Lecture 8: Application Layer P2P Applications and DHTs

COMP 332, Spring 2018
Victoria Manfredi

Acknowledgements: materials adapted from Computer Networking: A Top Down Approach 7th edition: ©1996-2016, J.F Kurose and K.W. Ross, All Rights Reserved as well as from slides by Abraham Matta at Boston University, slides by Daniel Zappala at Brigham and Young University, and some material from Computer Networks by Tannenbaum and Wetherall.

Today

1. Announcements
   - hwk 3 due today at 11:59p, hwk 4 posted this evening
     - solution code for homework 4 will be posted once homework3 turned in
   - hopefully will have hw1 and hw2 graded to give back to you tomorrow
     - Will post on piazza and leave outside my office
2. Peer-to-peer applications
3. BitTorrent
4. Distributed hash tables

P2P Applications
OVERVIEW

Pure P2P architecture

How can an ordinary person, with limited money and bandwidth, serve content to a worldwide audience?

No always-on server

- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses

Examples

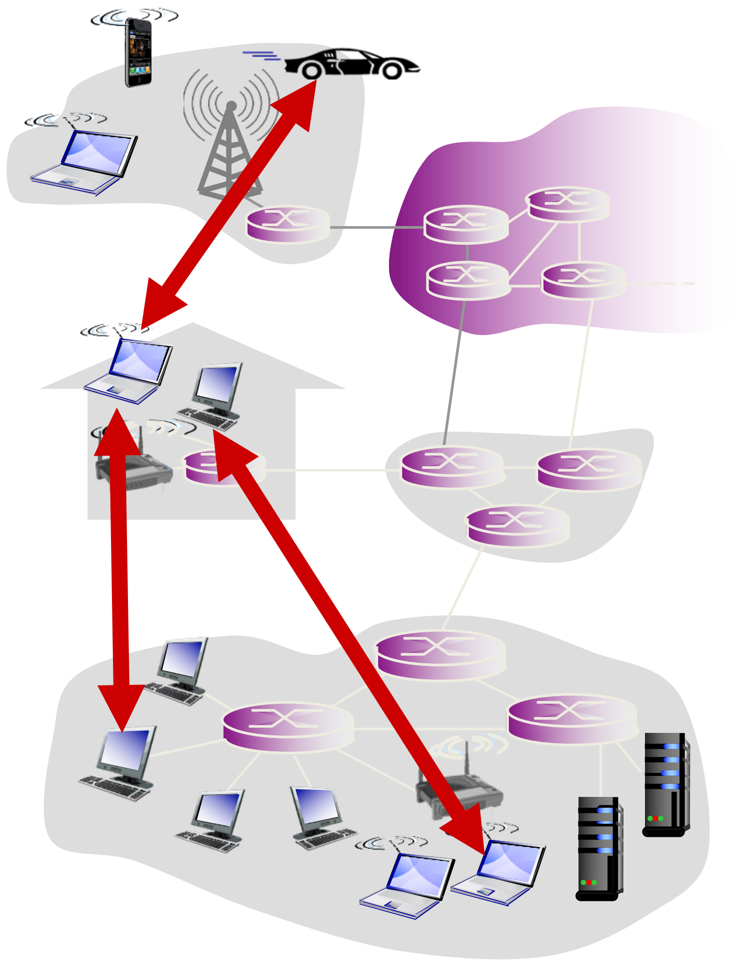
- file distribution (BitTorrent)
- VoIP (Skype)

Pros: highly scalable!

- since all peers are servers

Cons: difficult to manage

- how do you find peers and content?



Full and half-duplex links

Full duplex

- can upload and download at same time
- e.g, wired links, telephone line

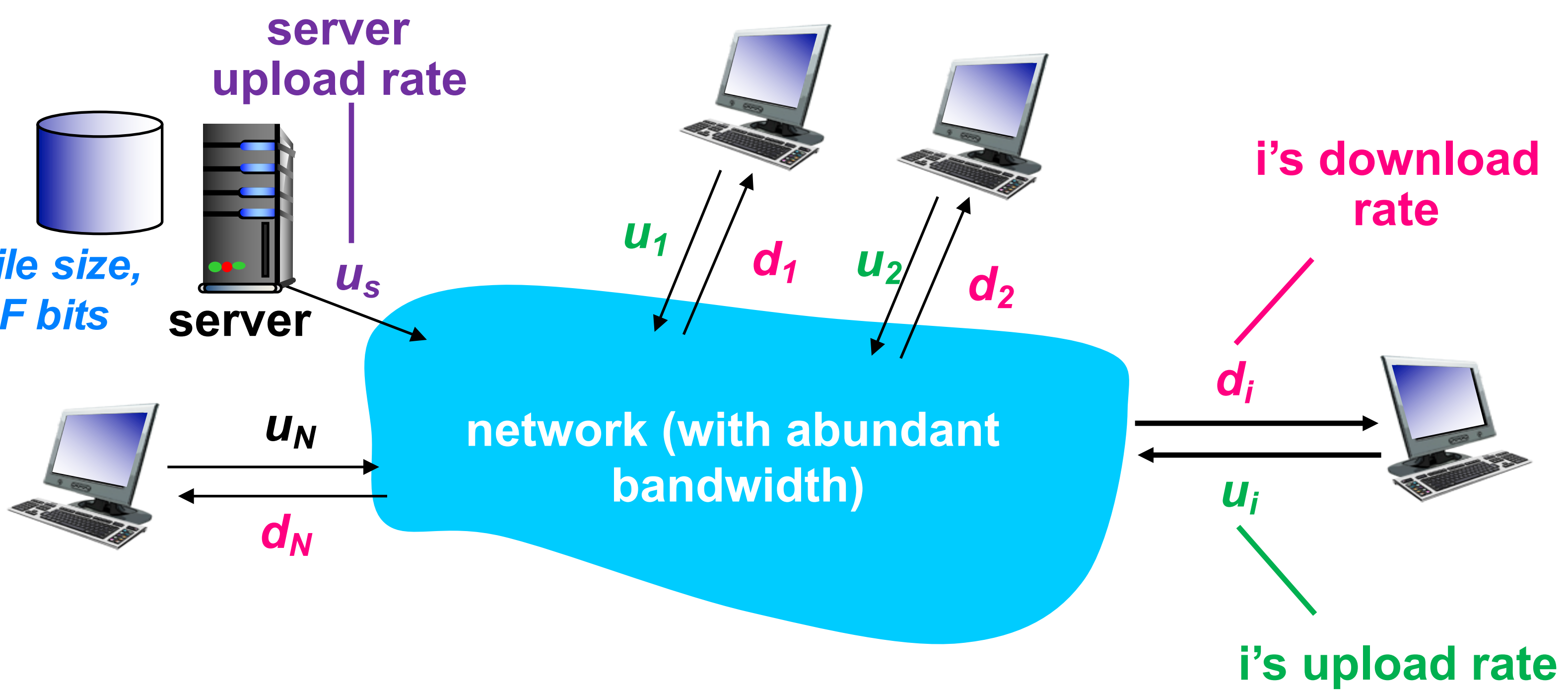
Half-duplex links

- cannot upload and download simultaneously
- e.g., wireless links, walkie-talkies

File distribution time

Question

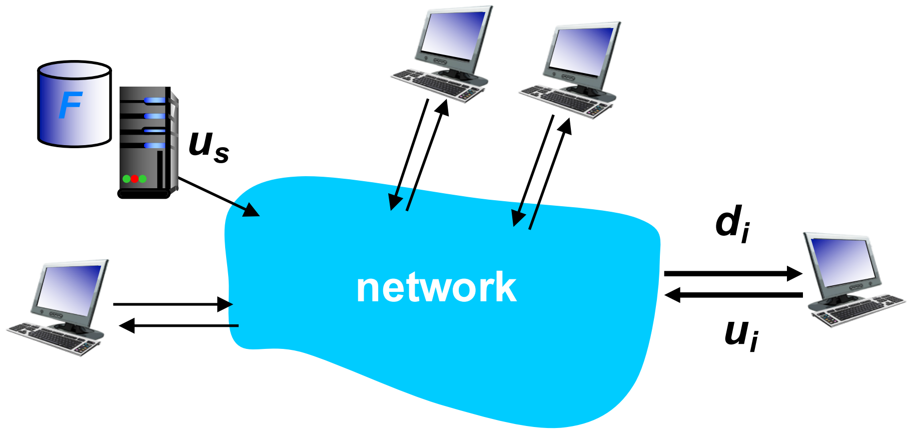
- how much time to distribute file (size F) from one server to N peers?
- peer upload/download capacity is limited resource



Client-server file distribution time

Server transmission

- sequentially sends (uploads) N file copies
  - time to send one copy: F/u_s
  - time to send N copies: NF/u_s



Client

- each client downloads file copy
  - d_{min} = slowest client download rate
  - slowest client download time: F/d_{min}

Server is bottleneck

Slowest client is bottleneck

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

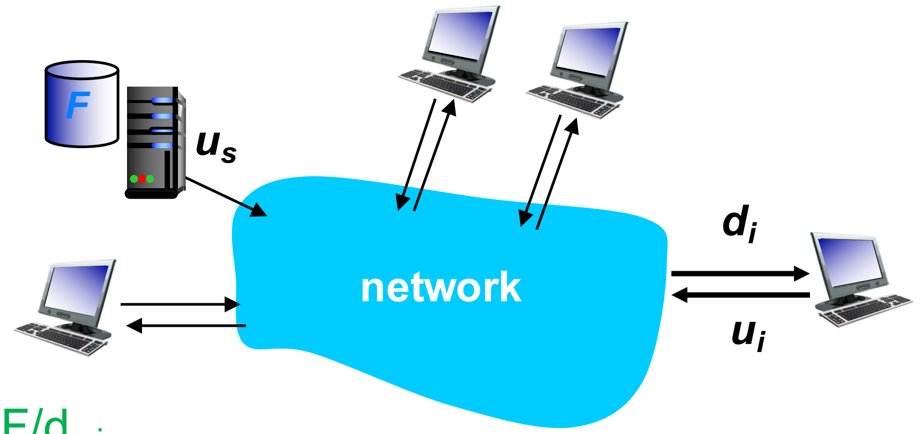
$$D_{cs} \geq \max\{NF/u_s, F/d_{min}\}$$

increases linearly in N (for large N)

P2P file distribution time

Server transmission

- must upload at least one copy
- time to send one copy: F/u_s



Client

- each client downloads copy
  - slowest client download time: F/d_{min}
- as aggregate must download *NF* bits
  - max upload rate (limiting max download rate): $u_s + \Sigma u_i$

Server is bottleneck

Slowest peer's download rate is bottleneck

Upload rate is bottleneck

Time to distribute F to N clients using P2P approach

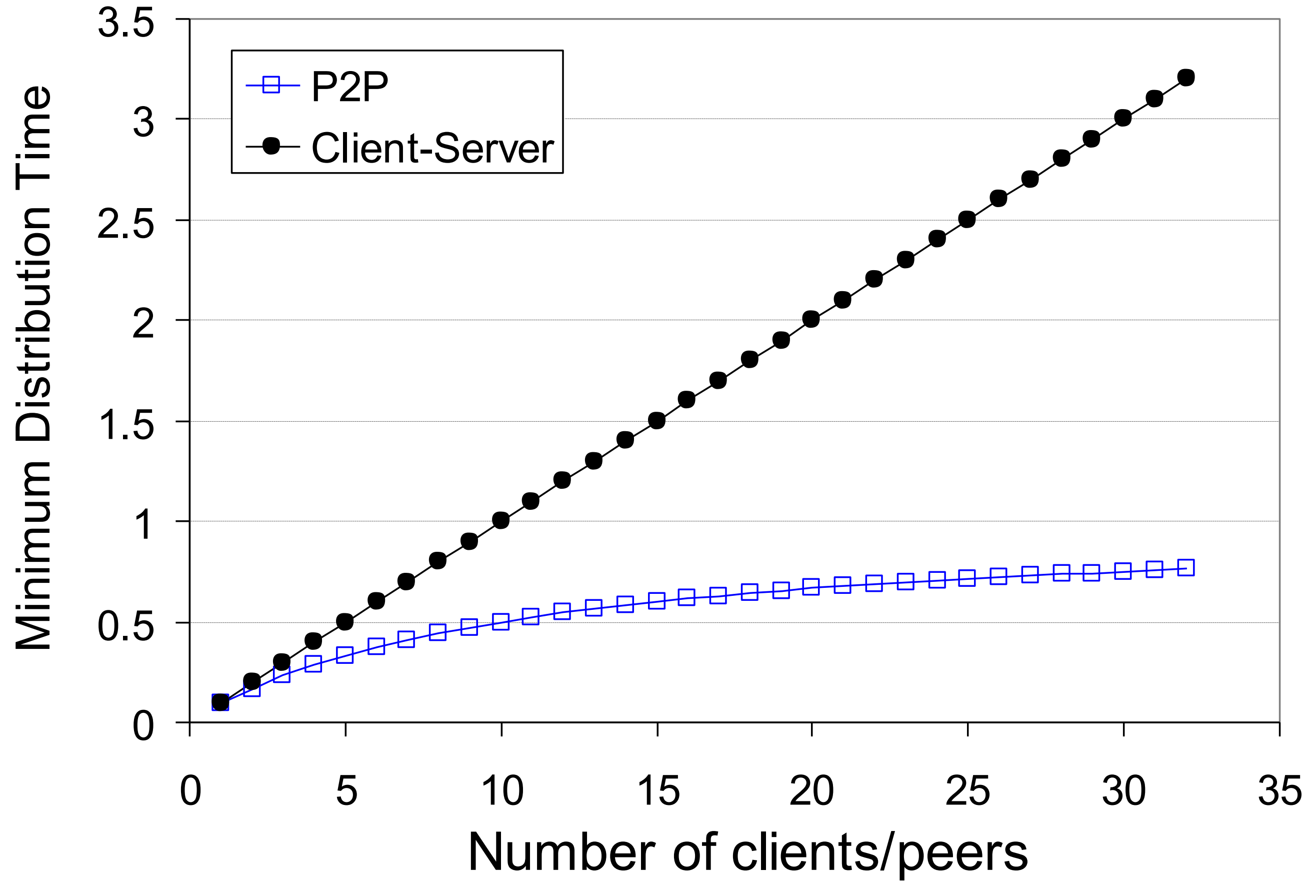
$$D_{P2P} \geq \max\{F/u_s,\ F/d_{min},\ NF/(u_s + \Sigma u_i)\}$$

increases linearly in *N* …

… but so does this, as each peer brings service capacity

Client-server vs. P2P: example

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



P2P download is self-scaling: the more peers that download, the more bandwidth is available for upload

P2P takeaways

Self-scaling

- more peers means more bandwidth

Robustness

- if a peer fails, another peer can do the same job
  - harder to set up another server

Better privacy

- no single provider able to monitoring entire system

P2P Applications

BITTORRENT

BitTorrent Protocol

P2P file sharing protocol

- developed by Bram Cohen in 2001 (www.bittorrent.org)
- 2013: responsible for ~3% of all bandwidth used worldwide
- break files into chunks, download chunks from different users
  - makes better use of bandwidth than downloading from one user
  - upload bandwidth constrains download rate and is often less than download bandwidth

BitTorrent tracker (server): TCP, port 6969

- keeps track of which peers are currently downloading which files
- assists with making download more efficient

BitTorrent client: TCP, port 6881-6889

- perform p2p file sharing using BitTorrent protocol
- BitTorrent client is no longer open source but still free
- many free open source clients
  - e.g., qBitTorrent: https://github.com/qbittorrent/qBittorrent

Publishing content

1. Create "torrent" and distribute
   - metainfo file derived from content you want to publish
     - includes filename, file size, hash info, tracker info
     - anyone who wants to download content needs .torrent file
   - can typically create using BitTorrent client

2. Start a downloader called a "seed"
   - peer that has entire file, must upload at least one complete copy

Downloading content

1. Bootstrapping
   - download .torrent file from web server
   - contact listed tracker for list of peers

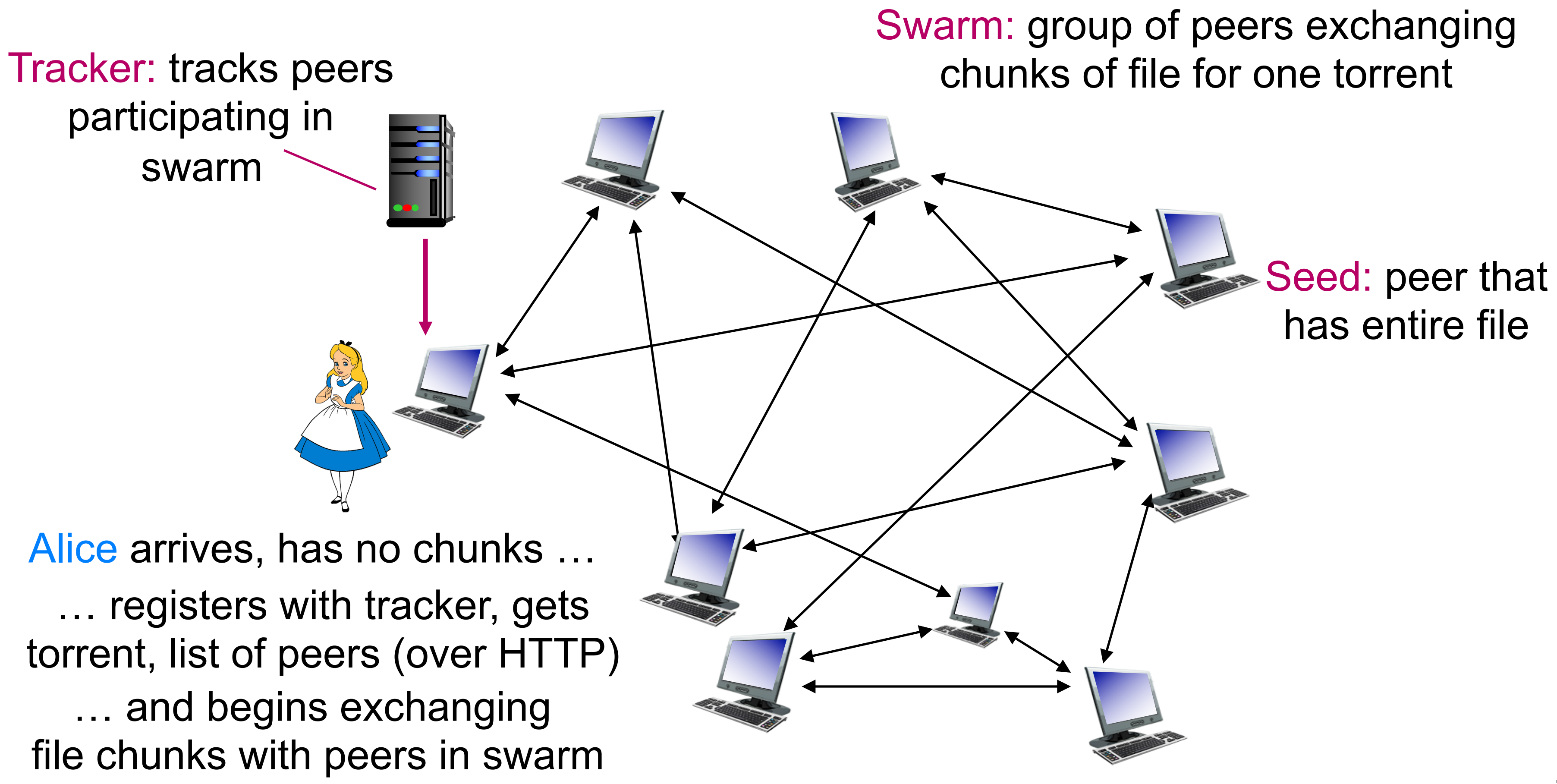
2. Peer discovery
   - periodically contact tracker if current peers don't have content you need or they leave the system

3. Content location
   - check with each peer to determine which blocks they have
   - download rarest blocks first

BitTorrent file distribution

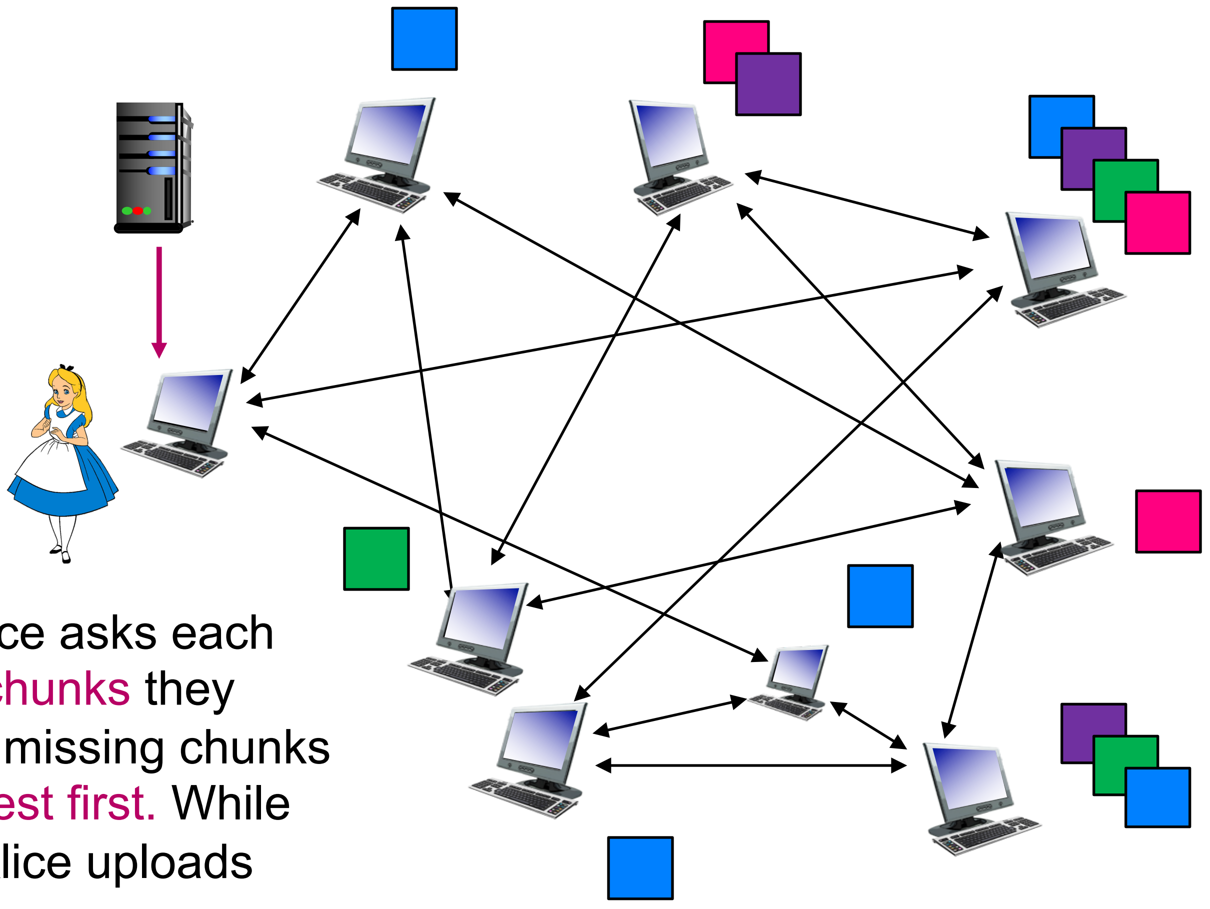
File divided into 256Kb chunks

- peers in swarm send/receive file chunks
- Q: why 256 Kb chunks?


Tracker: tracks peers participating in swarm
Swarm: group of peers exchanging chunks of file for one torrent
Seed: peer that has entire file
Alice arrives, has no chunks …
… registers with tracker, gets torrent, list of peers (over HTTP)
… and begins exchanging file chunks with peers in swarm

Requesting (downloading) chunks

At any given time, different peers have different subsets of file chunks



Periodically, Alice asks each peer for list of chunks they have, requests missing chunks from peers, rarest first. While downloading, Alice uploads chunks to other peers

Problem 1

Freeloaders

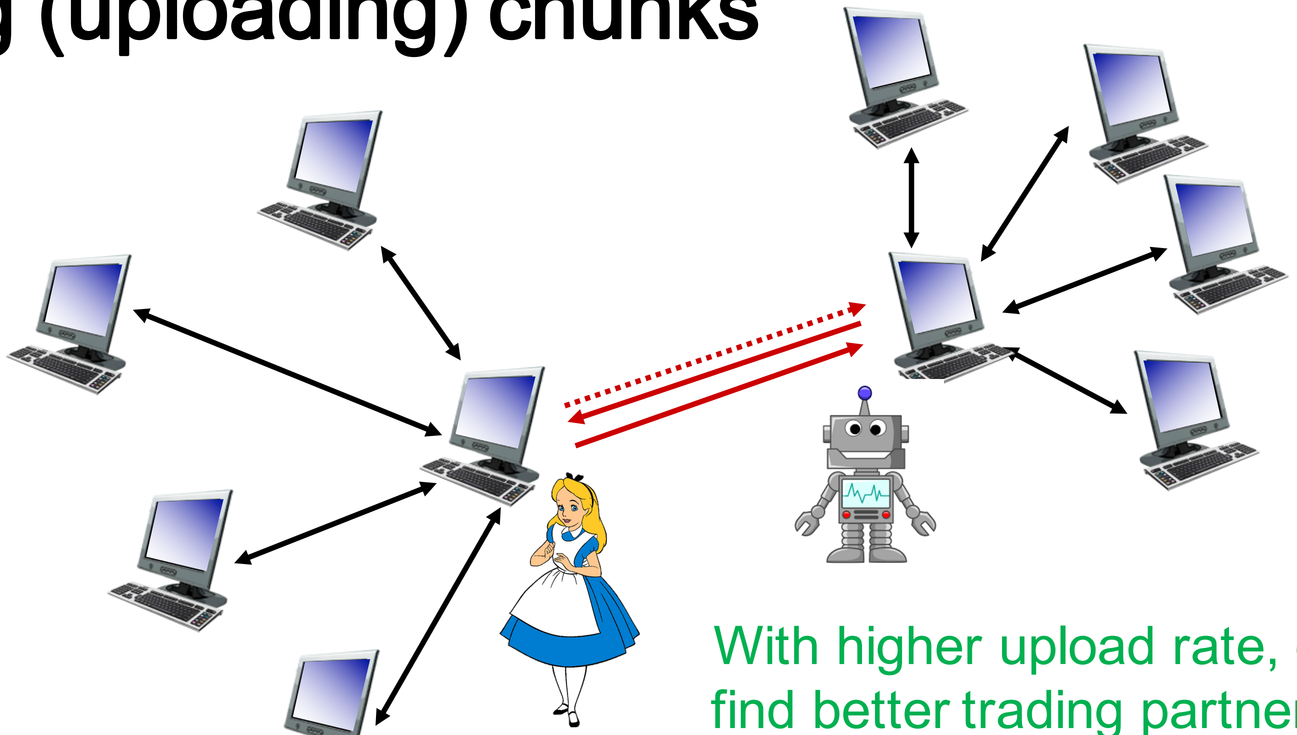
- peers who download without uploading
- p2p system is only self-scaling if every peer adds resources
  - users who download without uploading
  - break self-scaling behavior of p2p file distribution

Q: how do peers encourage each other to upload content as well as download content for themselves?

Solution: Tit-for-tat

- serve content to k connections at a time
- serve connections that give you best download rate
- periodically serve content to a random connection to see if it can do better than a current connection
- deny content to all others

Sending (uploading) chunks



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

Tit-for-tat

- Alice sends chunks to 4 peers sending her chunks at highest rate
  - other peers are choked by Alice (do not receive chunks from her)
  - re-evaluate top 4 every 10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - "optimistically unchoke" this peer, newly chosen peer may join top 4

Problem 2

Churn: peers may come and go

- once peer has entire file
  - it may (selfishly) leave
  - or (altruistically) remain in swarm

Consequence

- peer may change peers with whom it exchanges chunks
- need mechanism to locate peers with needed chunks even as churn occurs

Q: How does a peer find other peers that have content it wants to download?

- not all peers have all content
- every content provider creates a content description called a torrent

P2P search solutions

Centralized directory

- Napster: led to its demise due to copyright infringement
  - centralized solution makes it easy to find people to sue
- BitTorrent (and centralized tracker)

Decentralized directory

- Gnutella
  - flood queries, bound by number of hops
    - leads to overload on peers
    - hard to find unpopular files
- BitTorrent (and decentralized tracker)
  - distributed hash table: effectively each peer is a tracker
  - peers have storage to share, connect them in a structured network
  - map each file to a particular peer, easy to find any file if you know name
  - run over UDP, ports negotiated

P2P Applications

DISTRIBUTED HASH TABLE

How to distribute files across peers?

Hash table

- central coordination
- stores (key, value) pairs
  - key: ss number; value: human name
  - key: content type; value: IP address
- hash key to efficiently determine location of (key, value) pair

H(key) → location

Distributed Hash Table (DHT): invented in 2001

- no central coordination: distributed p2p database
- hash table where (key, value) pairs are distributed across devices
  - need mechanisms to
    - assign pairs to devices
    - efficiently determine device location of pair
    - cope with devices joining and leaving

Assigning (key, value) pairs to devices in DHT

Each peer chooses random integer identifier in range $[0, 2^m-1]$

- each identifier represented by n bits

Peer → integer identifier in range $[0, 2^m-1]$

Require each key to be integer in same range

- to get integer keys, hash original key
  - e.g., integer key = h("Pink Floyd")
  - why called a distributed "hash" table

H(filename) → integer key in range $[0, 2^m-1]$

Assigning (key, value) pairs to devices in DHT

Assign (key, value) pair to peer with closest integer identifier

- assume closest is immediate successor of key

Example

- m = 4 implies integer identifiers are in range [0:2^4-1]
    - peers: 1,3,4,5,8,10,12,15
- key = 14
    - immediate successor peer is 15
- key = 15
    - immediate successor peer is 1

Circular DHT

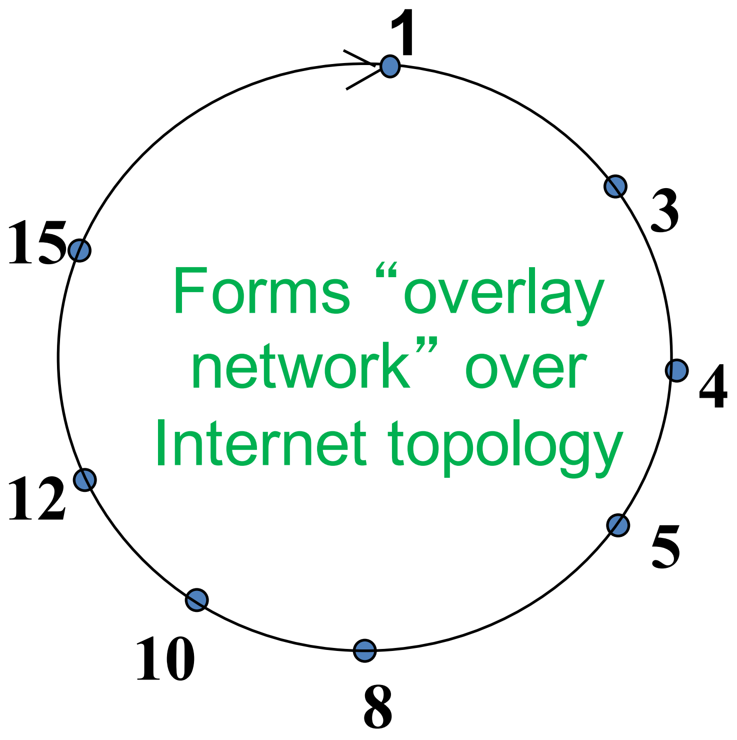
Form a ring of peers

- use SHA-1 to hash node's IP address into m-bit identifier
- peers store pointers to predecessor and successor peers on ring

Store keys on peers

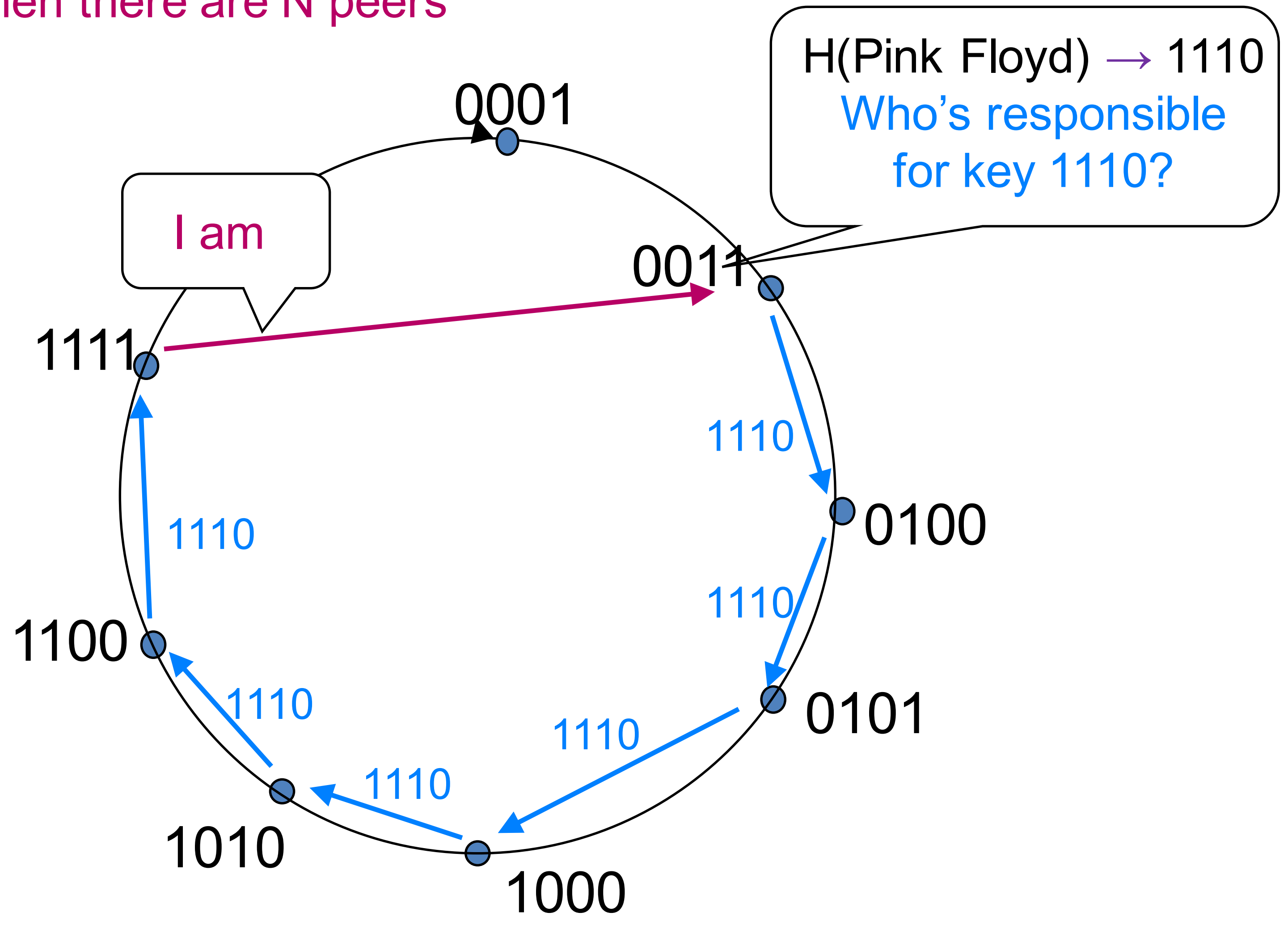
- hash filename into m-bit key, k
- store file on successor(k)
    - 1st peer whose identifier is ≥ k

File search: navigate from current peer to peer storing key k



Searching a circular DHT

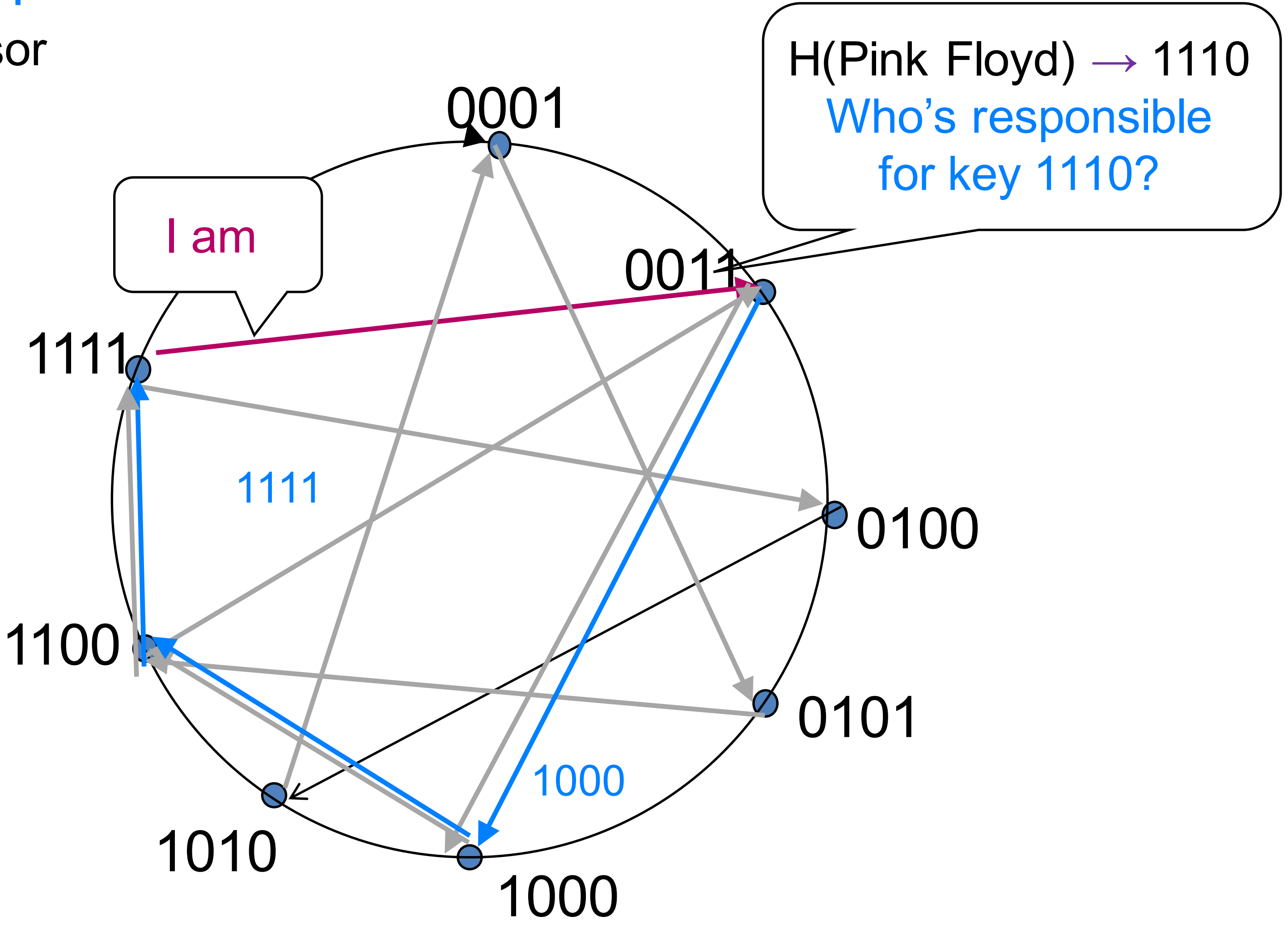
O(N) messages on average to resolve query, when there are N peers



Circular DHT with shortcuts

Each peer keeps track of IP addresses of

- predecessor
- successor
- shortcuts



Circular DHT with shortcuts

Shortcuts

- design so O(log N) neighbors and O(log N) messages in query
- How? Finger table
  - kind of routing table, lets peer jump at least halfway to target

Finger table at peer with identifier k

- m entries in table, each pointing to different peer
- each entry i has 2 fields
  - start = $k + 2^i \pmod{2^m}$
  - IP address of successor(start[i])

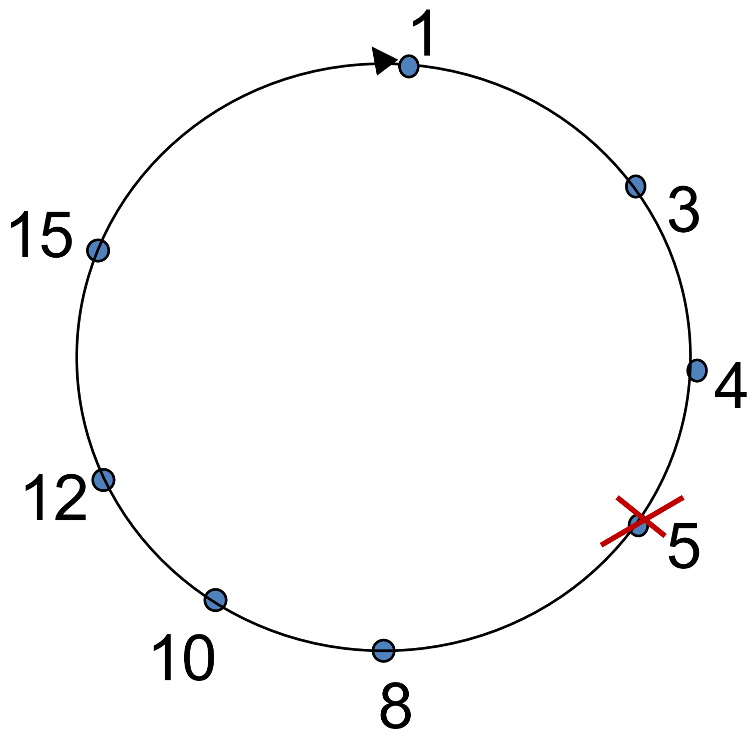
Usage

- if key between k and successor (k)
  - successor(k) peer is holding info, search terminates
- else query finger table
  - find entry whose start field is closest predecessor of key

Peer Churn

How to handle?

- require each peer to know the IP address of its two successors
- each peer periodically pings its two successors to see if they are still alive



Peer 5 abruptly leaves

What happens? Peer 4 detects

- makes 8 its immediate successor
- asks 8 who its immediate successor is
- makes 8's immediate successor its second successor

Q: what if peer 13 wants to join?

Joining DHT

Relies on consistent hashing or efficiency

- uniformly distribute keys and identifiers
  - negligible collision probability
- peers can join and leave with minimal disruption to DHT

Steps

1. Choose random integer identifier in range $[0,2^m-1]$
2. Use lookup function to determine identifier and ring location
   - determine future successor
   - determine predecessor from successor
3. Let successors and predecessors know that you joined

DHT summary

Pros

- fault tolerant, scalable, decentralized, provable guarantees

Cons

- more complex to manage
- additional communication overhead to manage DHT

Protocols

- DHT is not just a data structure
  - since distributed, need communication (managed by protocol)
- Chord, Pastry, Tapestry, Kademlia …
- https://en.wikipedia.org/wiki/Chord_(peer-to-peer)
  - what we overviewed
- https://en.wikipedia.org/wiki/Kademlia