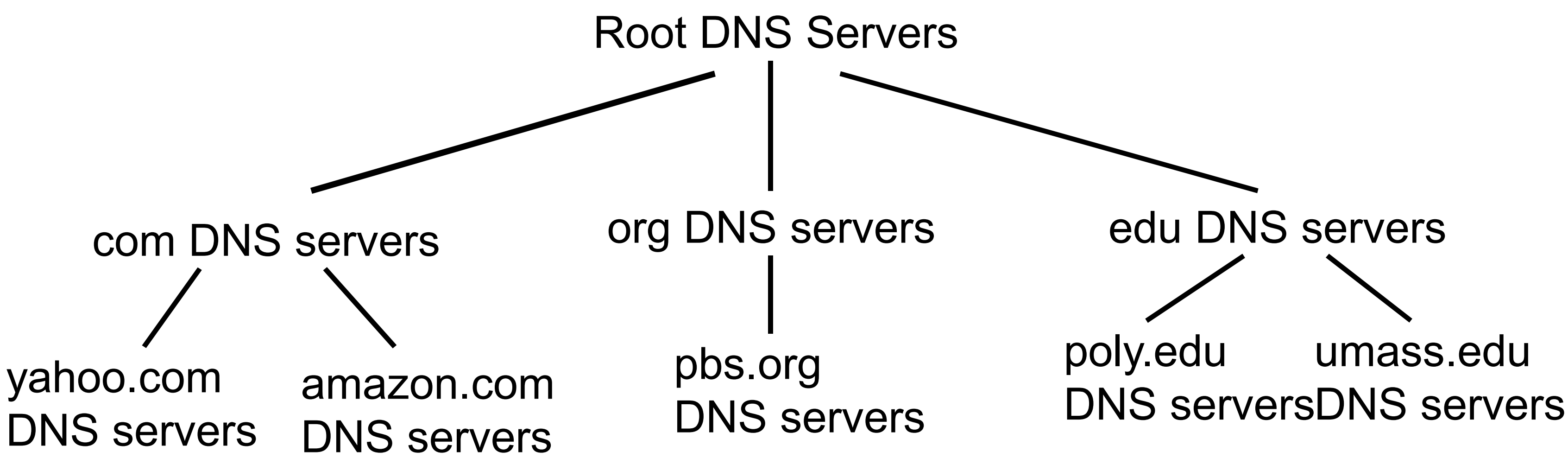
DNS

Why not centralize DNS?

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

doesn't *scale!*

Distributed, Hierarchical Database

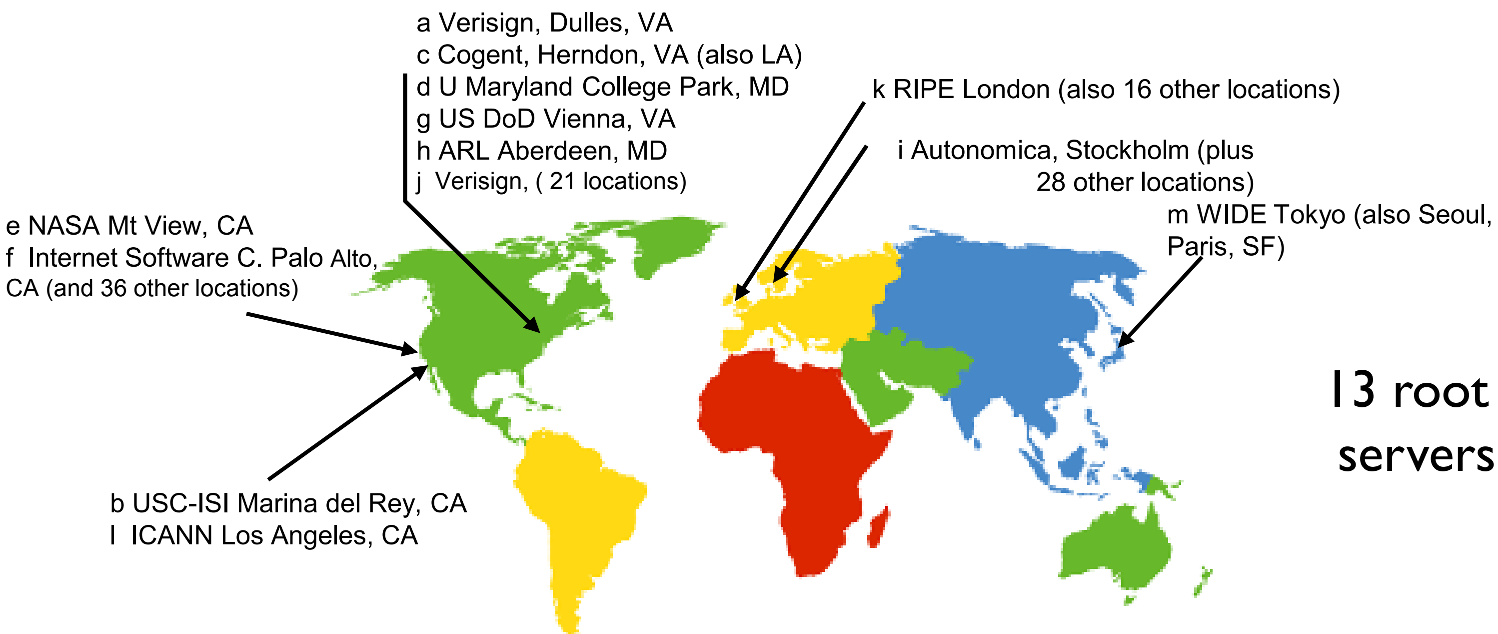


client wants IP for www.amazon.com; 1st approx:

- 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, aero, jobs, museums, and all top-level country domains, e.g.: 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

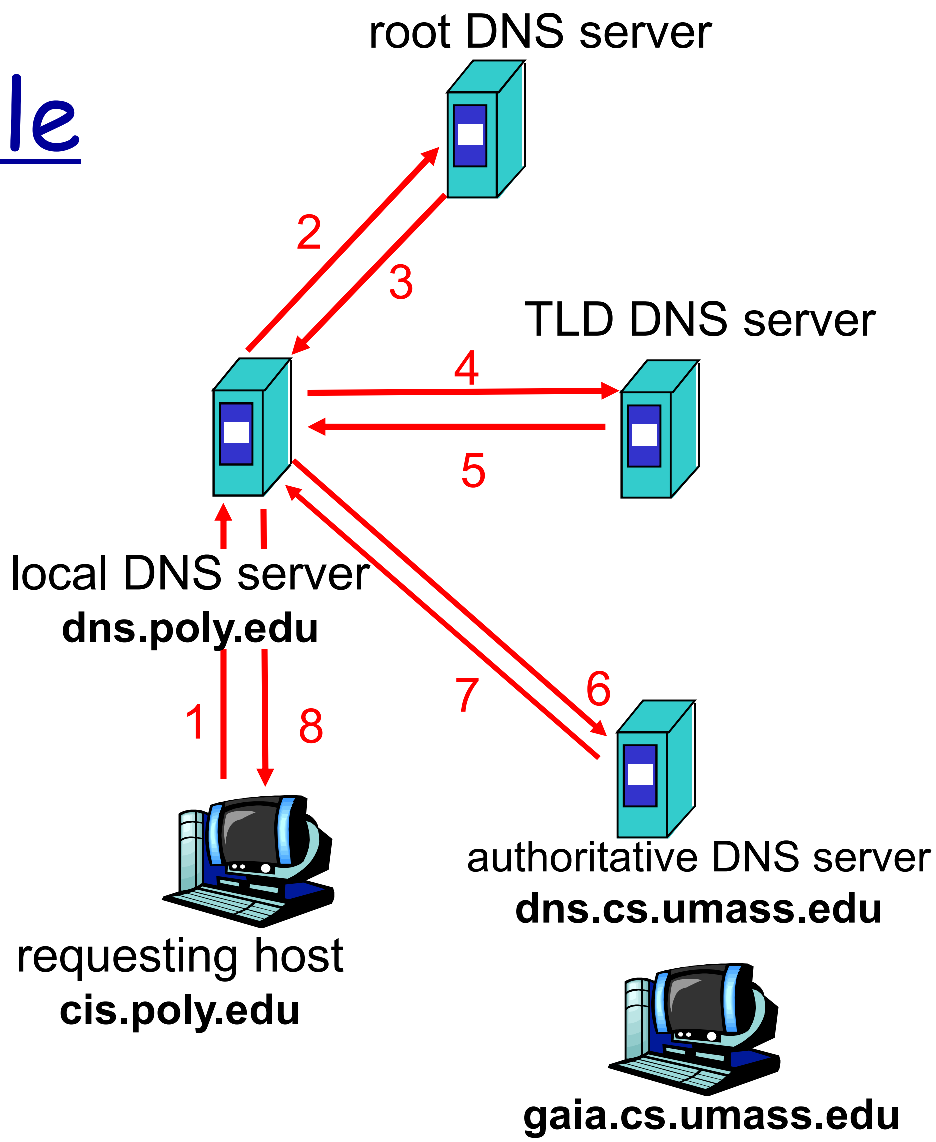
- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
  - also called "default name server"
- when host makes DNS query, query is sent to its local DNS server
  - acts as proxy, forwards query into hierarchy

DNS name resolution example

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

iterated query:

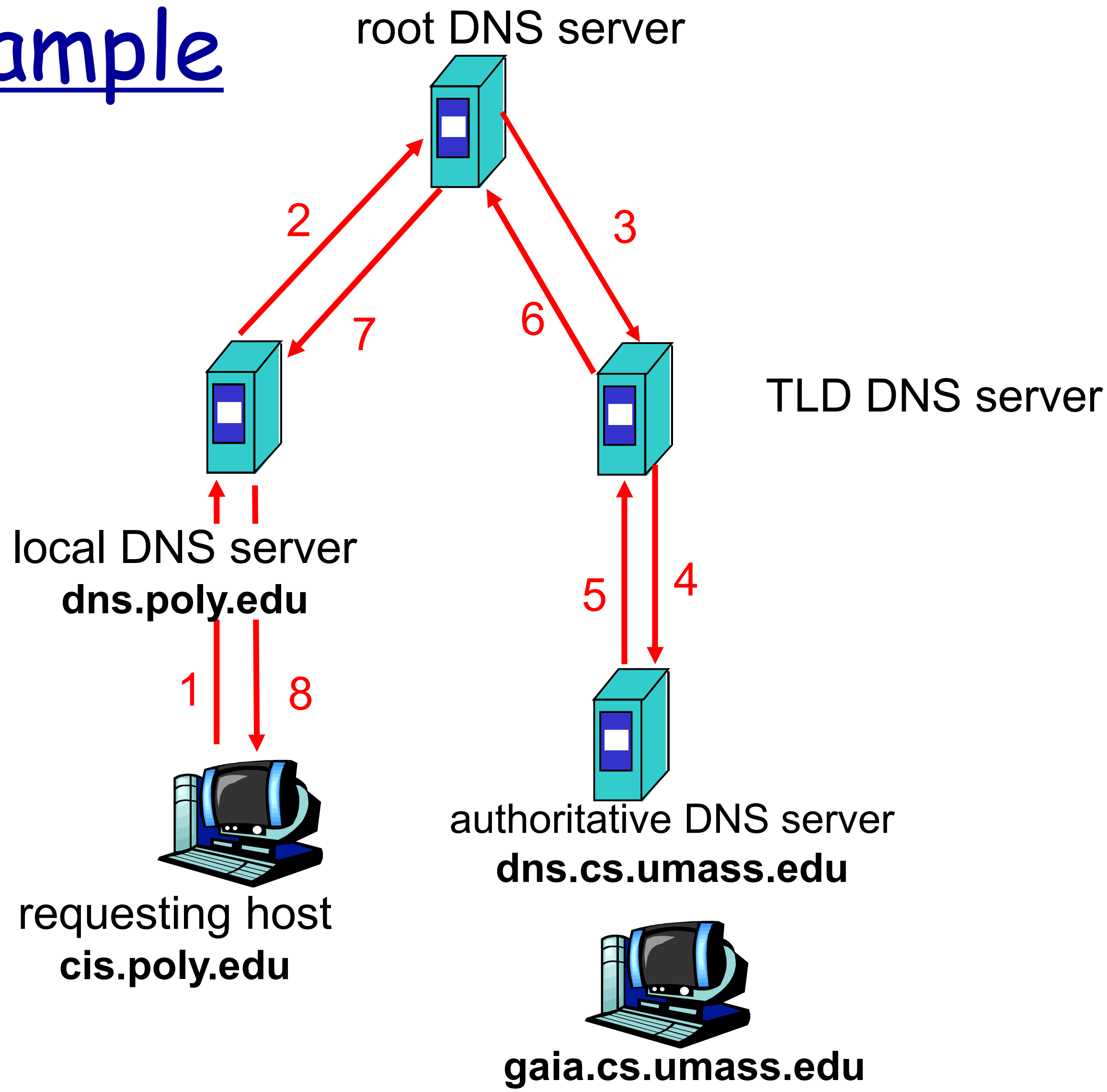
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



DNS name resolution example

recursive query:

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



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 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

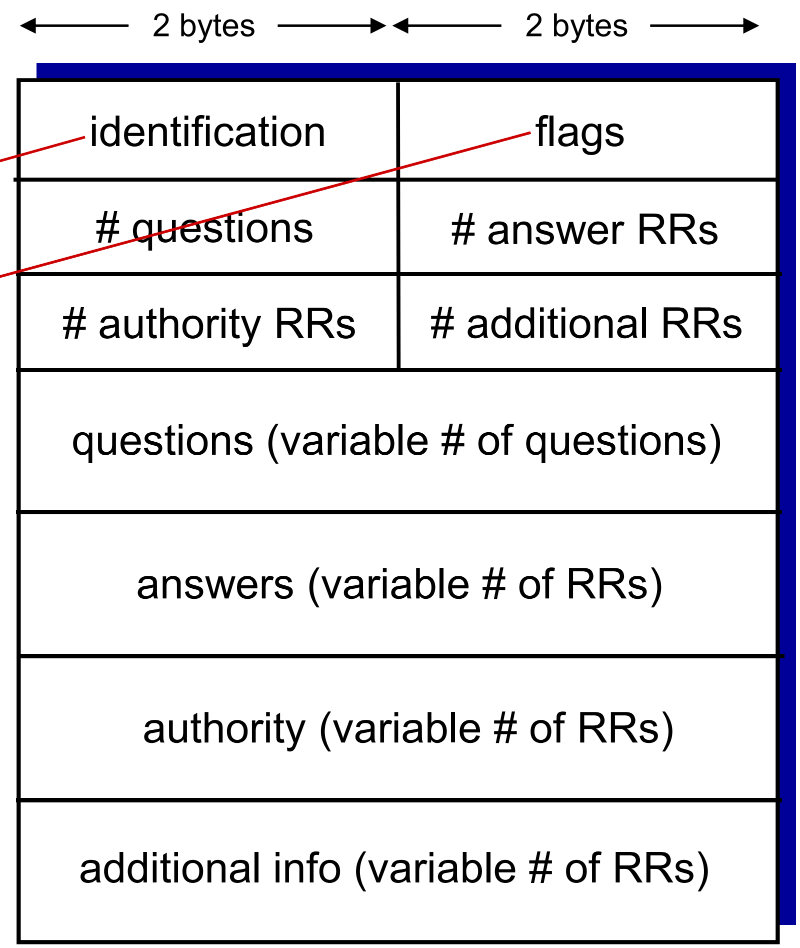
- **`value`** is name of mailserver associated with **`name`**

DNS protocol, messages

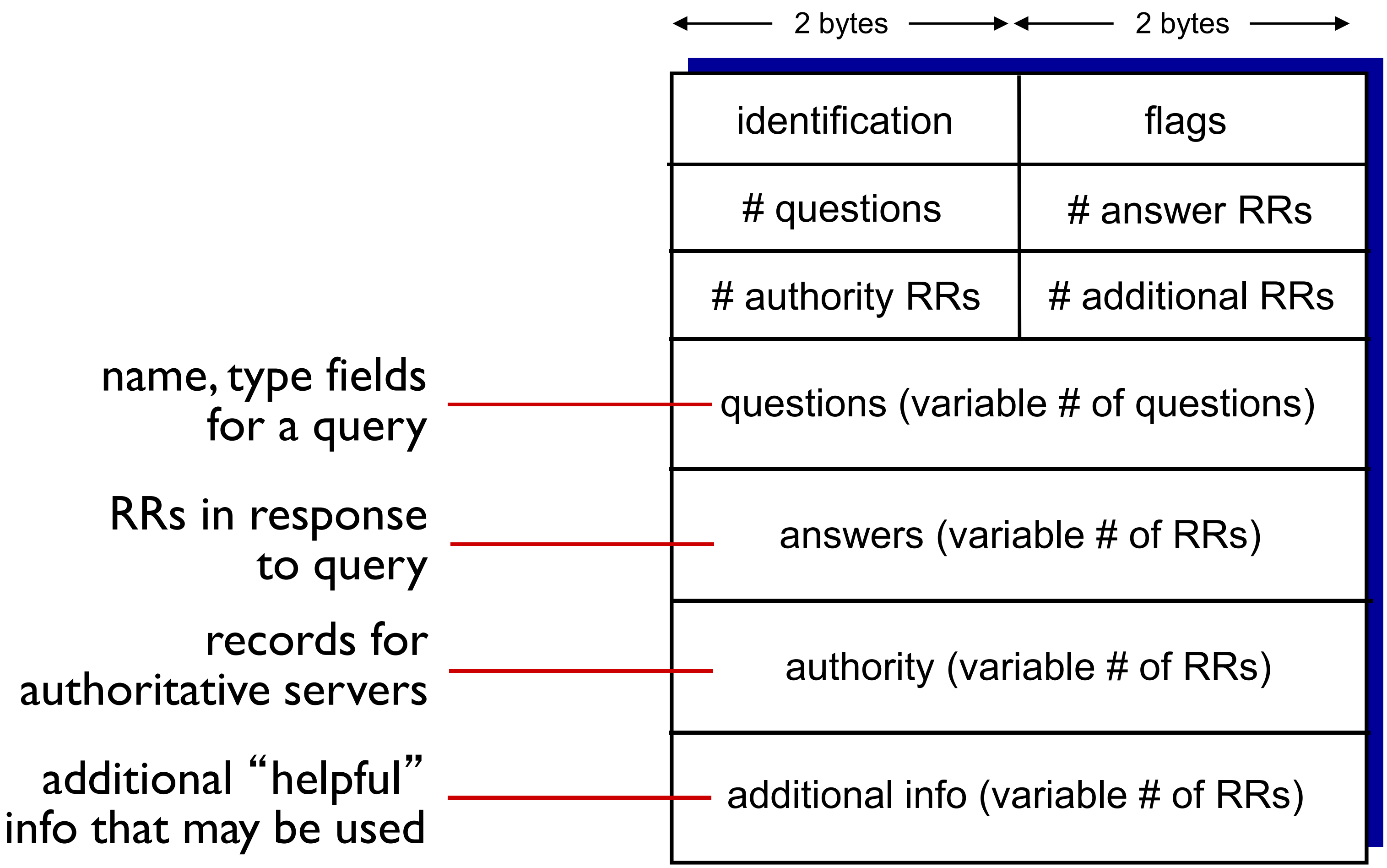
- *query* and *reply* messages, both with same *message format*

msg header

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



DNS protocol, messages



Inserting records into DNS

- example: new startup "Network Utopia"
- register name networkuptopia.com at *DNS registrar*
- *Examples of registrars?*
- *See -- http://www.internic.net*

Inserting records into DNS

- register verifies your name is unique and enters your domain into the DNS database.
  - 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?

Application Protocol Summary

- 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:

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

Application Layer: Summary

- application architectures
  - client-server
  - P2P
  - hybrid
- 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, Skype
- socket programming