Chapter 2
Application Layer – Part B

Peer-to-Peer Applications

- Adapted from Computer Networking: A Top Down Approach, Jim Kurose, Keith Ross
  Addison-Wesley

Outline

2.6 P2P applications

- Introduction
- P2P Architectures
- P2P Protocols
- Case Study: BitTorrent

What is P2P?

- "the sharing of computer resources and services by direct exchange of information"

What is P2P?

- "P2P is a class of applications that take advantage of resources – storage, cycles, content, human presence – available at the edges of the Internet. Because accessing these decentralized resources means operating in an environment of unstable and unpredictable IP addresses P2P nodes must operate outside the DNS system and have significant, or total autonomy from central servers"

What is P2P?

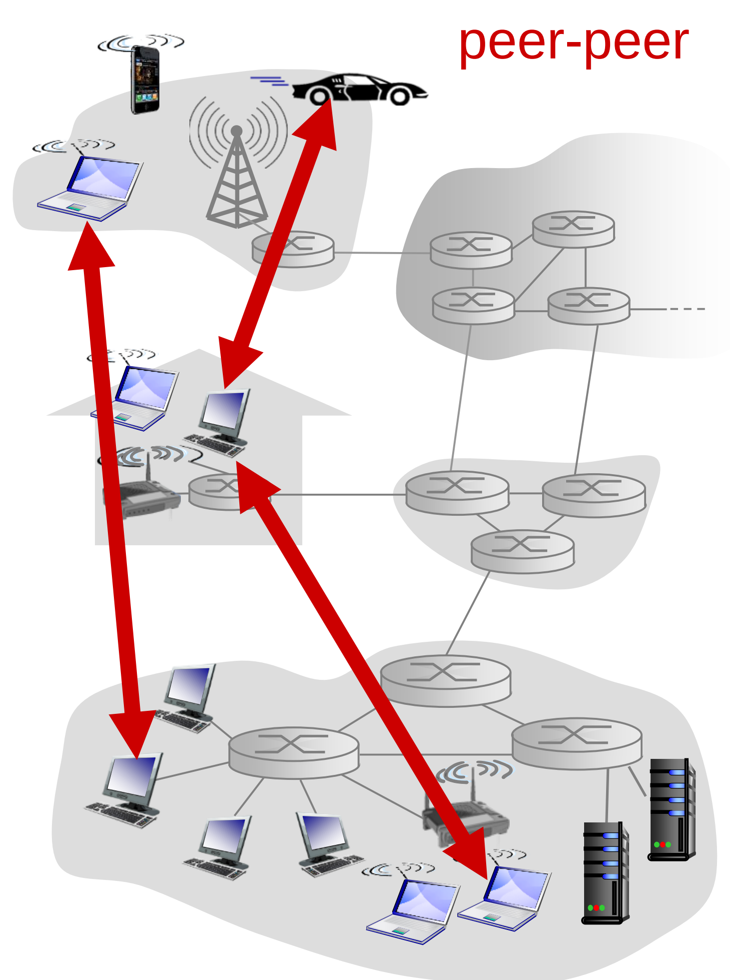
- "A distributed network architecture may be called a P2P network if the participants share a part of their own resources. These shared resources are necessary to provide the service offered by the network. The participants of such a network are both resource providers and resource consumers"

What is a *peer*?

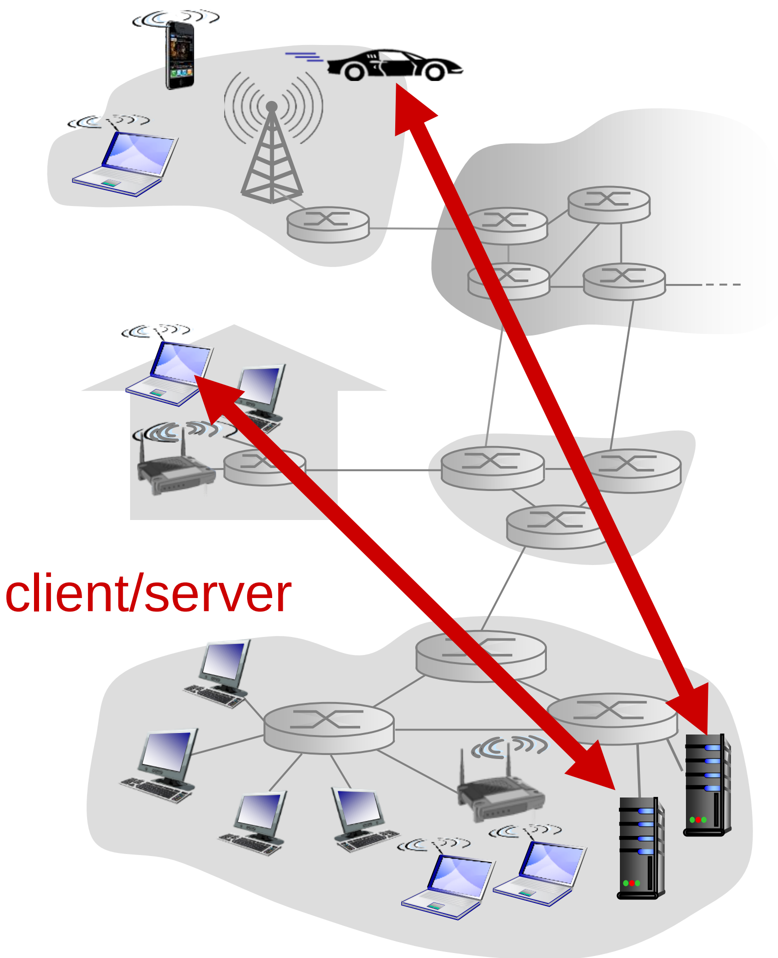
- "…an entity with capabilities similar to other entities in the system."

P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - *self scalability* – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
  - complex management



Client-server architecture



server:

- always-on host
- permanent IP address
- data centers for scaling

clients:

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

P2P Network Characteristics

- Clients are also servers and routers
  - Nodes contribute content, storage, memory, CPU
- Nodes are autonomous (no administrative authority)
- Network is dynamic: nodes enter and leave the network "frequently"
- Nodes collaborate directly with each other (not through well-known servers)
- Nodes have widely varying capabilities

P2P Goals and Benefits

- Efficient use of resources
  - Unused bandwidth, storage, processing power at the "edge of the network"
- Scalability
  - No central information, communication and computation bottleneck
  - Aggregate resources grow naturally with utilization
- Reliability
  - Replicas
  - Geographic distribution
  - No single point of failure
- Ease of administration
  - Nodes self-organize
  - Built-in fault tolerance, replication, and load balancing
  - Increased autonomy
- Anonymity – Privacy
  - not easy in a centralized system
- Dynamism
  - highly dynamic environment
  - ad-hoc communication and collaboration

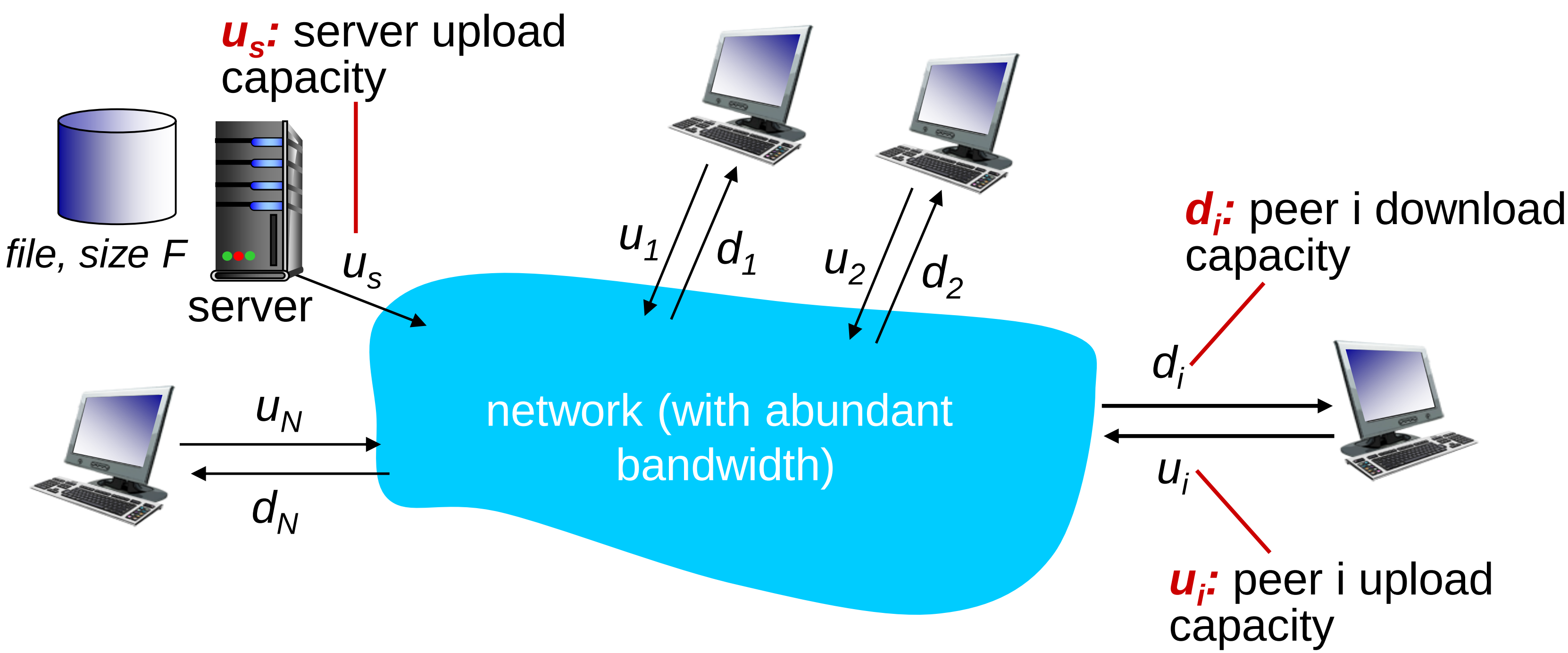
P2P Applications

- File sharing (Napster, Gnutella, Kazaa, others?)
- Multiplayer games (Unreal Tournament, DOOM)
- Collaborative applications (ICQ, shared whiteboard)
- Distributed computation (Seti@home)
- Ad-hoc networks

File distribution: client-server vs P2P

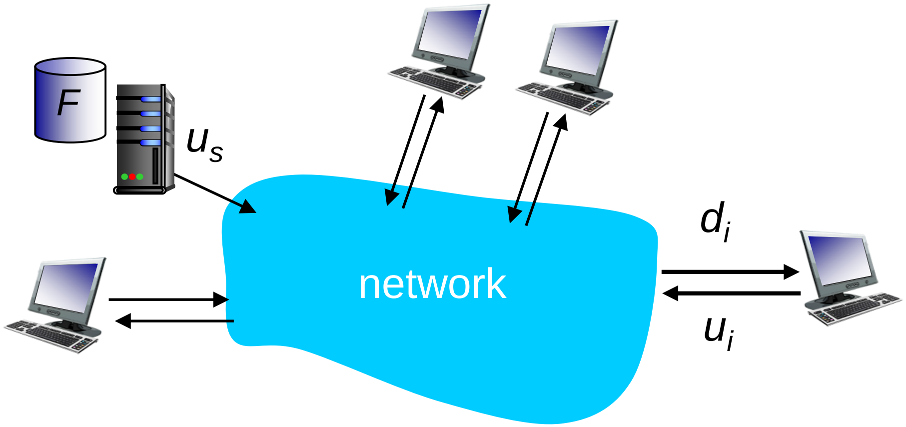
Question: how much time to distribute file (size *F*) from one server to *N* *peers*?

- 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}



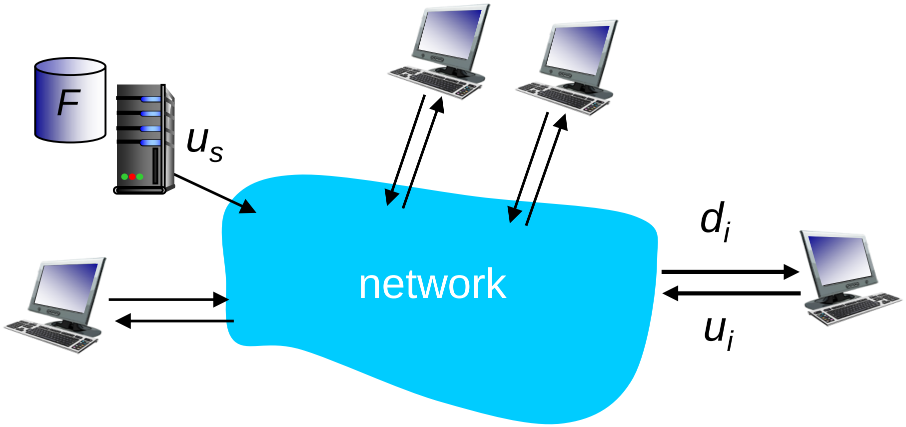
time to distribute F to N clients using client-server approach

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

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

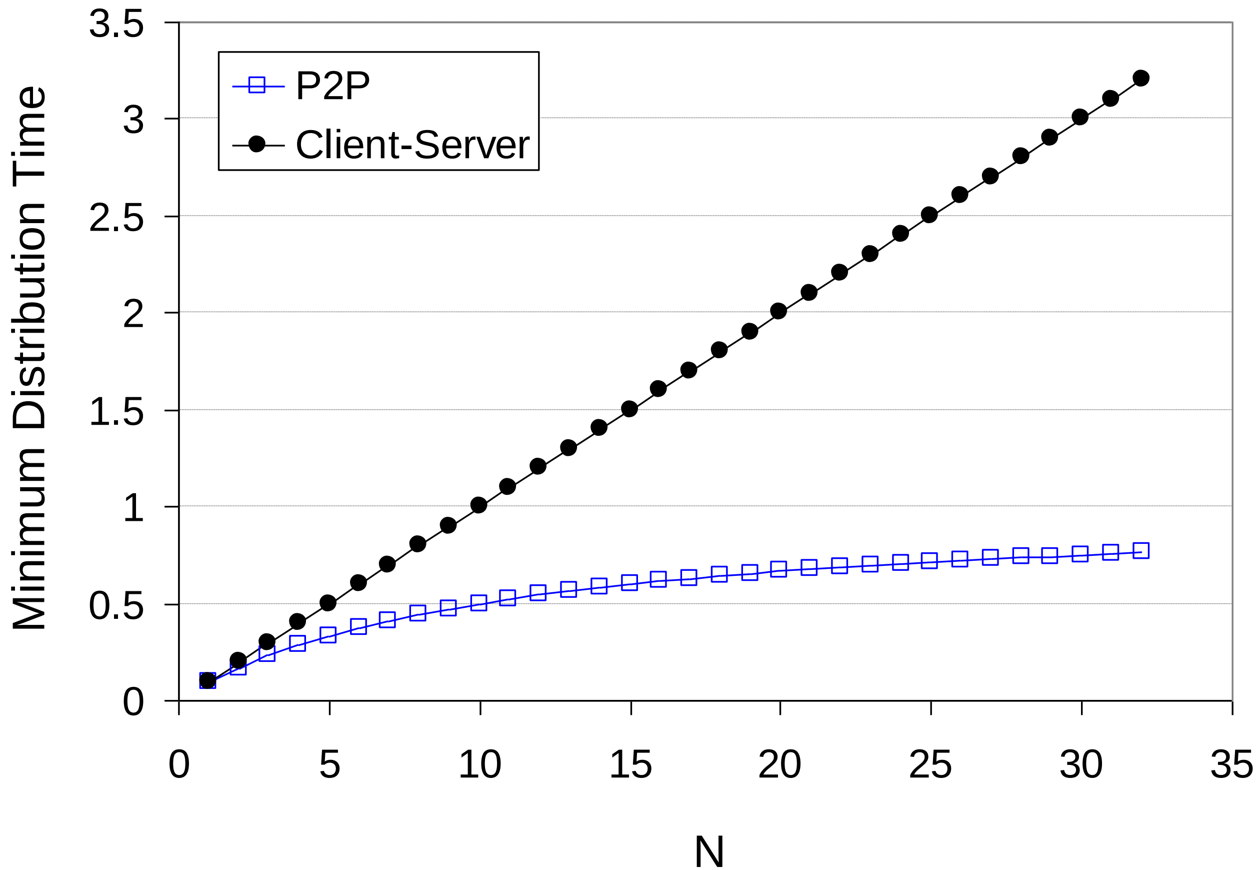
$$D_{P2P} \geq max\{F/u_{s,},F/d_{min,},NF/(u_s + \Sigma u_i)\}$$

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

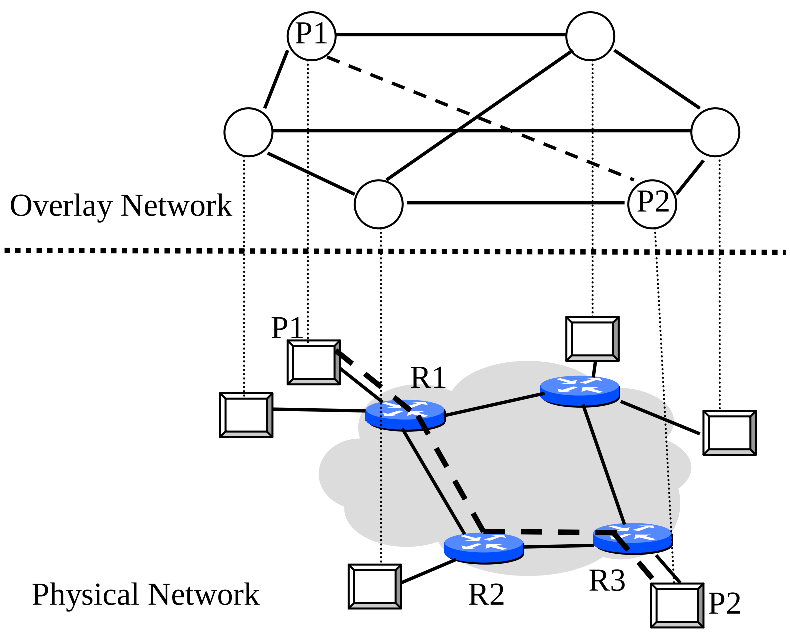


Introduction to P2P

- Main operations in P2P systems
  - Join the P2P overlay network
  - Resource discovery
    - Publish resources to be shared (optional)
    - Discover resource
  - Resource retrieval

P2P protocols

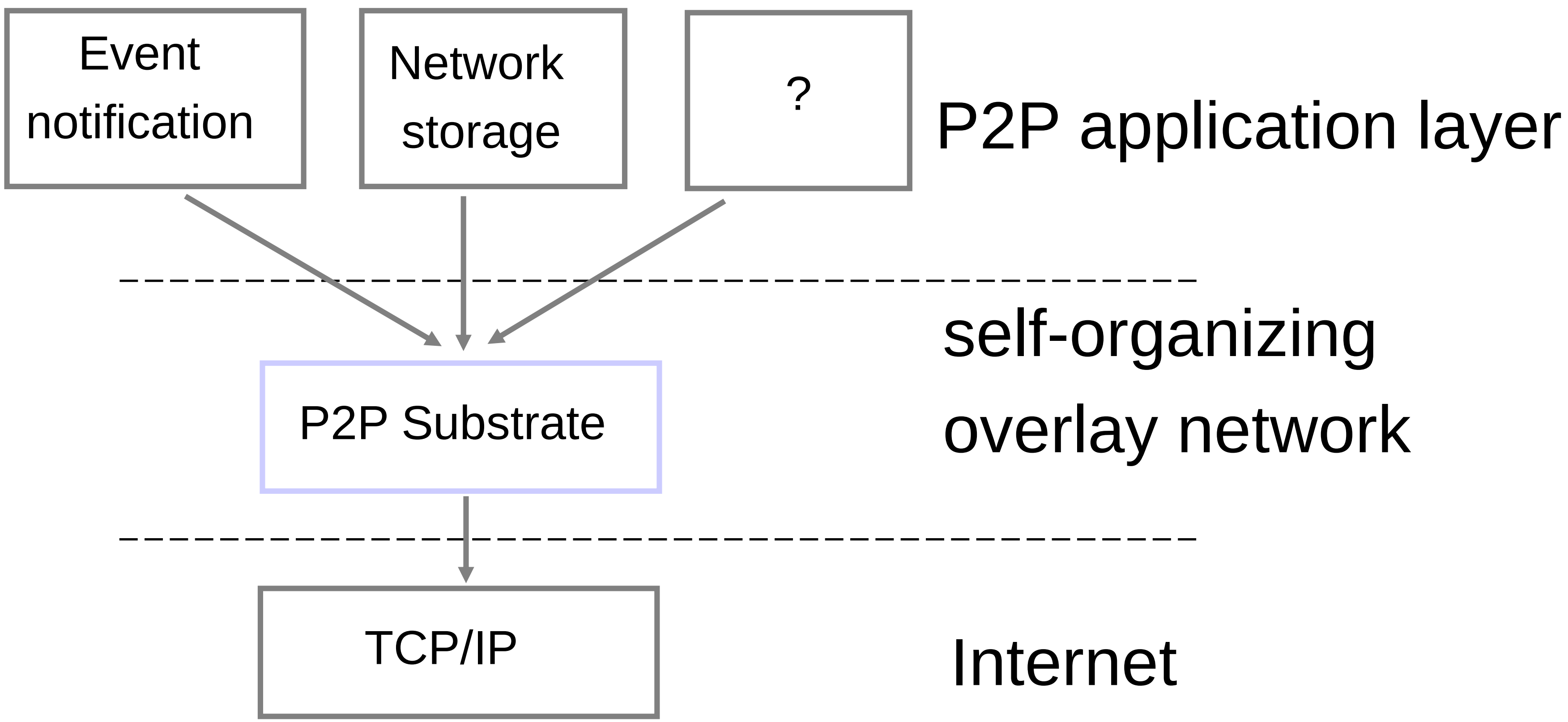
- Distributed network architecture
  - A virtual overlay network at the application layer
  - Participants act as both a client and a server



Node: peers

Edge: TCP/UDP connection

P2P Operation



P2P Architectures

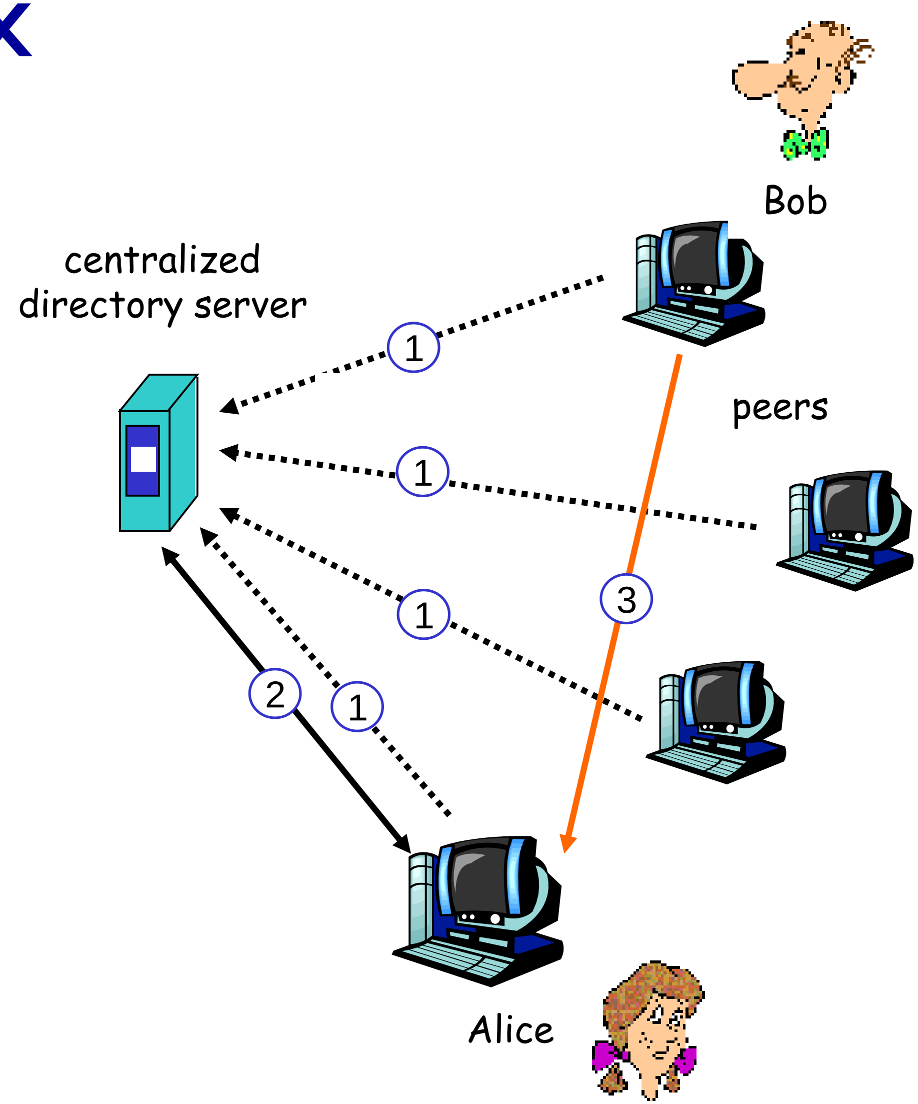
- Three types of P2P systems
  - Centralized
  - Decentralized and unstructured
  - Decentralized but structured
- Some P2P systems also adopt hybrid architecture
  - Hybrid of centralized and decentralized (unstructured or structured)

Centralized index

file transfer is decentralized, but locating content is highly centralized

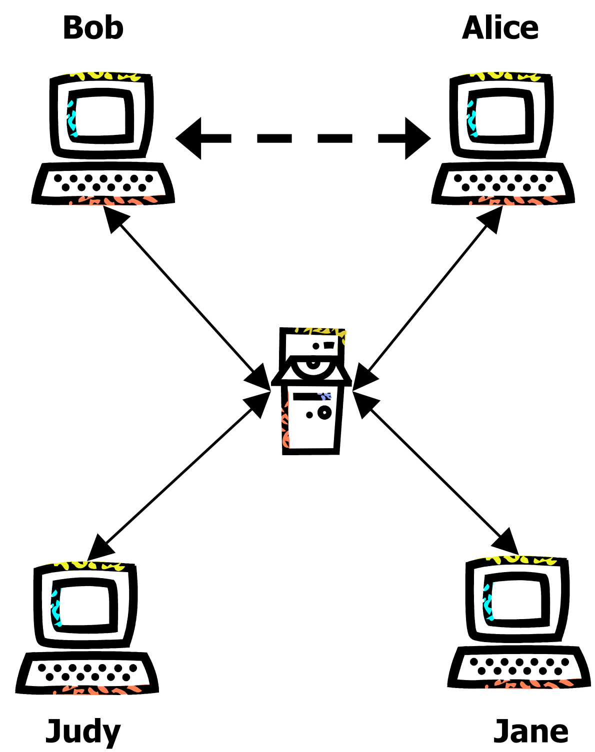
original "Napster" design

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



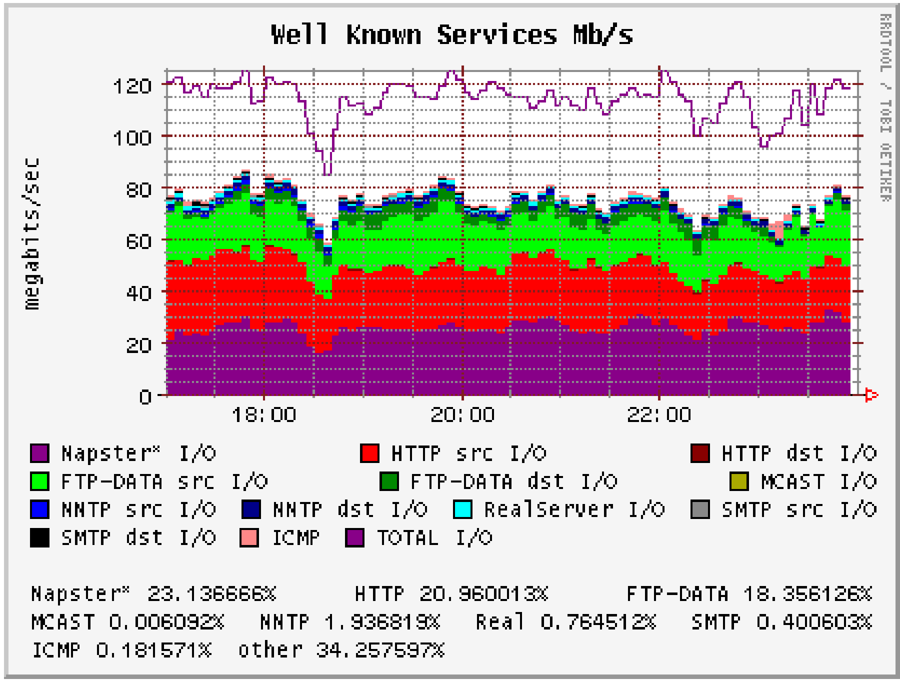
Centralized

- Benefits:
  - Low per-node state
  - Limited bandwidth usage
  - Short location time
  - High success rate
  - Fault tolerant
- Drawbacks:
  - Single point of failure
  - Limited scale
  - Possibly unbalanced load
- copyright infringement



Napster

- program for sharing files over the Internet
- a "disruptive" application/technology?
- history:
  - 5/99: Shawn Fanning (freshman, Northeasten U.) founds Napster Online music service
  - 12/99: first lawsuit
  - 3/00: 25% UWisc traffic Napster
  - 2000: est. 60M users
  - 2/01: US Circuit Court of Appeals: Napster knew users violating copyright laws
  - 7/01: # simultaneous online users: Napster 160K, Gnutella: 40K, Morpheus: 300K



Napster: how does it work

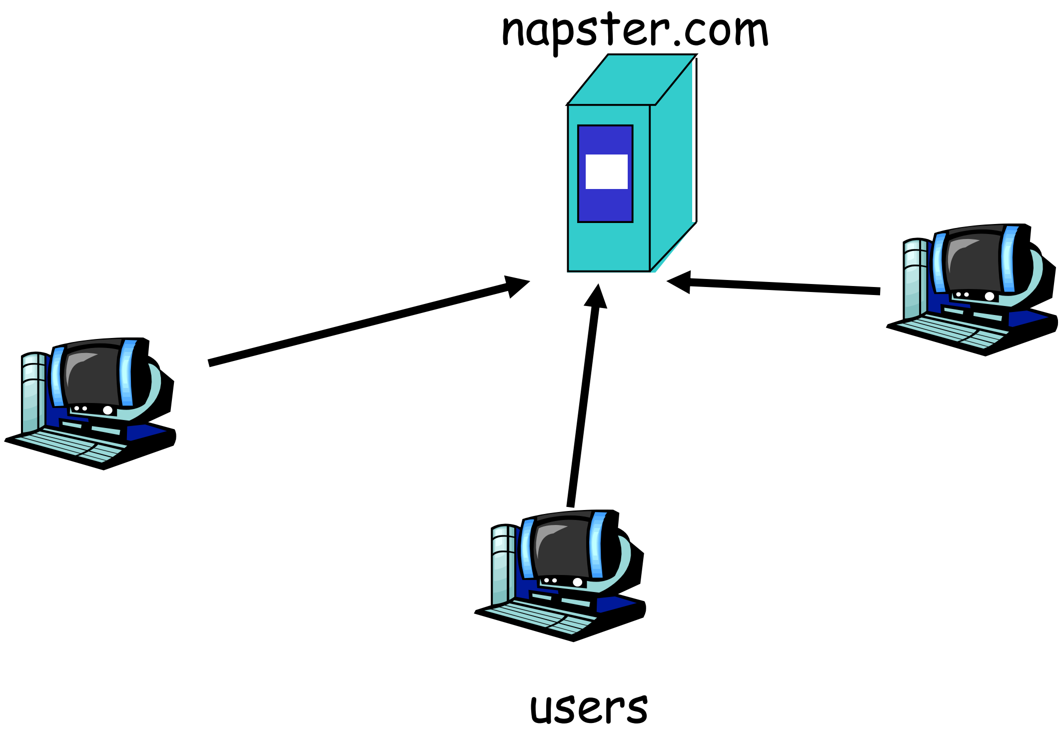
Application-level, client-server protocol over point-to-point TCP

Four steps:

- Connect to Napster server
- Upload your list of files (push) to server.
- Give server keywords to search the full list with.
- Select "best" of correct answers. (pings)

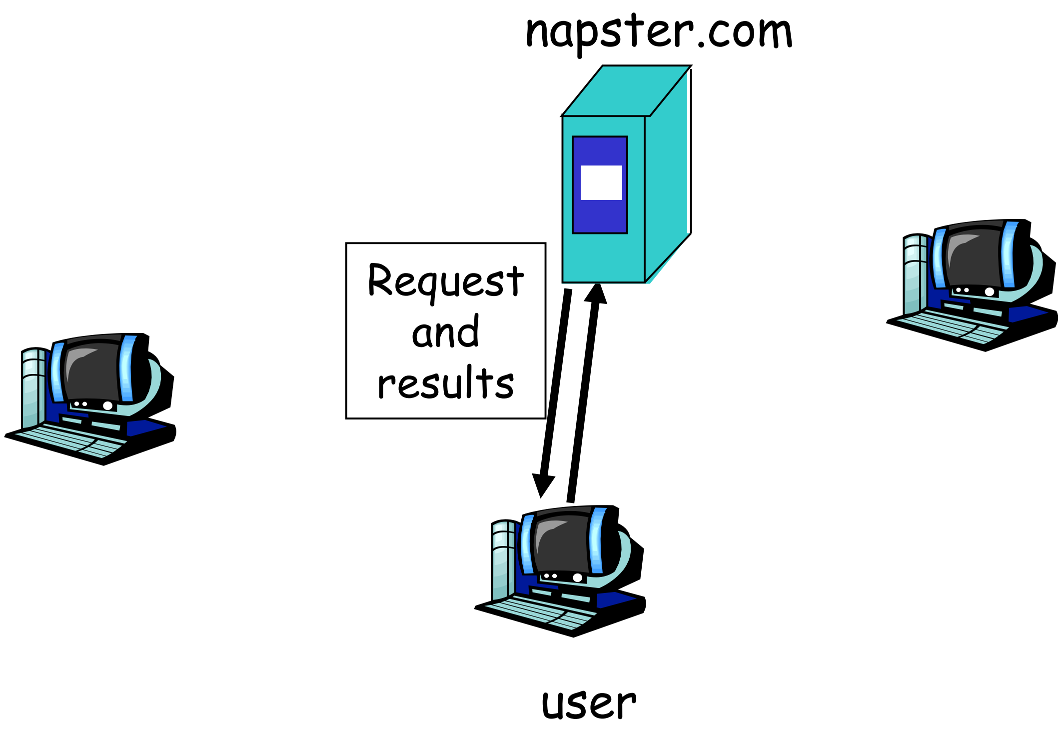
Napster

1. File list is uploaded



Napster

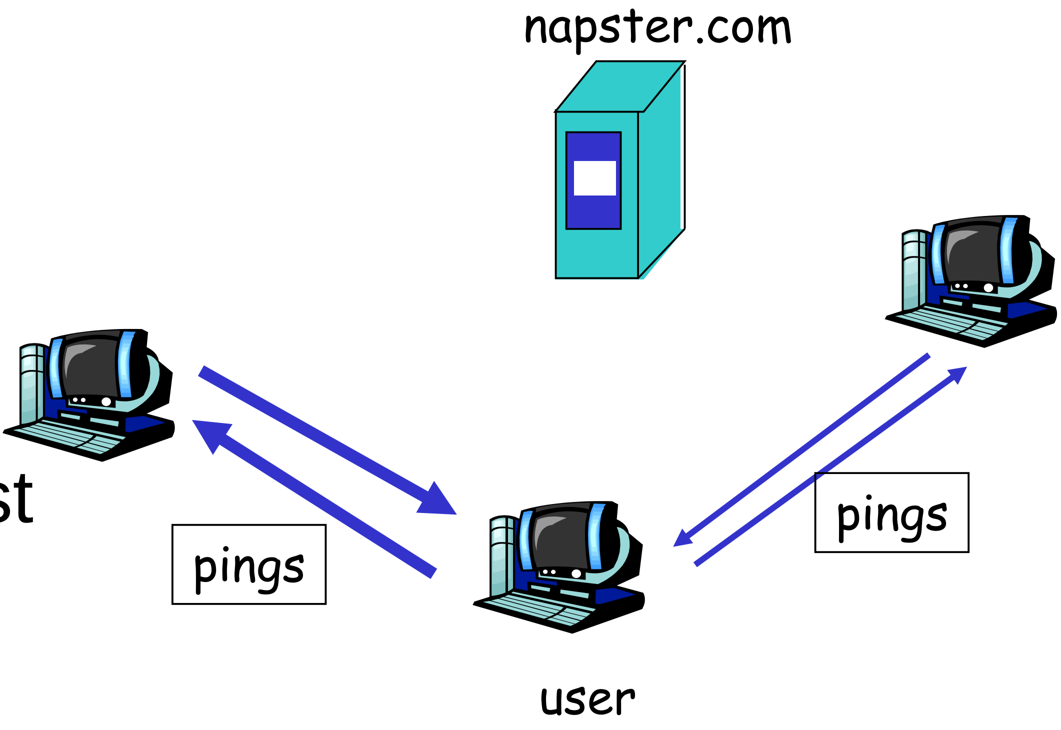
2. User requests search at server.



Napster

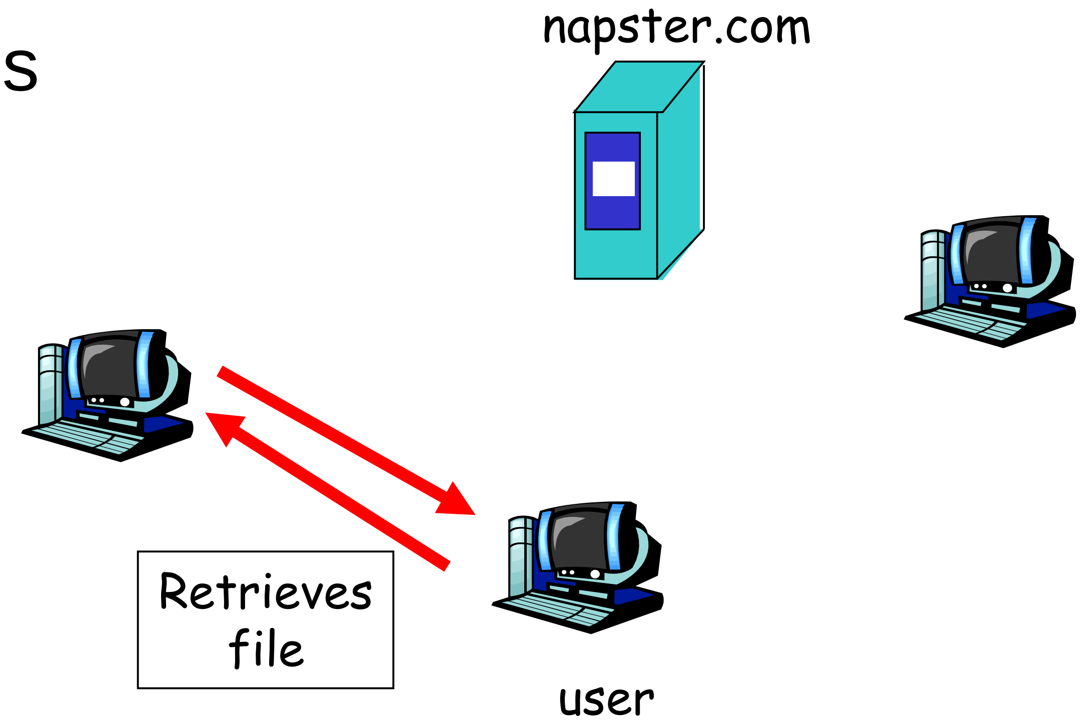
3. User pings hosts that apparently have data.

   Looks for best transfer rate.



Napster

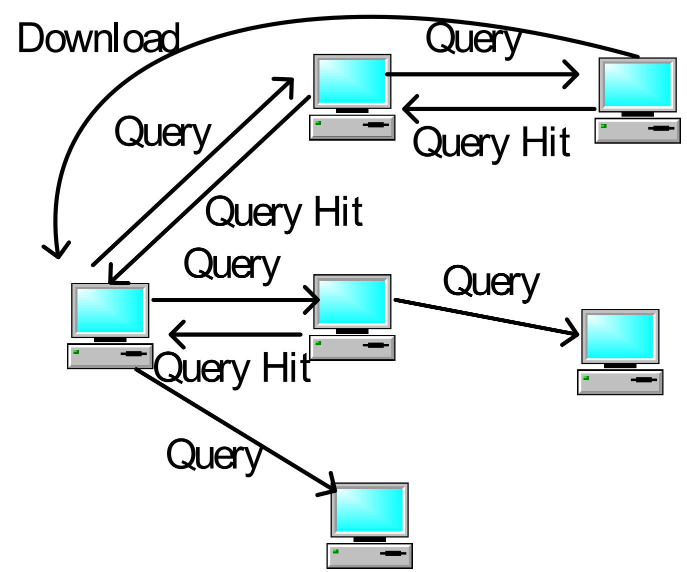
4. User retrieves file



Decentralized and Unstructured P2P

- Floods query messages to peers to search for shared objects
  - No central server, no publication of shared objects
  - Limited-scope flooding to reduce flooding messages
  - A query hit message is returned along the reverse path back to the inquirer

Example: Gnutella



Decentralized and Unstructured P2P

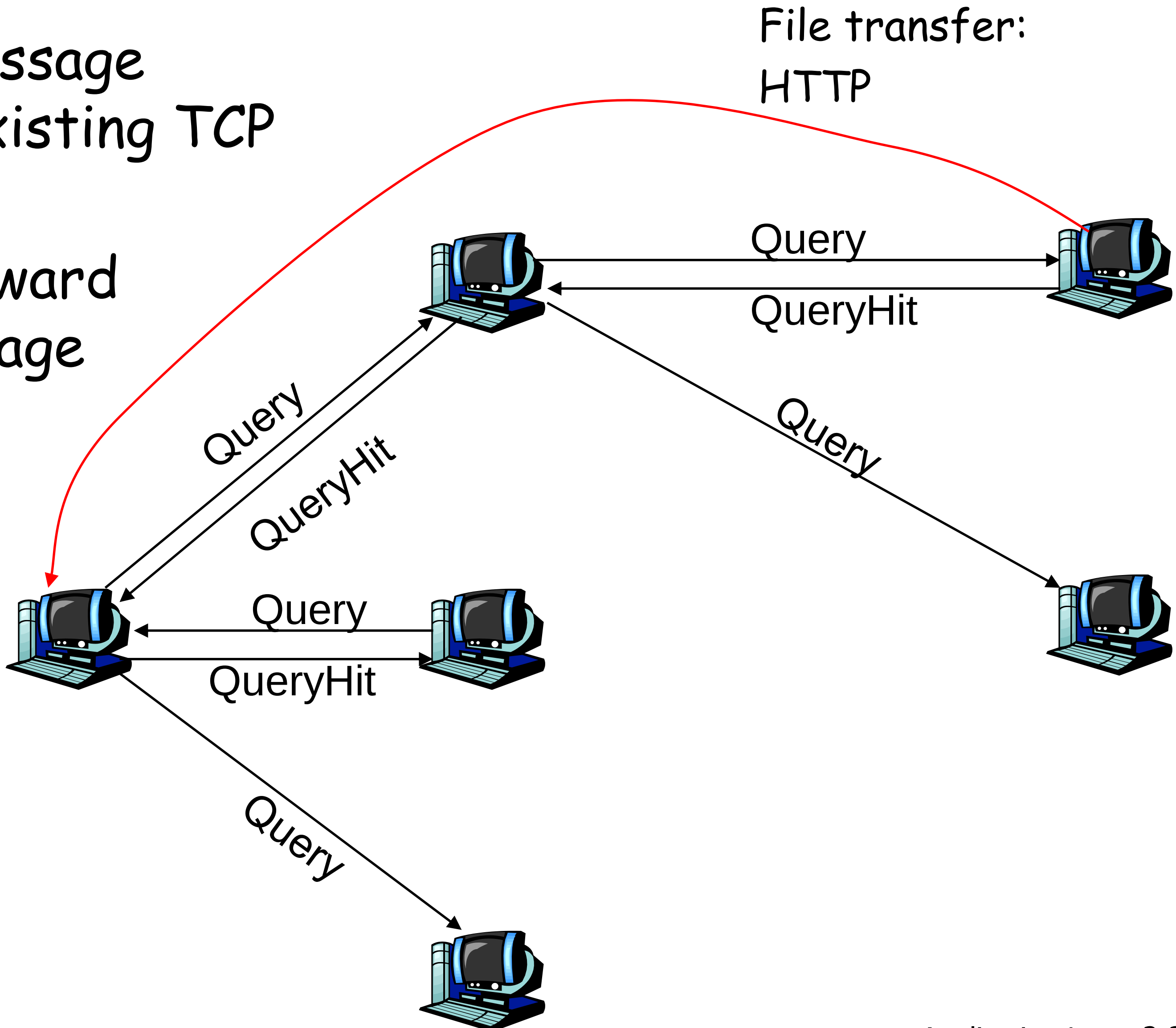
- Join procedure
  - A newcomer sends a join message to a peer already on the overlay.
  - The existing peer replies its identity as well as a list of its neighbors
    - *May also forward the join message to its neighbors*
  - Upon receiving join reply messages, the newcomer knows more peers on the overlay.

Decentralized and Unstructured P2P

- Advantages
  - Fully distributed
  - Reliable, fault-tolerant
  - No single point of failure
- Disadvantages
  - Excessive query traffic make it not scalable
  - May fail to find content that is actually in the system
  - Super peer may become overloaded or been attacked

Gnutella: Query flooding

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



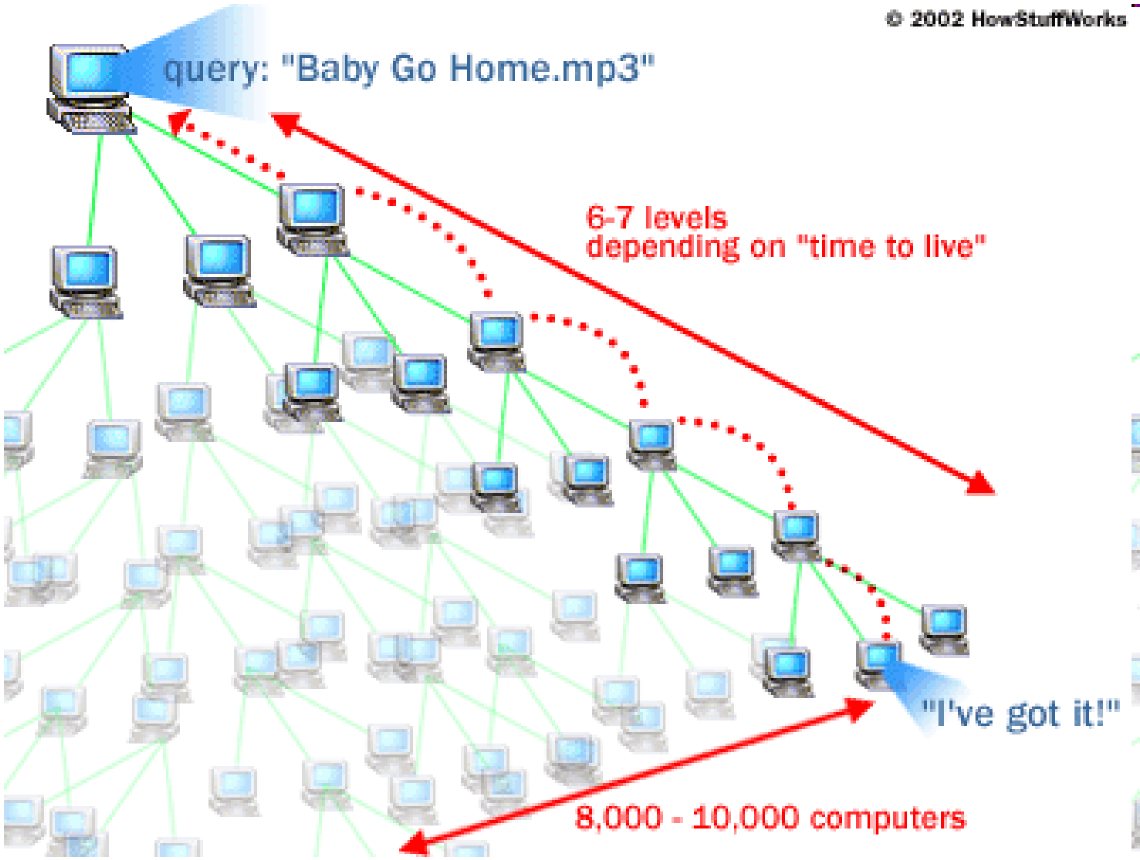
Gnutella: Peer joining

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

Gnutella

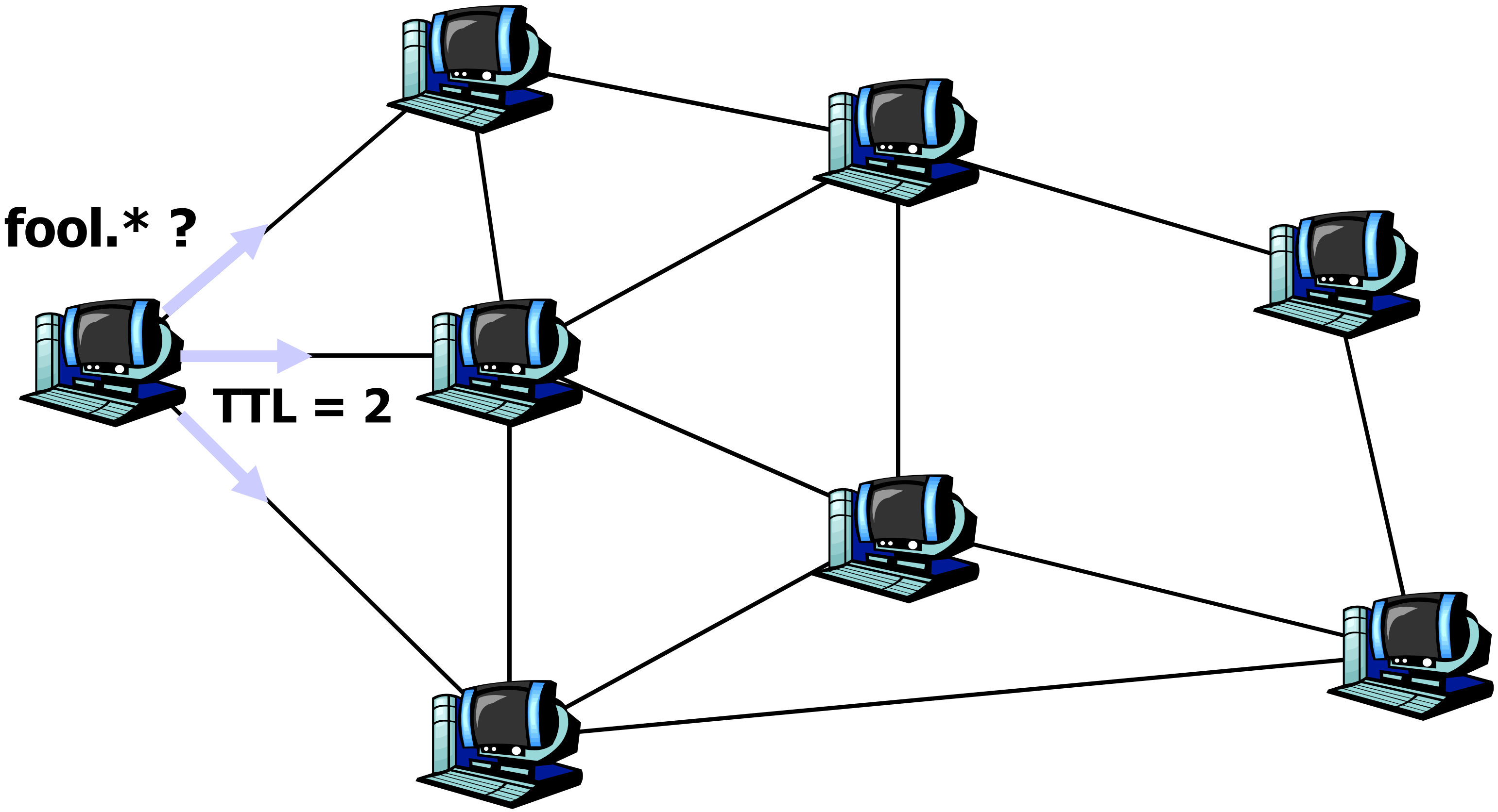
Searching by flooding:

- If you don't have the file you want, query 7 of your neighbors.
- If they don't have it, they contact 7 of their neighbors, for a maximum hop count of 10.
- Requests are flooded, but there is no tree structure.
- No looping but packets may be received twice.
- Reverse path forwarding

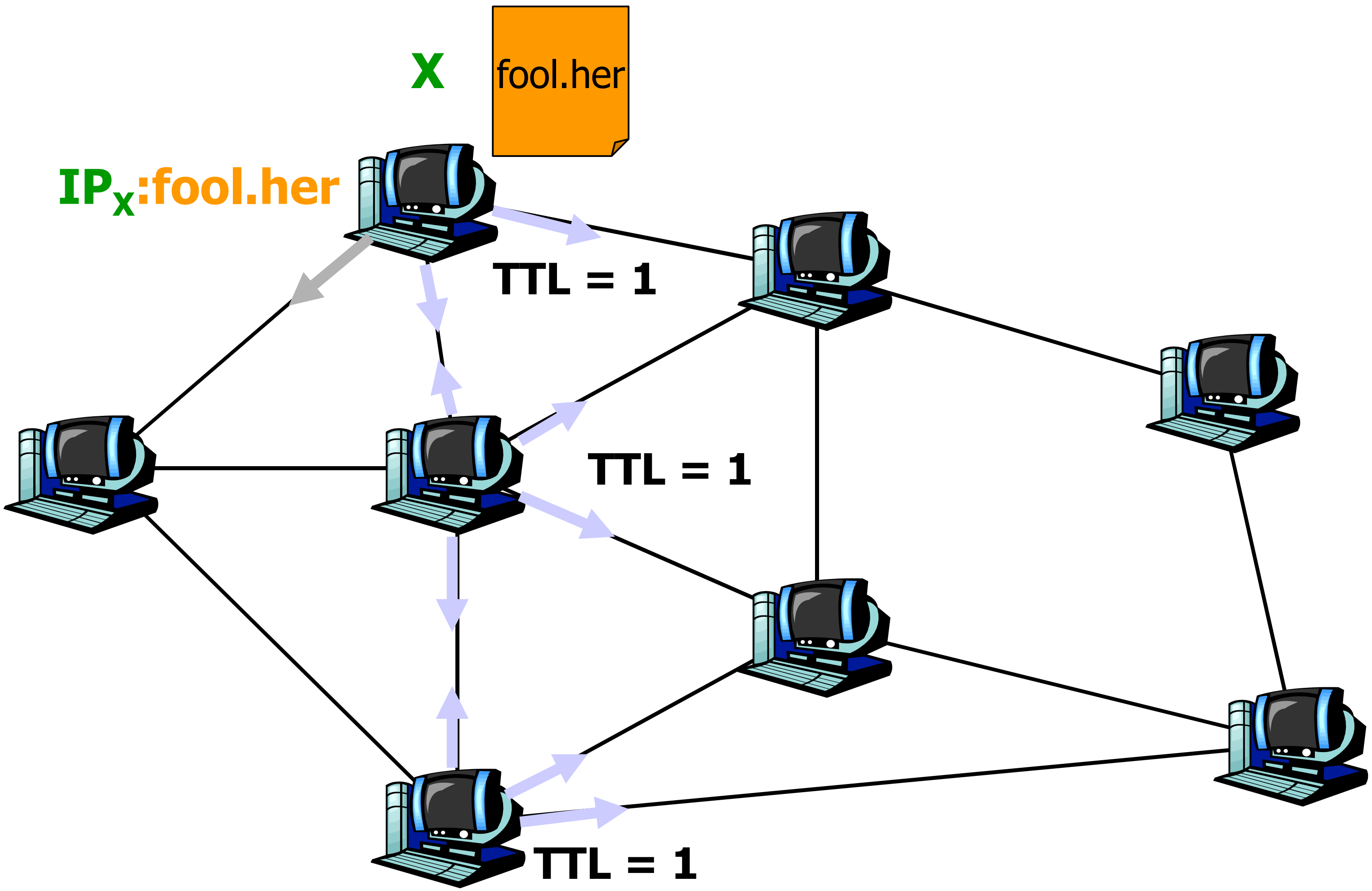


* Figure from http://computer.howstuffworks.com/file-sharing.htm

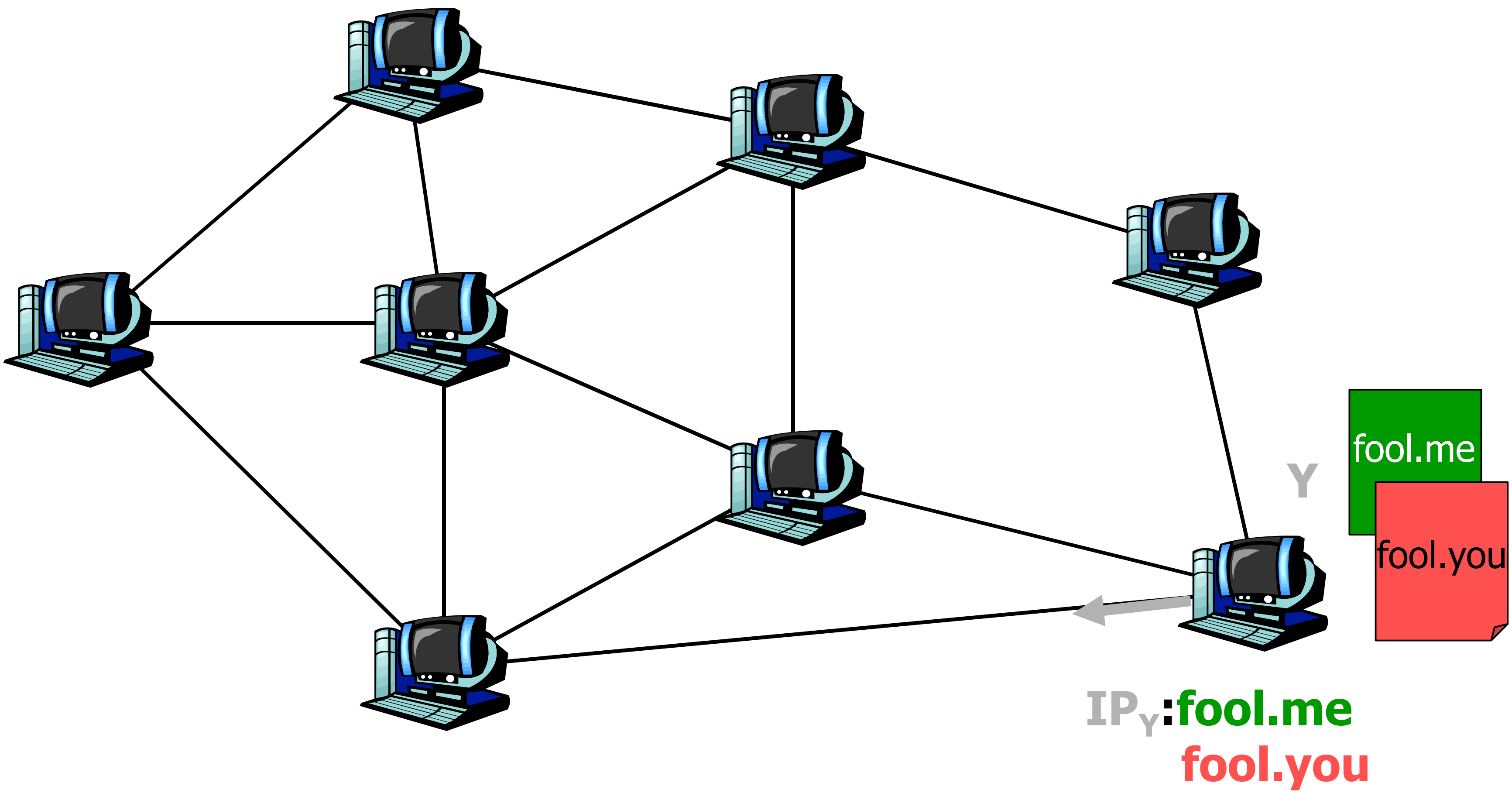
Gnutella



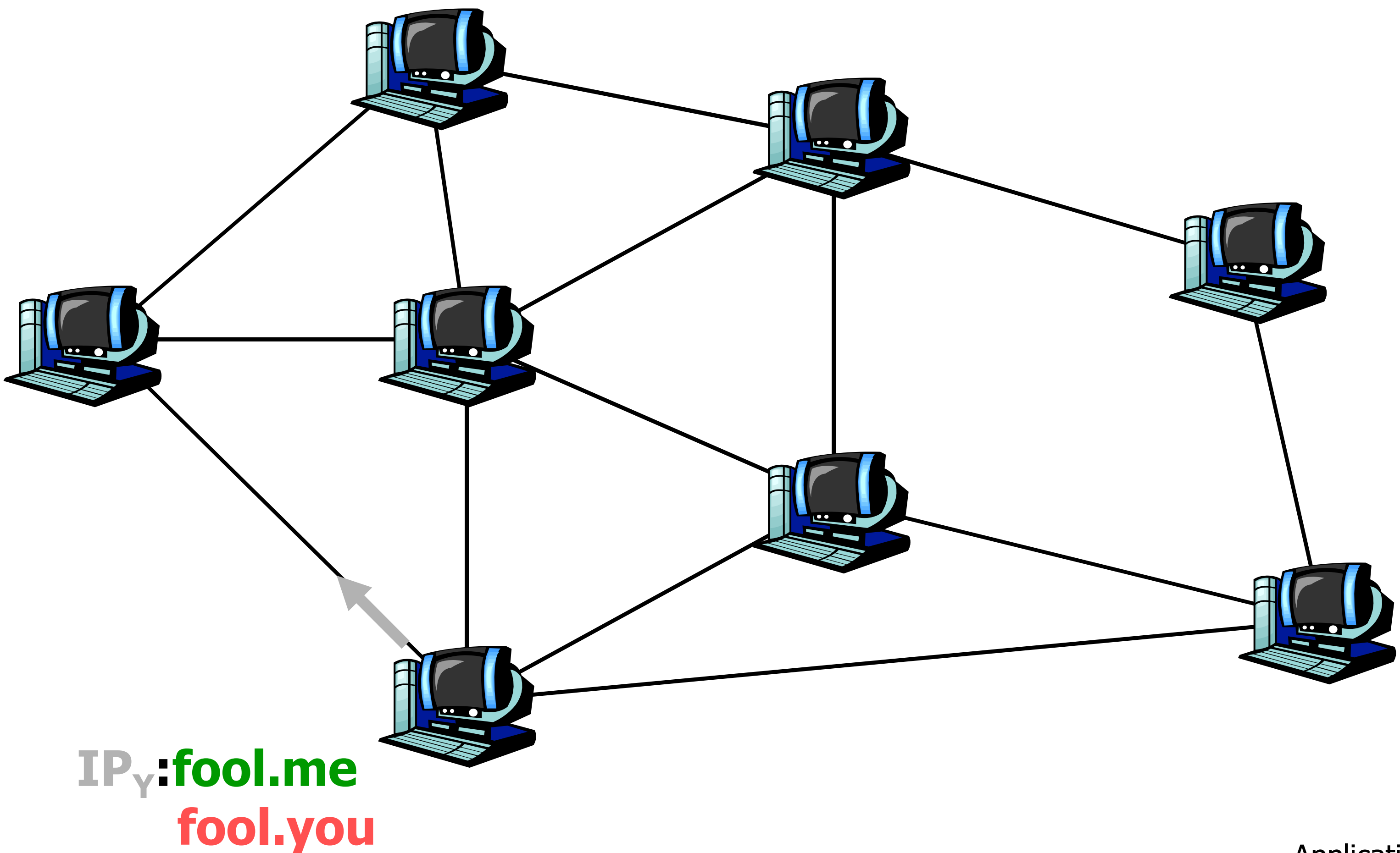
Gnutella



Gnutella



Gnutella



Gnutella: strengths and weaknesses

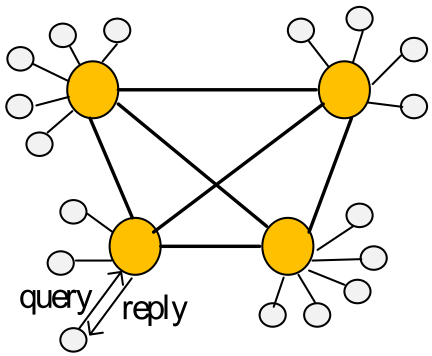
- pros:
  - ☑ flexibility in query processing
  - ☑ complete decentralization
  - ☑ simplicity
  - ☑ fault tolerance/self-organization
- cons:
  - ☒ severe scalability problems
  - ☒ susceptible to attacks
- Pure P2P system

Gnutella: initial problems and fixes

- 2000: avg size of reachable network only 400-800 hosts. Why so small?
  - modem users: not enough bandwidth to provide search routing capabilities: routing black holes
- Fix: create peer hierarchy based on capabilities
  - previously: all peers identical, most modem black holes
  - preferential connection:
    - favors routing to well-connected peers
    - favors reply to clients that themselves serve large number of files: prevent freeloading

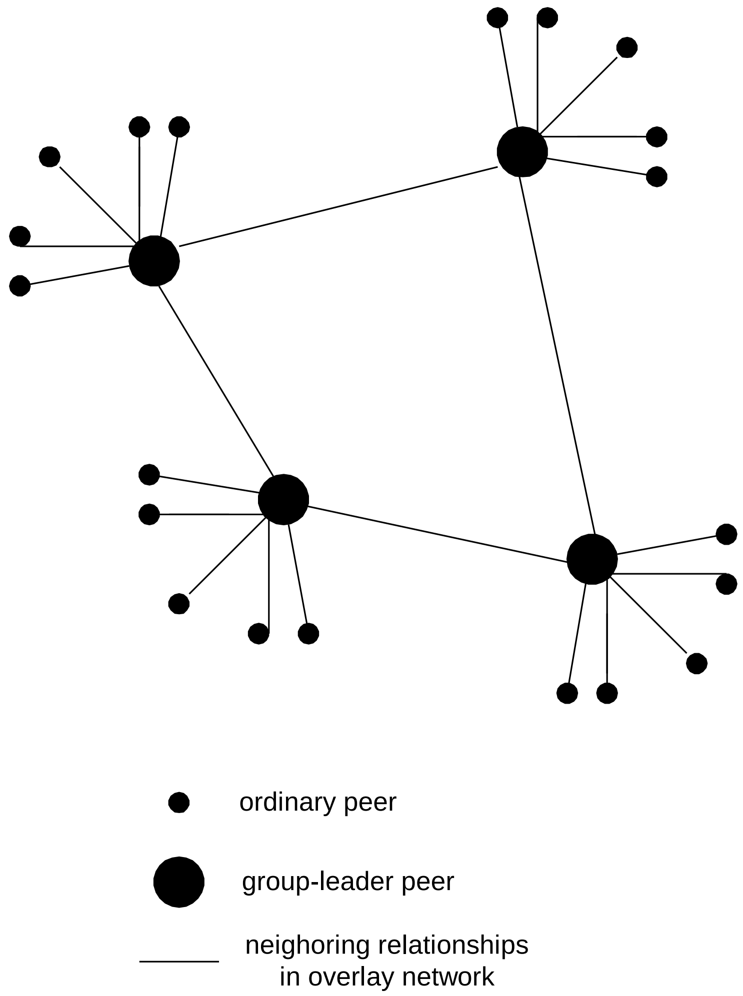
Decentralized and Unstructured P2P

- Hierarchical overlay with super peers
  - Flooding is apparently not scalable
  - FastTrack adopts a hierarchical overlay
  - A super peer acts as a local directory database which stores the indexes of objects shared by ordinary peers
  - Two-level hierarchical overlay
    - The lower level adopts the central server approach
    - The upper level (super peers) adopts the decentralized and unstructured approach.



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



Kazaa (Fasttrack network)

- Hybrid of centralized Napster and decentralized Gnutella
  - hybrid P2P system

- Super-peers act as local search hubs
  - Each super-peer is similar to a Napster server for a small portion of the network
  - Super-peers are automatically chosen by the system based on their capacities (storage, bandwidth, etc.) and availability (connection time)

- Users upload their list of files to a super-peer
- Super-peers periodically exchange file lists
- You send queries to a super-peer for files of interest

Unstructured vs Structured P2P

- The systems we described do not offer any guarantees about their performance (or even correctness)
- Structured P2P
  - Scalable guarantees on numbers of hops to answer a query
  - Maintain all other P2P properties (load balance, self-organization, dynamic nature)

- Approach: Distributed Hash Tables (DHT)

Decentralized but Structured

- Combine the distributed directory service with an efficient query routing scheme
- Key ideas
  - Distributed directory service
    - Hash function maps peers and objects into the same address space
  - Efficient query routing
    - Peers are organized into a structured overlay based on their positions in the address space

Distributed Hash Table (DHT)

- DHT: a *distributed P2P database*
- database has (key, value) pairs; examples:
  - key: ss number; value: human name
  - key: movie title; value: IP address
- Distribute the (key, value) pairs over the (millions of peers)
- a peer queries DHT with key
  - DHT returns values that match the key
- peers can also insert (key, value) pairs

Distributed Hash Table (DHT)

- Distributed version of a hash table data structure
- Stores (key, value) pairs
  - The key is like a filename
  - The value can be file contents, or pointer to location
- Goal: Efficiently insert/lookup/delete (key, value) pairs

- Each peer stores a subset of (key, value) pairs in the system
- Core operation: Find node responsible for a key
  - Map key to node
  - Efficiently route insert/lookup/delete request to this node
- Allow for frequent node arrivals/departures

DHT Desirable Properties

- Keys should mapped evenly to all nodes in the network (load balance)
- Each node should maintain information about only a few other nodes (scalability, low update cost)
- Messages should be routed to a node efficiently (small number of hops)
- Node arrival/departures should only affect a few nodes

Basic Approach

In all approaches:

- keys are associated with globally unique IDs
  - integers of size m (for large m)
- key ID space (search space) is uniformly populated - mapping of keys to IDs using (consistent) hashing
- a node is responsible for indexing all the keys in a certain subspace (zone) of the ID space
- nodes have only partial knowledge of other node's responsibilities

Decentralized but Structured

- Operations overview
  - Each peer generates its ID by a hash function
  - Each peer generates IDs of objects to be shared by the same or another hash function
  - For each object, the peer sends a register message to the node that has the node ID same as the object's ID.
  - To query an object, a peer uses the hash function to generate the object ID and sends the query message to the node that hosts the object's ID.

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

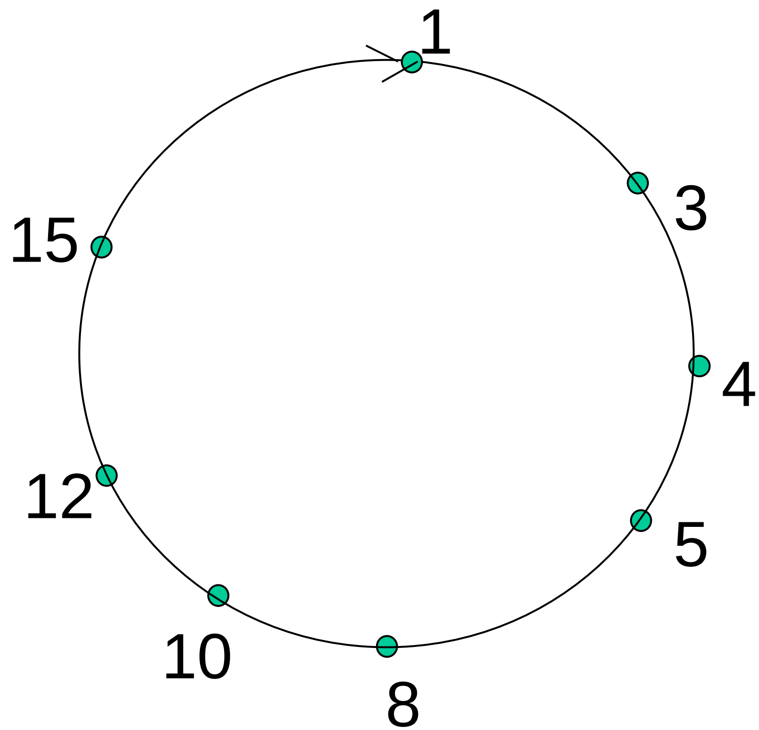
- assign integer identifier to each peer in range $[0,2^n-1]$ for some *n*.
  - each identifier represented by *n* bits.

- 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

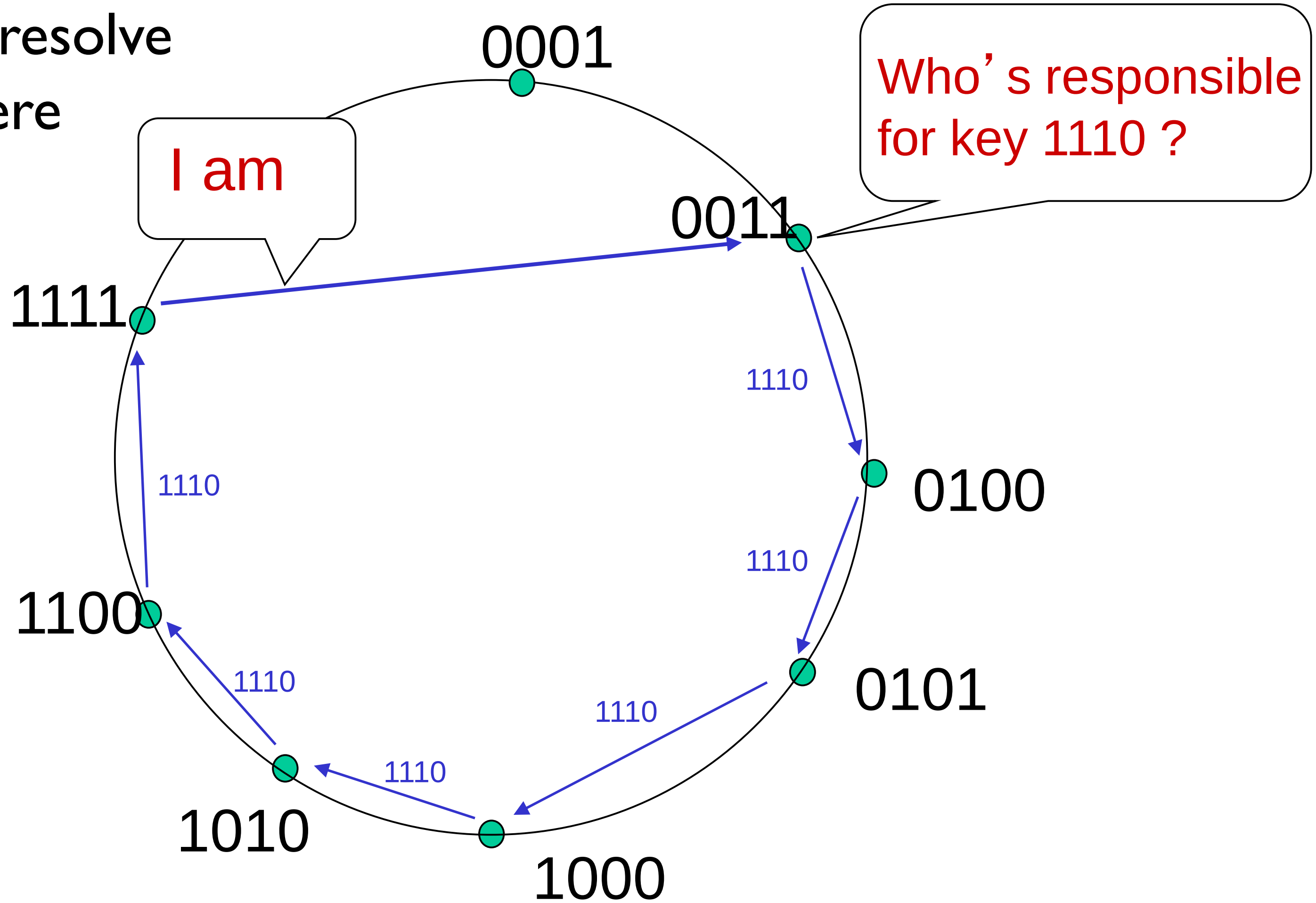
Circular DHT (1)



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

Circular DHT (1)

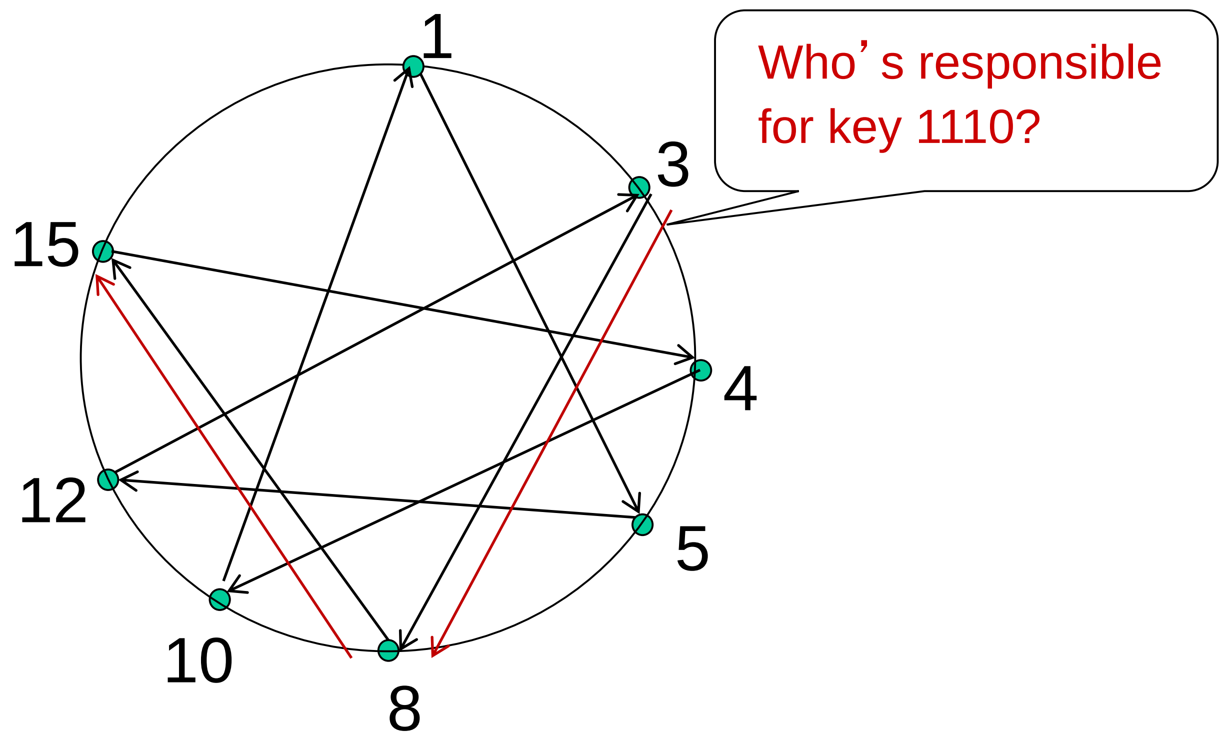
O(N) messages
on avgerage to resolve
query, when there
are *N* peers



Define closest
as closest
successor

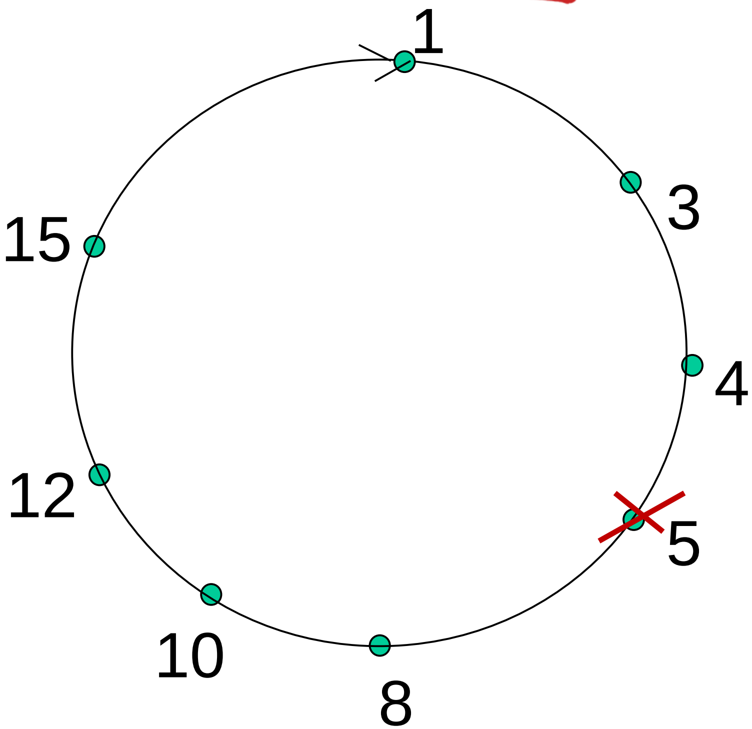
Application 2-54

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



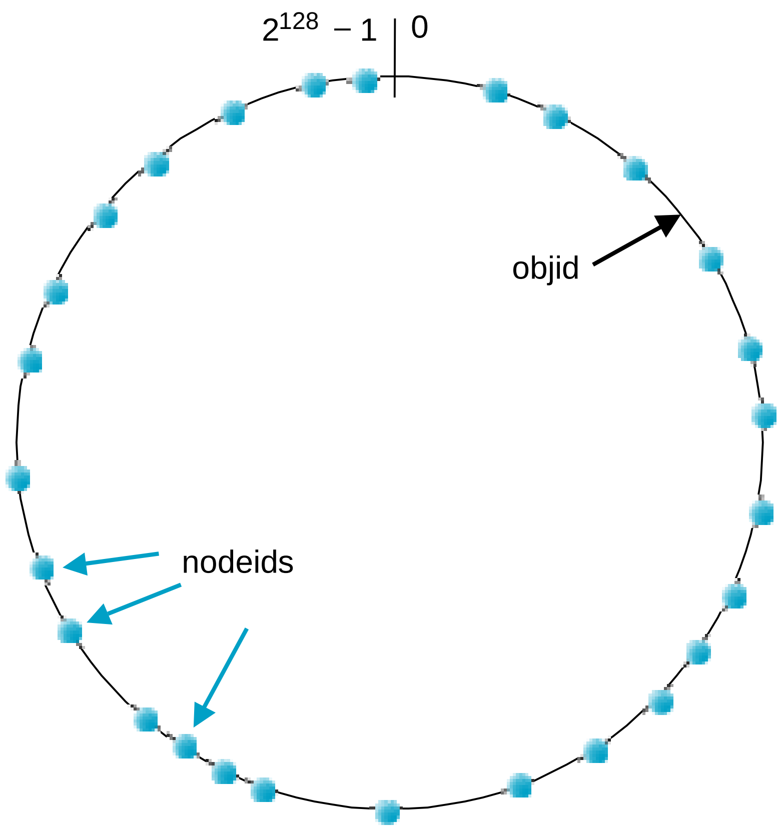
handling peer churn:

- 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?

Object Distribution



Consistent hashing [*Karger et al. '97*]

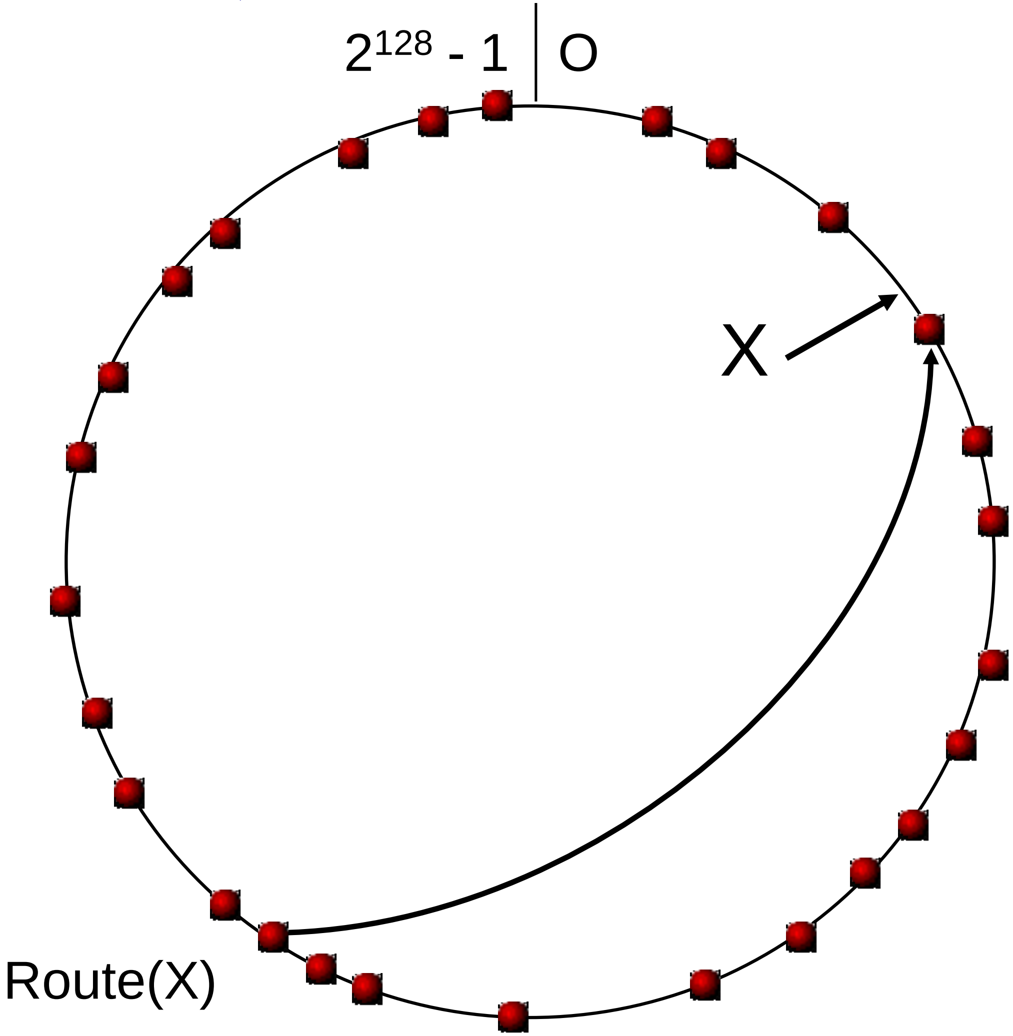
128 bit circular id space

nodeIds (uniform random)

objIds (uniform random)

Invariant: node with numerically closest nodeId maintains object

Object Insertion/Lookup



Msg with key *X* is routed to live node with nodeId closest to X

Problem: complete routing table not feasible

Routing

Integrity of overlay:

- guaranteed unless L/2 simultaneous failures of nodes with adjacent nodeIds

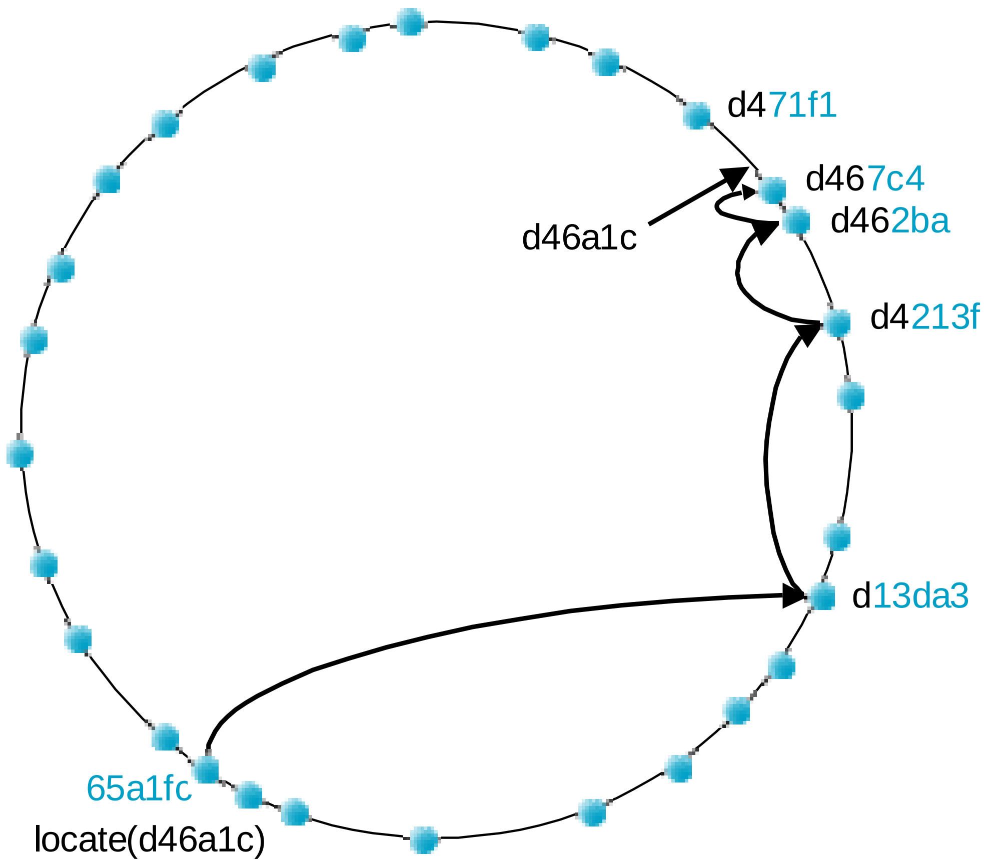
Number of routing hops:

- No failures: < $log_{16} N$ expected, 128/b + 1 max
- During failure recovery:
  - *O(N)* worst case, average case much better

Routing Procedure

if (destination is within range of our leaf set)
  forward to numerically closest member
else
  let *l* = length of shared prefix
  let *d* = value of *l*-th digit in *D*'s address
  if (R_l^d exists)
    forward to R_l^d
  else
    forward to a known node that
    (a) shares at least as long a prefix
    (b) is numerically closer than this node

Routing



Properties

- $\log_{16} N$ steps
- $O(\log N)$ state

DHT Routing Protocols

- DHT is a generic interface

- There are several implementations of this interface
  - Chord [MIT]
  - Pastry [Microsoft Research UK, Rice University]
  - Tapestry [UC Berkeley]
  - Content Addressable Network (CAN) [UC Berkeley]

  - SkipNet [Microsoft Research US, Univ. of Washington]
  - Kademlia [New York University]
  - Viceroy [Israel, UC Berkeley]
  - P-Grid [EPFL Switzerland]
  - Freenet [Ian Clarke]

Decentralized but Structured

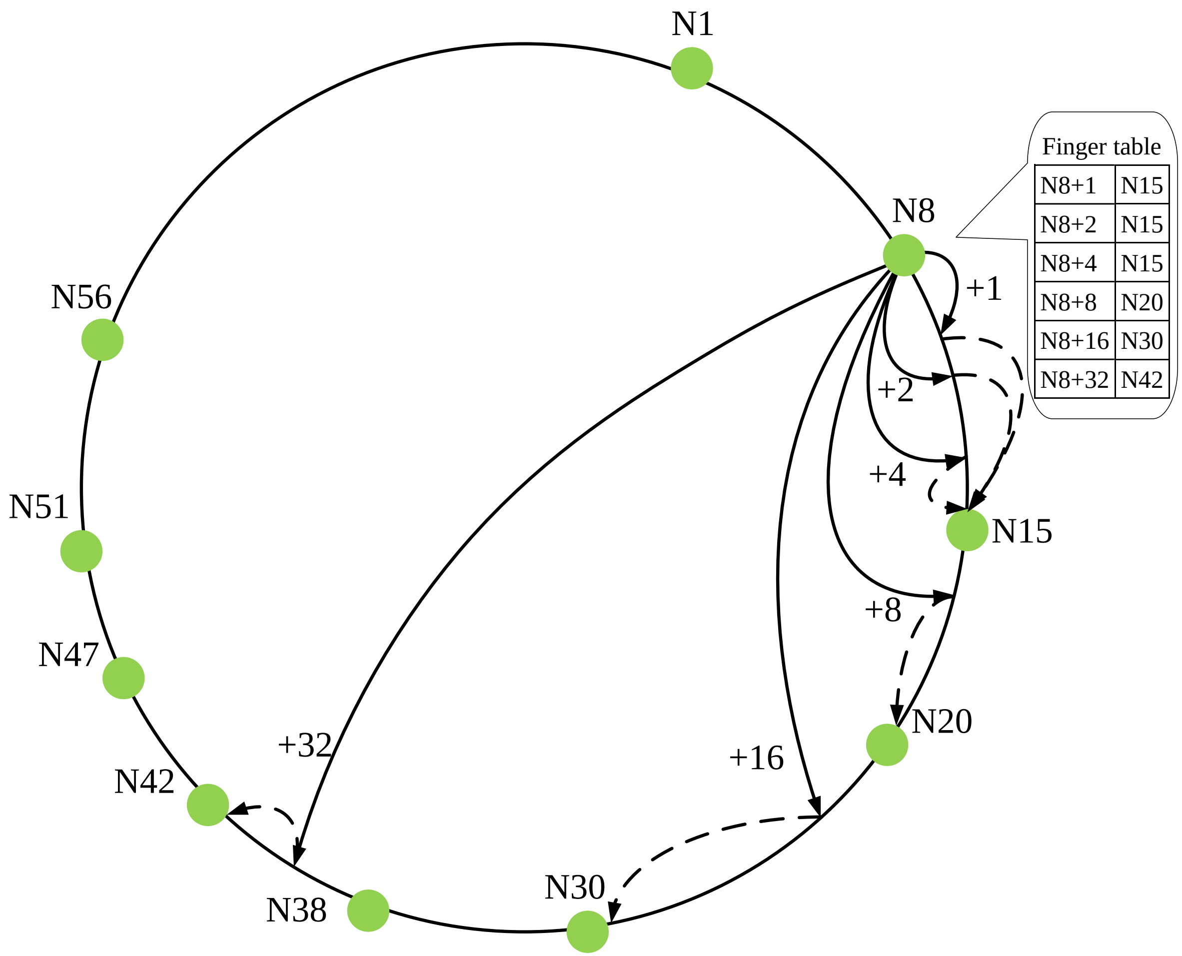
- Message routing (use Chord as an example)
  - Key idea: have each peer maintain a specially designed routing table such that every peer could forward the arriving message to a neighboring peer with node ID that is further closer to the destination.
  - Consider a 10-node Chord overlay in a 6-bit address space
  - Chord views its address space as a one-dimensional circular space such that peers in the space form a ring overlay.

Message Routing in Chord

- The routing table in Chord is called a finger table.
- For an m-bit address space, the finger table of a node with ID=x consists of at most m entries and the i-th entry points to the first node with ID following the ID of $x+2^{i-1}$ modulo 2^m, for $1 \leq i \leq m$.

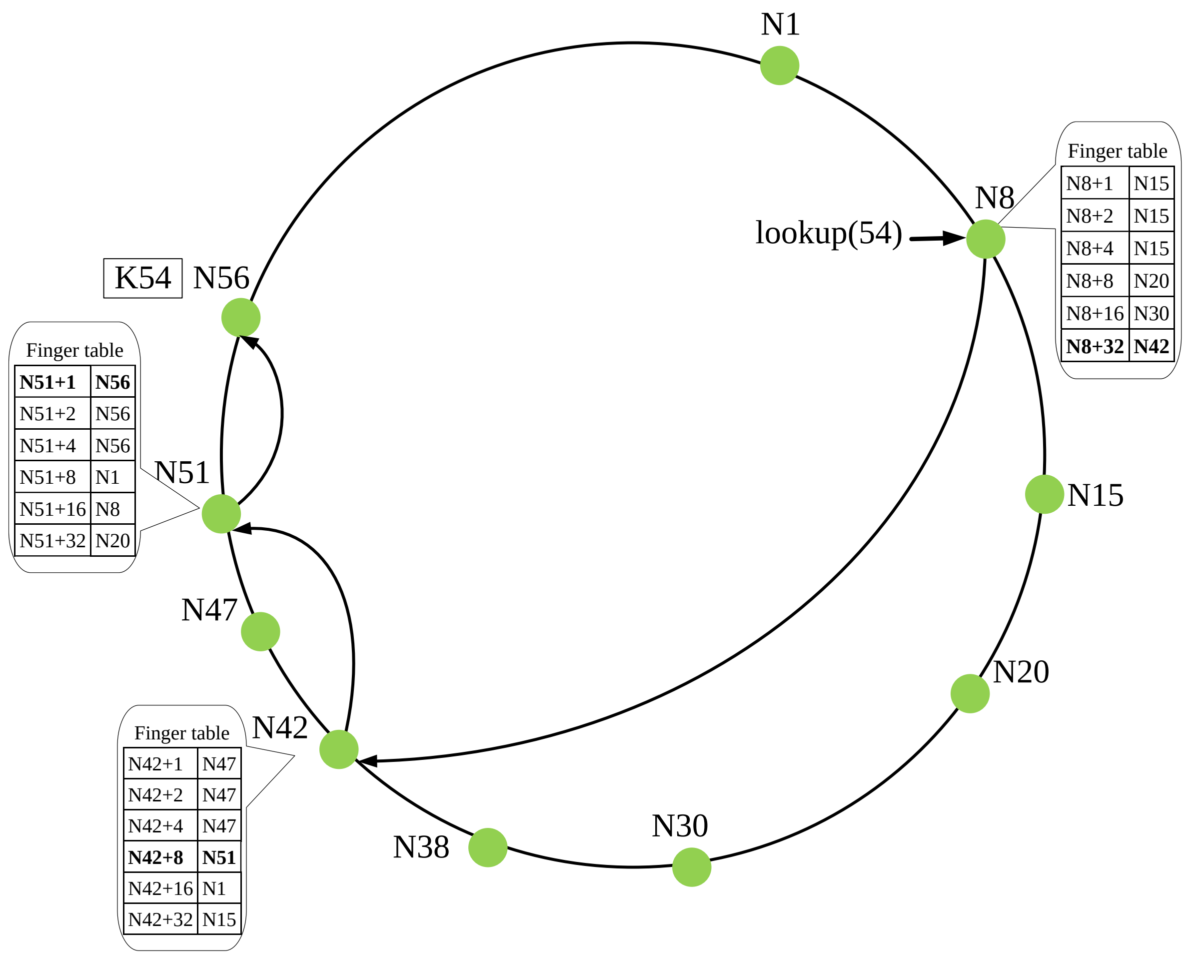
Finger Table of Chord

- Finger table of node *N8*, where *m* =6.

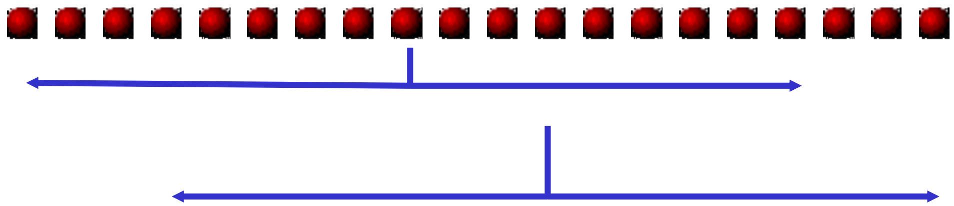


Routing a Query Message

- Routing a query message for object 54 from *N8*



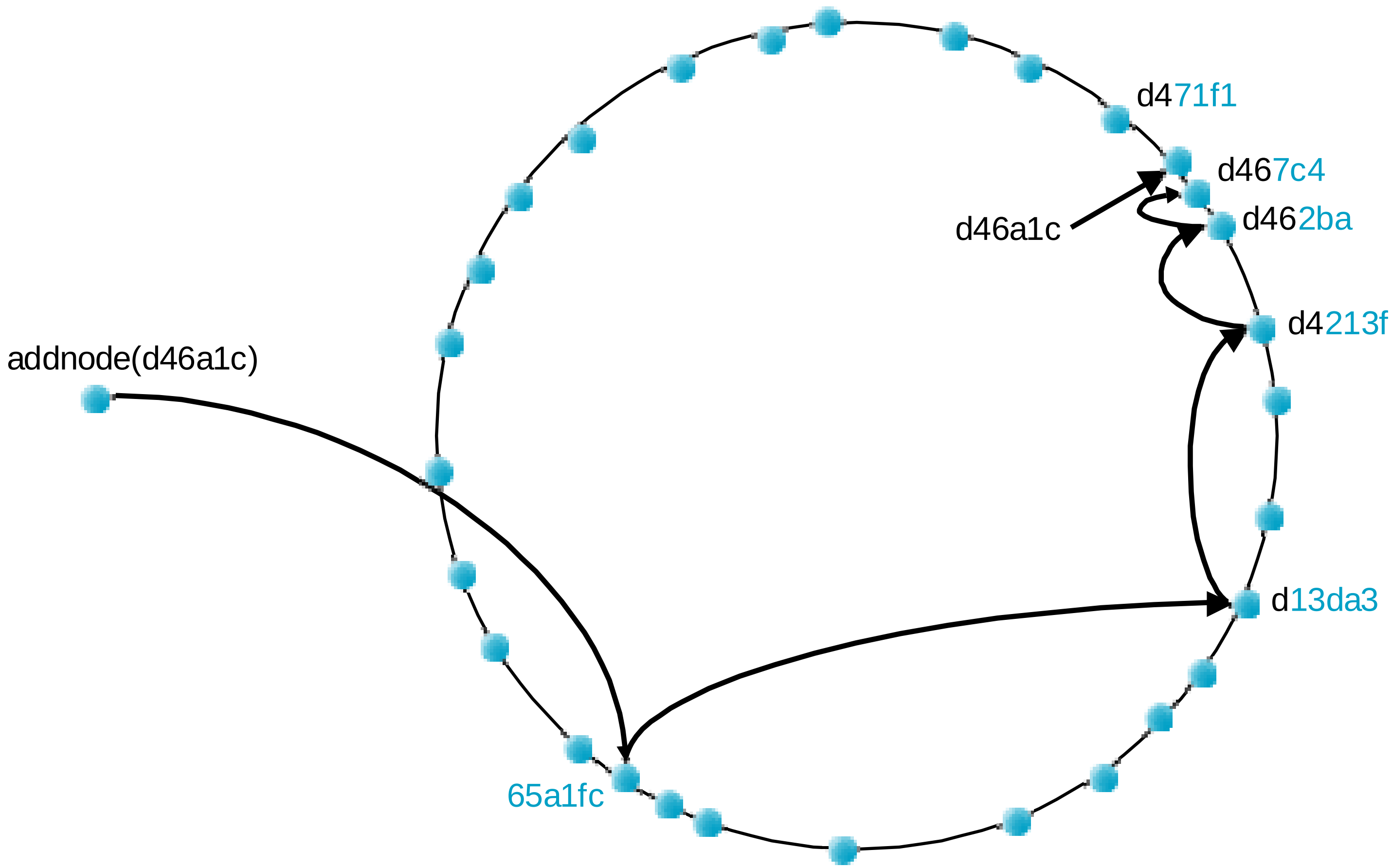
Leaf Sets



Each node maintains IP addresses of the nodes with the L numerically closest larger and smaller nodeIds, respectively.

- routing efficiency/robustness
- fault detection (keep-alive)
- application-specific local coordination

Node Addition



Node Departure (Failure)

Leaf set members exchange keep-alive messages

- **Leaf set repair (eager):** request set from farthest live node in set
- **Routing table repair (lazy):** get table from peers in the same row, then higher rows

Performance Issues of P2P Applications

- Free Riding
- Flash Crowd
- Topology Awareness
- NAT Traversal
- Churn
- Security
- Copyright Infringement

Free Riding

- Scalability of P2P systems relies on the contribution from peers
  - free rider: a peer only consumes but contributes little or no resources
  - 85% of peers share no files in Gnutella in 2005
- A common solution is to implement some incentive mechanisms.
  - tit-for-tat in BitTorrent.
  - reward-based
  - credit-based

Flash Crowd

- Definition: a sudden, unanticipated growth in the demand of a particular object
  - e.g., a new release of a DVD video or mp3 file
- Issues
  - A sudden large amount of query messages
  - To find and download the object within a short time period
- Solutions
  - Cache, duplicating popular objects

Topology Awareness

- A virtual link could be
  - a long end-to-end connection across continents
  - a short one within a local area network
  - How to avoid serious topology mismatch
- Solutions
  - Route-proximity or Neighbor-proximity
  - Routing or neighbor selection based on RTT measurement, preference of routing domain or ISP, or geographical information.

NAT Traversal

- Basic requirement for P2P systems
  - If both peers are behind NAT devices, they cannot connect to each other without help from other peers or STUN servers
- Solutions
  - In most cases, NAT traversal is solved by relay peers or super peers that have public IP addresses

Churn

- Churn refers to the phenomenon that peers dynamically join and leave the system at will.
  - high churn rate seriously affects the stability and scalability of a P2P system.
  - e.g., a high churn rate may cause a tremendous overlay maintenance overhead and dramatic routing performance degradation in DHT-based system
- Solutions
  - Avoid rigid structure or relation among peers
  - Peers maintain a list of potential neighbors for quick and dynamic neighbor replacement

Security

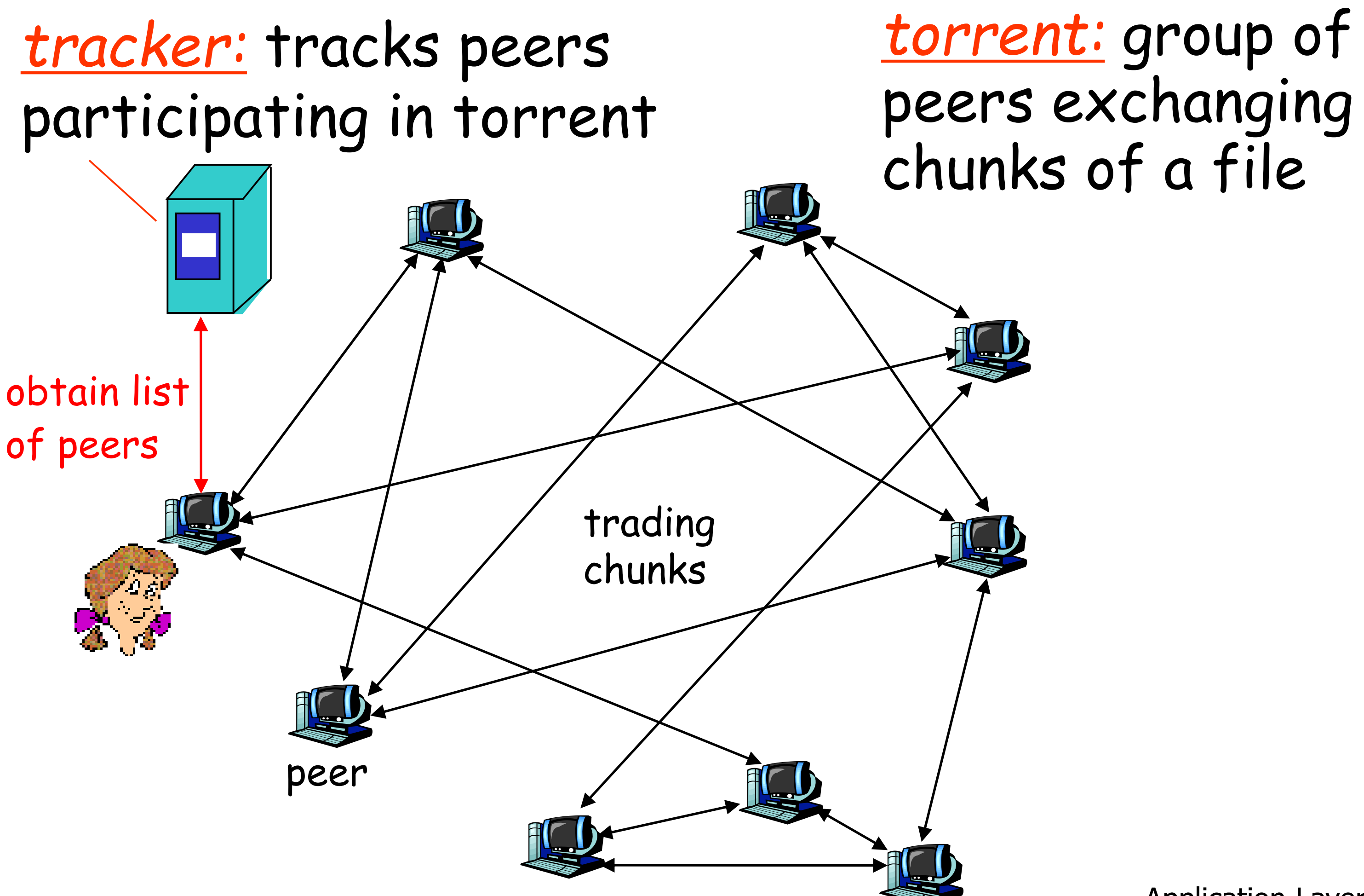
- Issues
  - P2P programs with back hole (Trojan Horse), spurious content, leaking of files not to be shared.
- Solutions to content pollution
  - Protect the content with message digest such as MD5
    - In BitTorrent, the MD5 digest of each piece of a shared file is stored in the metadata file
  - Peer reputation system
  - Object reputation system

Copyright Infringement

- Sharing copyrighted objects through P2P systems is a serious problem which hinders the promotion and usage of P2P systems.
- Not only P2P users are responsible for copyright infringement, so are the companies that host P2P applications
  - Especially in the case where P2P systems will not be able to exist without their servers (e.g., Napster)

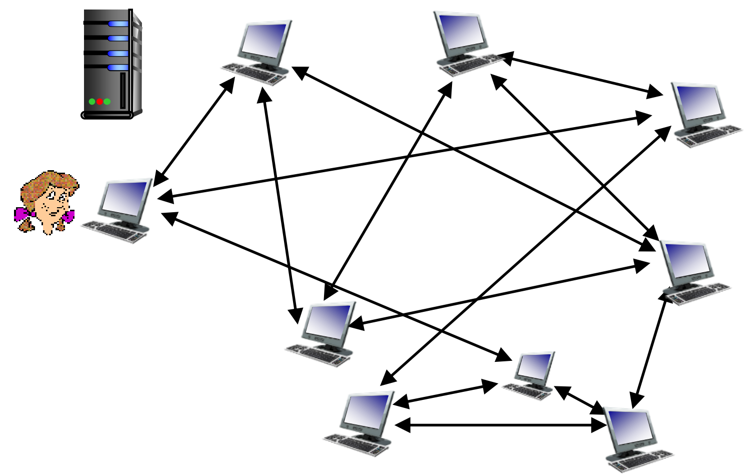
P2P file distribution: BitTorrent

- BitTorrent (BT) was originally designed by Bram Cohen in 2001



P2P file distribution: BitTorrent

- file divided into 256KB *chunks*.
- 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:

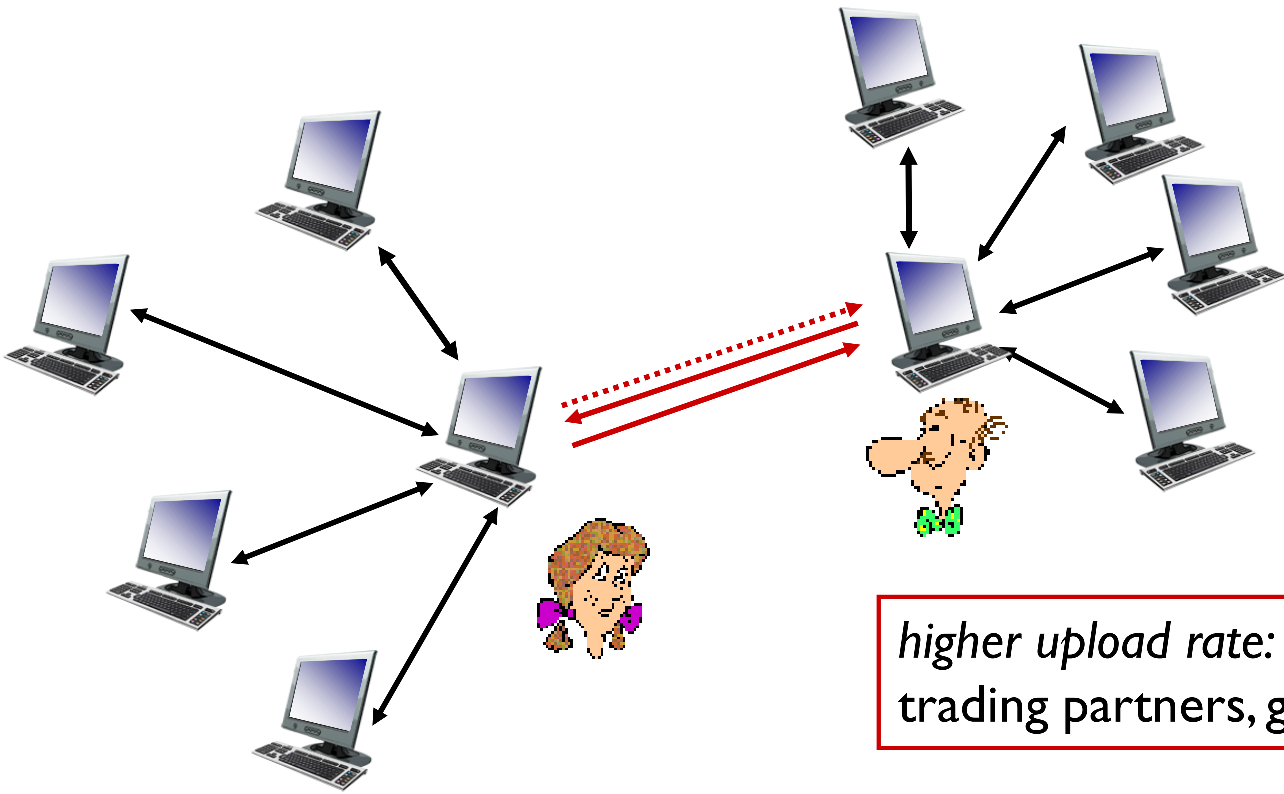
- at any given time, different peers have different subsets of file chunks
- periodically, Alice asks each peer for list of chunks that they have
- Alice requests missing chunks from peers, rarest first

sending chunks: tit-for-tat

- 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

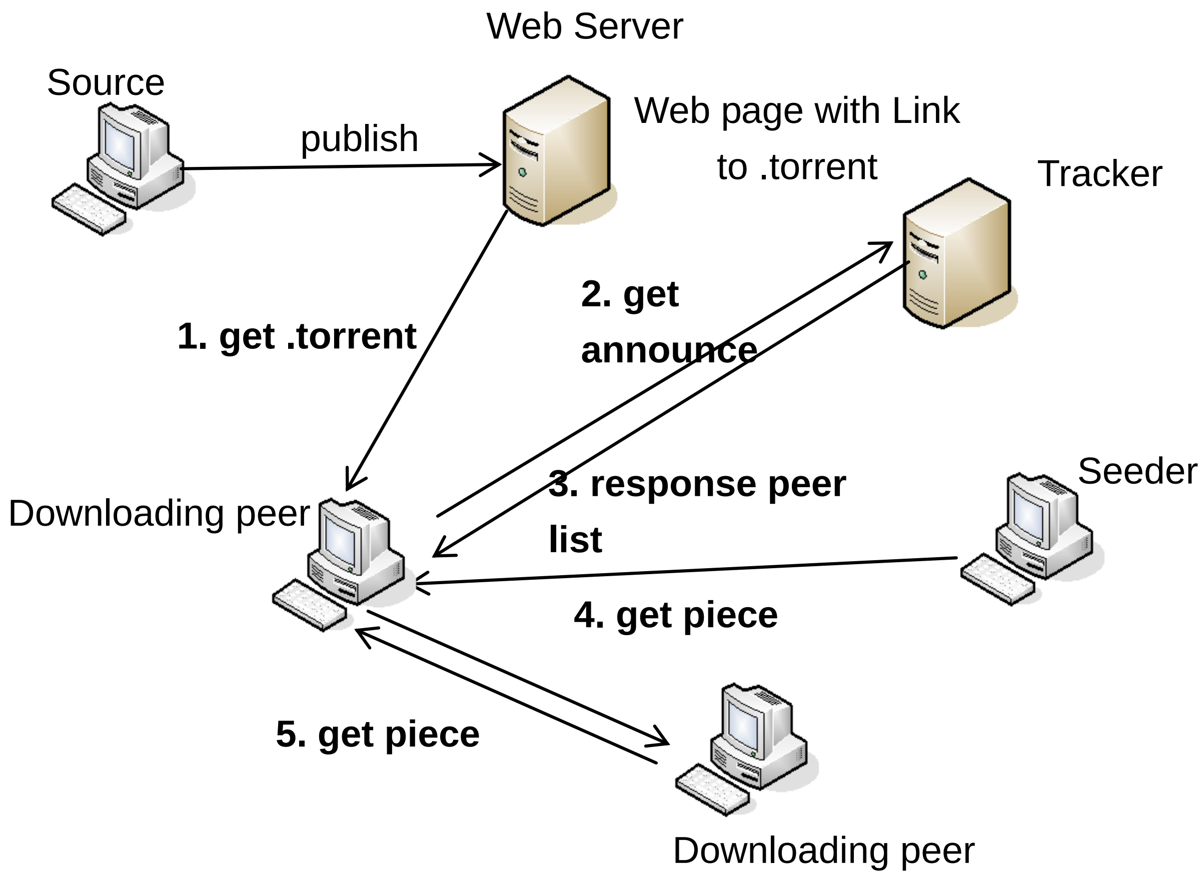
BitTorrent: tit-for-tat

(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



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

BT Operation Overview



BT Architecture

- Hybrid
  - Centralized: tracker plays the role of local central directory server for a file
  - Decentralized: peer discovers which piece to download from which peer/seeder in a distributed manner
  - New development: distributed tracker based on DHT (no centralized tracker)

Piece Selection

- Random first piece selection
  - For the first few pieces, the client just randomly selects a piece to download.
- Rarest first policy
  - Selects the most scarce piece to download first
- End-game mode
  - To speed up the completion of a file download at the end, a peer with only a few pieces missing will send requests for all missing pieces to all the peers

Peer Selection

- Choking/unchoking
  - Choking refers to a temporal refusal to upload to a peer.
  - At the beginning, all peers are chocked
  - Tit-for-tat algorithm selects a fixed number of peers from which the peer downloaded most to unchoke
- Optimistic unchoking
  - new peer needs to move its first step when initially joined the system
  - select one peer at random
- Anti-snubbing
  - If a peer is choked by all of its peers (snubbed), it is better to run optimistic unchoking more often to explore more peers that are willing to cooperate.