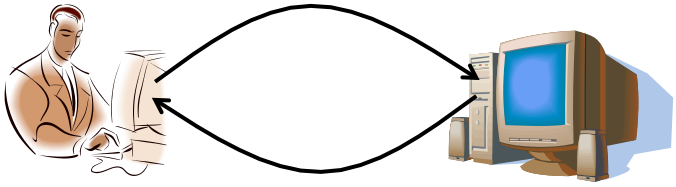
Peer-to-Peer Systems

Peer-to-Peer Systems

- Quickly grown in popularity:
  - Dozens or hundreds of file sharing applications
  - In 2004:
    - 35 million adults used P2P networks – 29% of all Internet users in USA
    - BitTorrent: a few million users at any given point
    - 35% of Internet traffic is from BitTorrent
  - Upset the music industry, drawn college students, web developers, recording artists and universities into court
- But P2P is not new and is probably here to stay
- P2P is simply the next iteration of scalable distributed systems

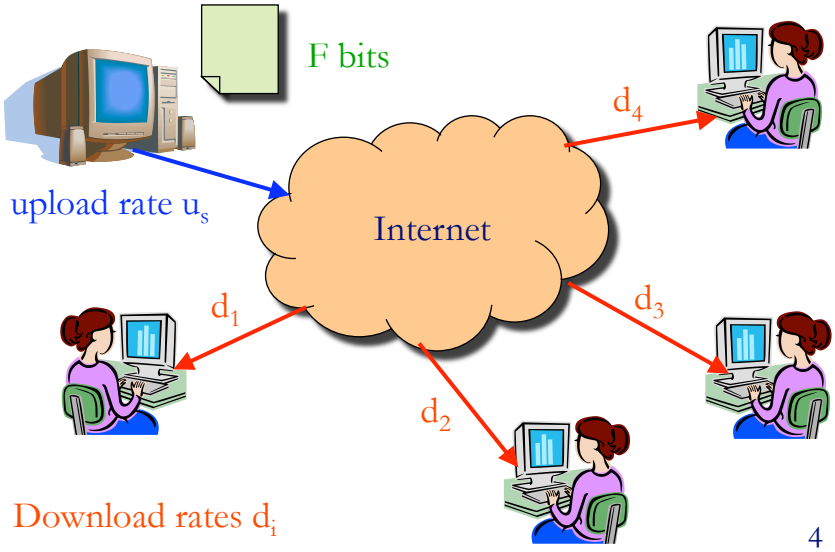
Client-Server Communication

- Client "sometimes on"
  - Initiates a request to the server when interested
  - E.g., Web browser on your laptop or cell phone
  - Doesn't communicate directly with other clients
  - Needs to know the server's address
- Server is "always on"
  - Services requests from many client hosts
  - E.g., Web server for the www.cnn.com Web site
  - Doesn't initiate contact with the clients
  - Needs a fixed, well-known address



3

Server Distributing a Large File



4

Server Distributing a Large File

- Server sending a large file to N receivers
  - Large file with F bits
  - Single server with upload rate u_s
  - Download rate d_i for receiver i
- Server transmission to N receivers
  - Server needs to transmit NF bits
  - Takes at least NF/u_s time
- Receiving the data
  - Slowest receiver receives at rate $d_{min} = min_i\{d_i\}$
  - Takes at least F/d_{min} time
- Download time: $max\{NF/u_s, F/d_{min}\}$

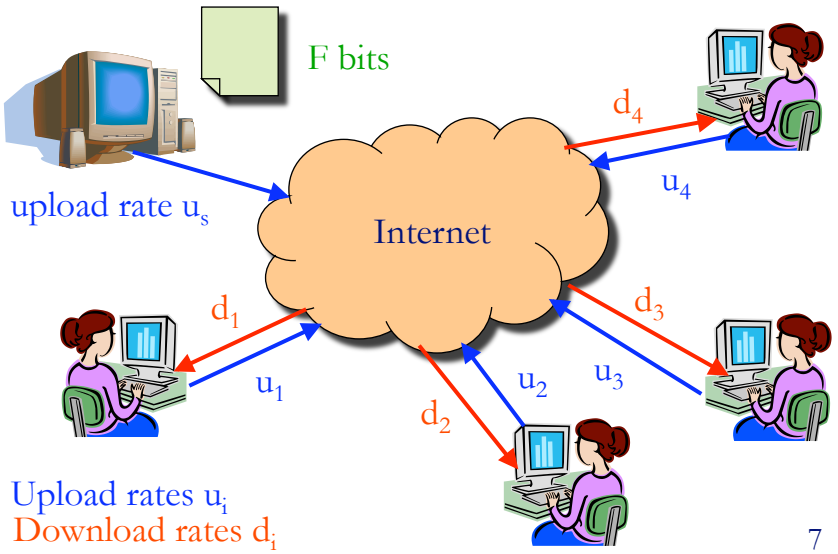
5

Speeding Up the File Distribution

- Increase the upload rate from the server
  - Higher link bandwidth at the one server
  - Multiple servers, each with their own link
  - Requires deploying more infrastructure
- Alternative: have the receivers help
  - Receivers get a copy of the data
  - And then redistribute the data to other receivers
  - To reduce the burden on the server

6

Peers Help Distributing a Large File



7

Peers Help Distributing a Large File

- Start with a single copy of a large file
  - Large file with *F* bits and server upload rate u_s
  - Peer *i* with download rate d_i and upload rate u_i
- Two components of distribution latency
  - Server must send each bit: min time F/u_s
  - Slowest peer receives each bit: min time F/d_{min}
- Total upload time using all upload resources
  - Total number of bits: NF
  - Total upload bandwidth $u_s + sum_i(u_i)$
- Total: $max\{F/u_s, F/d_{min}, NF/(u_s+sum_i(u_i))\}$

8

Comparing the Two Models

- Download time
  - Client-server: $max\{NF/u_s, F/d_{min}\}$
  - Peer-to-peer: $max\{F/u_s, F/d_{min}, NF/(u_s+sum_i(u_i))\}$
- Peer-to-peer is self-scaling
  - Much lower demands on server bandwidth
  - Distribution time grows only slowly with *N*
- But…
  - Peers may come and go
  - Peers need to find each other
  - Peers need to be willing to help each other

9

P2P vs. Youtube

- Let's compare BitTorrent vs. Youtube
- Capacity to accept and store content:
  - Youtube currently accepts 200K videos per day (or about 1TB)
  - 1000 TV channels producing 1Mb/s translates to about 10TB per day

P2P vs. Youtube

- BitTorrent capacity to serve the content
  - Piratebay has 5M users at any given point in time
  - Assume average lifetime of 6 hours and download of 0.5GB: total data served = 10,000 TB
  - Factor of 2 for other p2p systems, total = 20,000 TB
- Youtube served 100M videos per day about an year back
- Assume that the number is 200M videos, average video size is 5MB, total data served = 1000TB per day

P2P vs. Youtube

- Capacity to serve the content based on bandwidth capacity
- Piratebay: 5M leechers, 5M seeders
- Assume average of 400Kbps per user
- Translates to about 4 Tbps
- Youtube: assume a 10 Gbps connection from data center
- Then need about 400 data centers to match the serving capacity of BitTorrent

Challenges of Peer-to-Peer

- Peers come and go
  - Peers are intermittently connected
  - May come and go at any time
  - Or come back with a different IP address
- How to locate the relevant peers?
  - Peers that are online right now
  - Peers that have the content you want
- How to motivate peers to stay in system?
  - Why not leave as soon as download ends?
  - Why bother uploading content to anyone else?

13

Locating the Relevant Peers

- Three main approaches
  - Central directory (Napster)
  - Query flooding (Gnutella)
  - Hierarchical overlay (Kazaa, modern Gnutella)
- Design goals
  - Scalability
  - Simplicity
  - Robustness
  - Plausible deniability

14

Peer-to-Peer Networks: Napster

Shawn Fanning,
Northeastern freshman

- Napster history: the rise
  - January 1999: Napster version 1.0
  - May 1999: company founded
  - September 1999: first lawsuits
  - 2000: 80 million users
- Napster history: the fall
  - Mid 2001: out of business due to lawsuits
  - Mid 2001: dozens of P2P alternatives that were harder to touch, though these have gradually been constrained
  - 2003: growth of pay services like iTunes
- Napster history: the resurrection
  - 2003: Napster reconstituted as a pay service
  - 2007: still lots of file sharing going on

15

Napster Technology: Directory Service

- User installing the software
  - Download the client program
  - Register name, password, local directory, etc.
- Client contacts Napster (via TCP)
  - Provides a list of music files it will share
  - … and Napster's central server updates the directory
- Client searches on a title or performer
  - Napster identifies online clients with the file
  - … and provides IP addresses
- Client requests the file from the chosen supplier
  - Supplier transmits the file to the client
  - Both client and supplier report status to Napster

16

Napster Technology: Properties

- Server's directory continually updated
  - Always know what music is currently available
  - Point of vulnerability for legal action
- Peer-to-peer file transfer
  - No load on the server
  - Plausible deniability for legal action (but not enough)
- Proprietary protocol
  - Login, search, upload, download, and status operations
  - No security: cleartext passwords and other vulnerability
- Bandwidth issues
  - Suppliers ranked by apparent bandwidth & response time

17

Napster: Limitations of Central Directory

- Single point of failure
- Performance bottleneck
- Copyright infringement

File transfer is decentralized, but locating content is highly centralized

- So, later P2P systems were more distributed
  - Gnutella went to the other extreme…

18

Peer-to-Peer Networks: Gnutella

- Gnutella history
  - 2000: J. Frankel & T. Pepper released Gnutella
  - Soon after: many other clients (e.g., Morpheus, Limewire, Bearshare)
  - 2001: protocol enhancements, e.g., "ultrapeers"
- Query flooding
  - Join: contact a few nodes to become neighbors
  - Publish: no need!
  - Search: ask neighbors, who ask their neighbors
  - Fetch: get file directly from another node

19

Gnutella: Query Flooding

- Fully distributed
  - No central server
- Public domain protocol
- Many Gnutella clients implementing protocol

Overlay network: graph

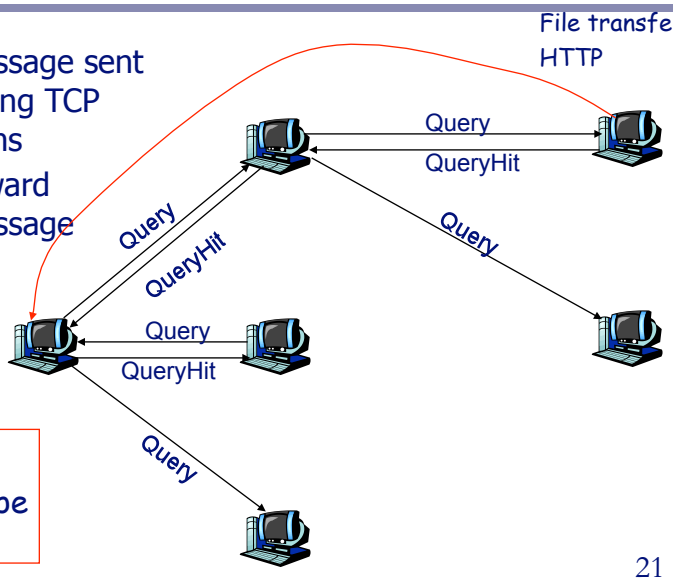
- Edge between peer X and Y if there's a TCP connection
- All active peers and edges is overlay net
- Given peer will typically be connected with < 10 overlay neighbors

20

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

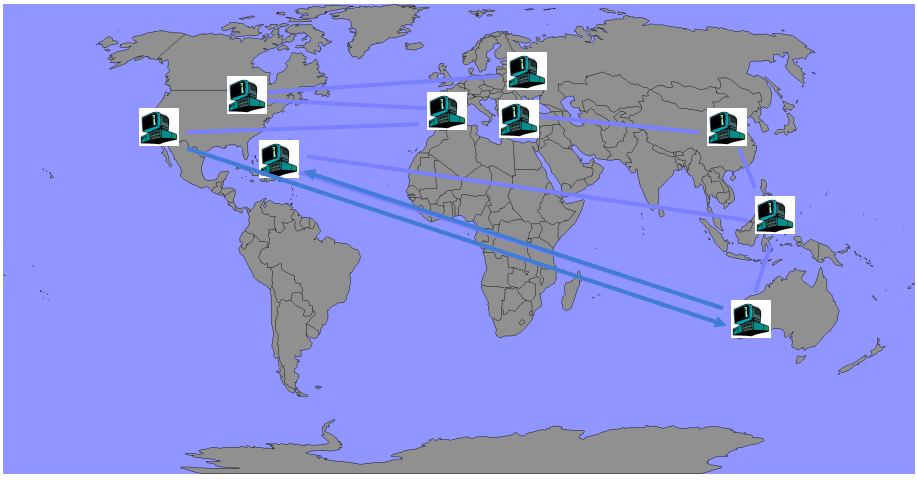
- Joining peer X must find some other peers
  - Start with a list of candidate peers
  - X sequentially attempts TCP connections with peers on list until connection setup with Y
- X sends Ping message to Y
  - Y forwards Ping message.
  - All peers receiving Ping message respond with Pong message
- X receives many Pong messages
  - X can then set up additional TCP connections

Gnutella: Pros and Cons

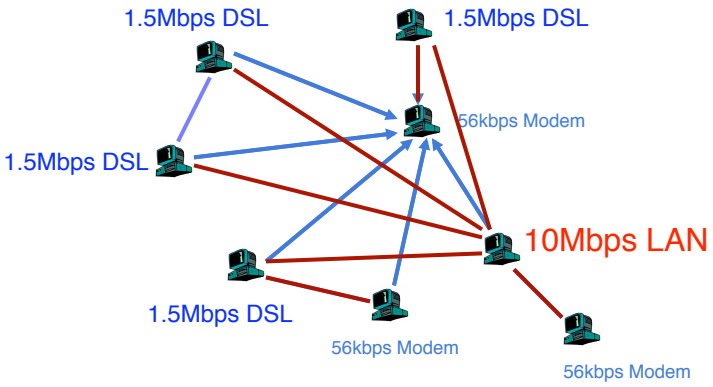
- Advantages
  - Fully decentralized
  - Search cost distributed
  - Processing per node permits powerful search semantics
- Disadvantages
  - Search scope may be quite large
  - Search time may be quite long
  - High overhead, and nodes come and go often

23

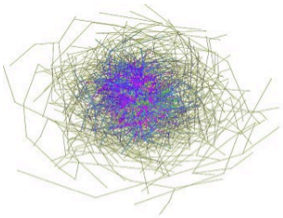
Aside: Search Time?



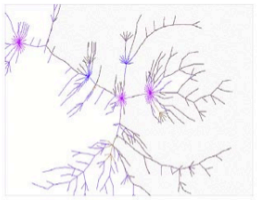
Aside: All Peers Equal?



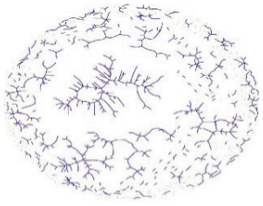
Aside: Network Resilience



Partial Topology



Random 30% die



Targeted 4% die

from Saroiu *et al.*, *MMCN* 2002

Peer-to-Peer Networks: KaAzA

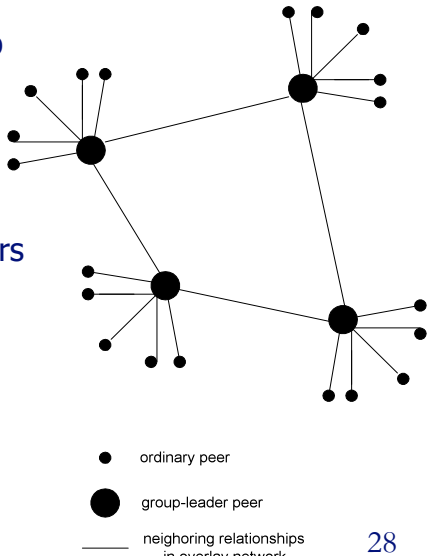
- KaZaA history
  - 2001: created by Dutch company (Kazaa BV)
  - Single network called FastTrack used by other clients as well
  - Eventually the protocol changed so other clients could no longer talk to it

- Smart query flooding
  - Join: on start, the client contacts a super-node (and may later become one)
  - Publish: client sends list of files to its super-node
  - Search: send query to super-node, and the super-nodes flood queries among themselves
  - Fetch: get file directly from peer(s); can fetch from multiple peers at once

27

KaZaA: Exploiting Heterogeneity

- 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



28

KaZaA: Motivation for Super-Nodes

- Query consolidation
  - Many connected nodes may have only a few files
  - Propagating query to a sub-node may take more time than for the super-node to answer itself
- Stability
  - Super-node selection favors nodes with high up-time
  - How long you've been on is a good predictor of how long you'll be around in the future

29

Peer-to-Peer Networks: BitTorrent

- BitTorrent history and motivation
  - 2002: B. Cohen debuted BitTorrent
  - Key motivation: popular content
    - Popularity exhibits temporal locality (Flash Crowds)
    - E.g., Slashdot effect, CNN Web site on 9/11, release of a new movie or game
  - Focused on efficient *fetching*, not searching
    - Distribute same file to many peers
    - Single publisher, many downloaders
  - Preventing free-loading

30

BitTorrent: Simultaneous Downloading

- Divide large file into many pieces
  - Replicate different pieces on different peers
  - A peer with a complete piece can trade with other peers
  - Peer can (hopefully) assemble the entire file
- Allows simultaneous downloading
  - Retrieving different parts of the file from different peers at the same time
  - And uploading parts of the file to peers
  - Important for very large files

31

BitTorrent: Tracker

- Infrastructure node
  - Keeps track of peers participating in the torrent
- Peers register with the tracker
  - Peer registers when it arrives
  - Peer periodically informs tracker it is still there
- Tracker selects peers for downloading
  - Returns a random set of peers
  - Including their IP addresses
  - So the new peer knows who to contact for data

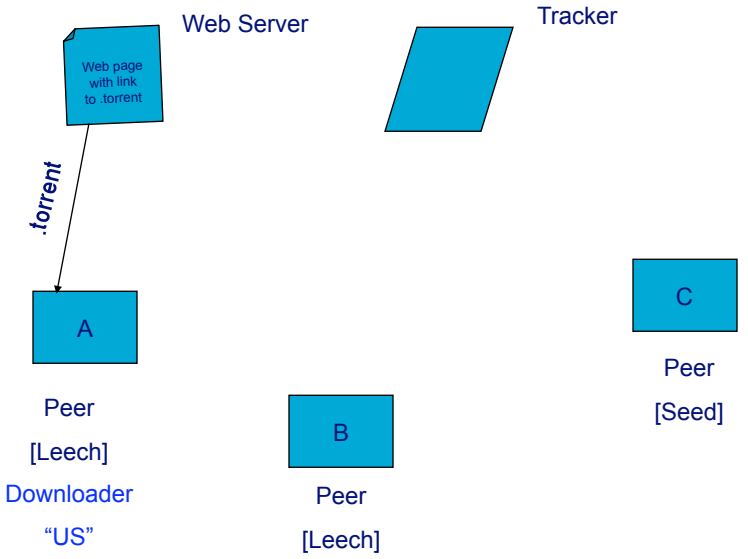
32

BitTorrent: Chunks

- Large file divided into smaller pieces
  - Fixed-sized chunks
  - Typical chunk size of 16KB - 256 KB
- Allows simultaneous transfers
  - Downloading chunks from different neighbors
  - Uploading chunks to other neighbors
- Learning what chunks your neighbors have
  - Broadcast to neighbors when you have a chunk
- File done when all chunks are downloaded

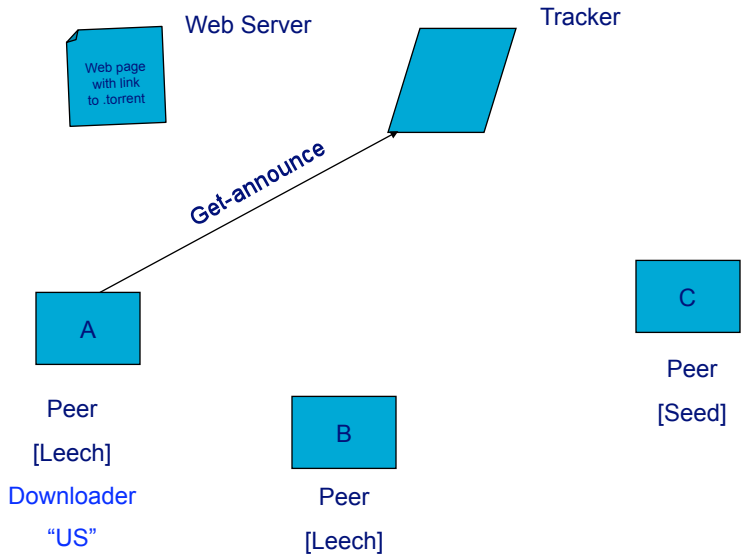
33

BitTorrent: Overall Architecture



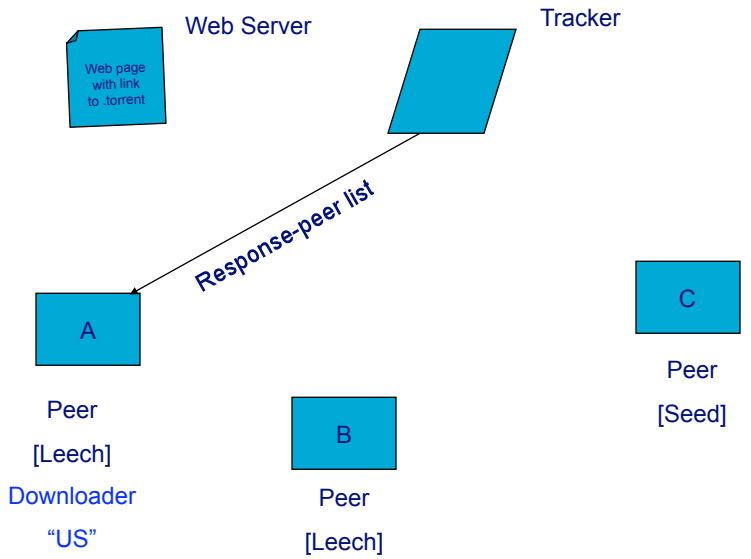
34

BitTorrent: Overall Architecture



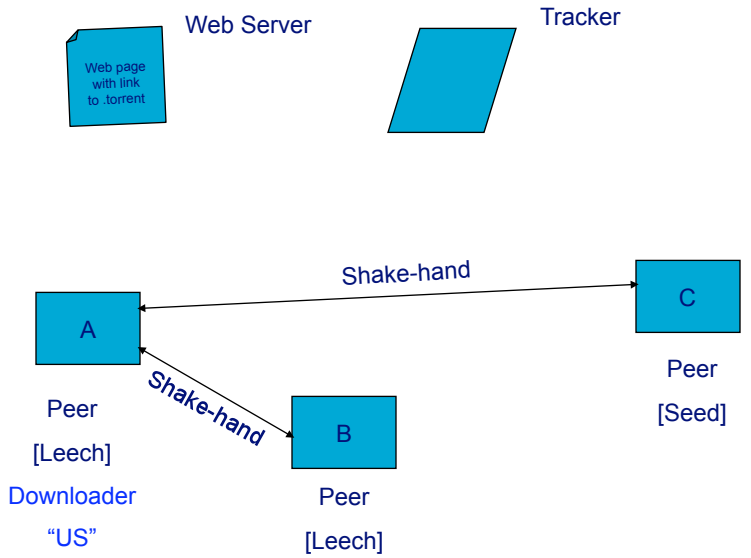
35

BitTorrent: Overall Architecture



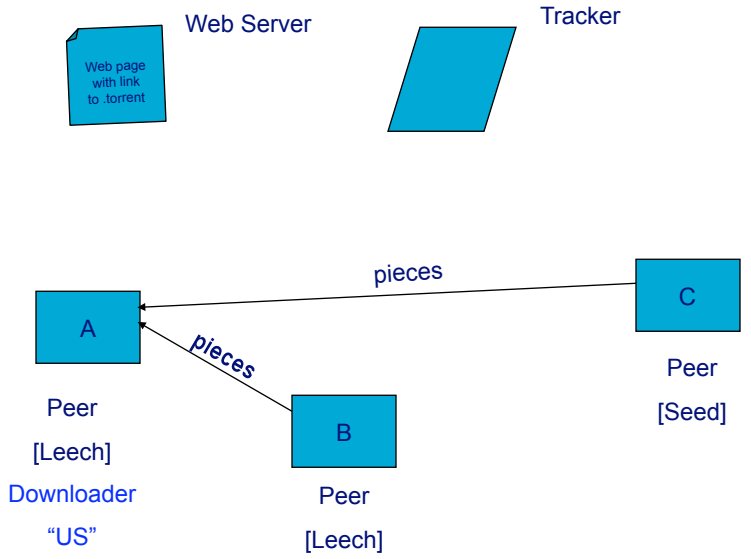
36

BitTorrent: Overall Architecture



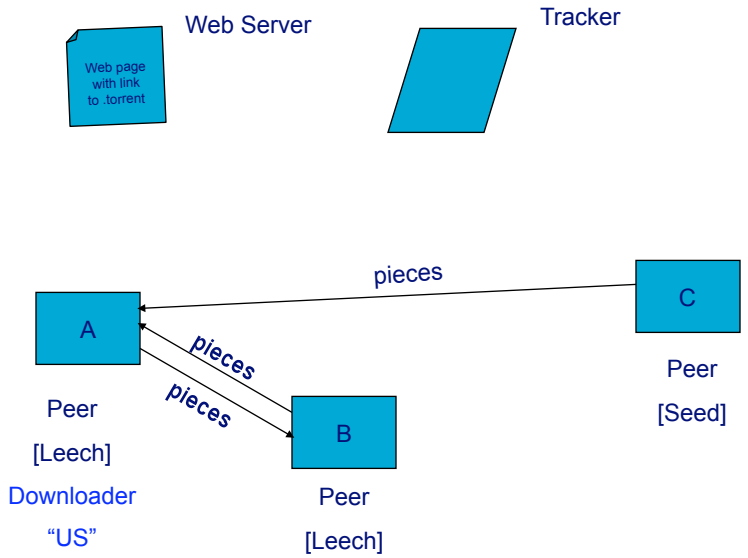
37

BitTorrent: Overall Architecture



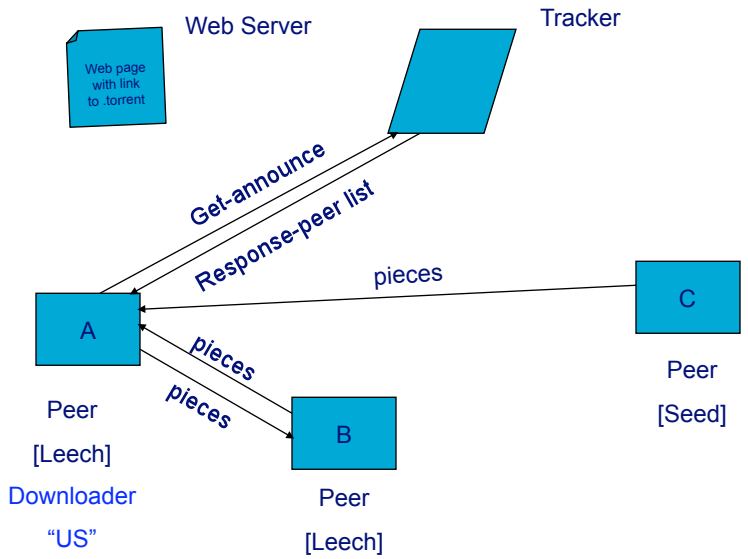
38

BitTorrent: Overall Architecture



39

BitTorrent: Overall Architecture



40

BitTorrent: Chunk Request Order

- Which chunks to request?
  - Could download in order
  - Like an HTTP client does
- Problem: many peers have the early chunks
  - Peers have little to share with each other
  - Limiting the scalability of the system
- Problem: eventually nobody has rare chunks
  - E.g., the chunks need the end of the file
  - Limiting the ability to complete a download
- Solutions: random selection and rarest first

41

Free-Riding Problem in P2P Networks

- Vast majority of users are free-riders
  - Most share no files and answer no queries
  - Others limit # of connections or upload speed
- A few "peers" essentially act as servers
  - A few individuals contributing to the public good
  - Making them hubs that basically act as a server
- BitTorrent prevent free riding
  - Allow the fastest peers to download from you
  - Occasionally let some free loaders download

42

Bit-Torrent: Preventing Free-Riding

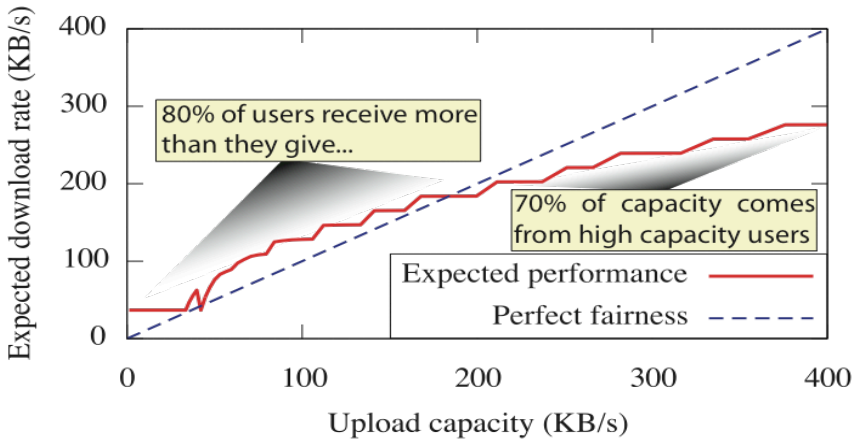
- Peer has limited upload bandwidth
  - And must share it among multiple peers
- Prioritizing the upload bandwidth
  - Favor neighbors that are uploading at highest rate
- Rewarding the top four neighbors
  - Measure download bit rates from each neighbor
  - Reciprocates by sending to the top four peers
  - Recompute and reallocate every 10 seconds
- Optimistic unchoking
  - Randomly try a new neighbor every 30 seconds
  - So new neighbor has a chance to be a better partner

43

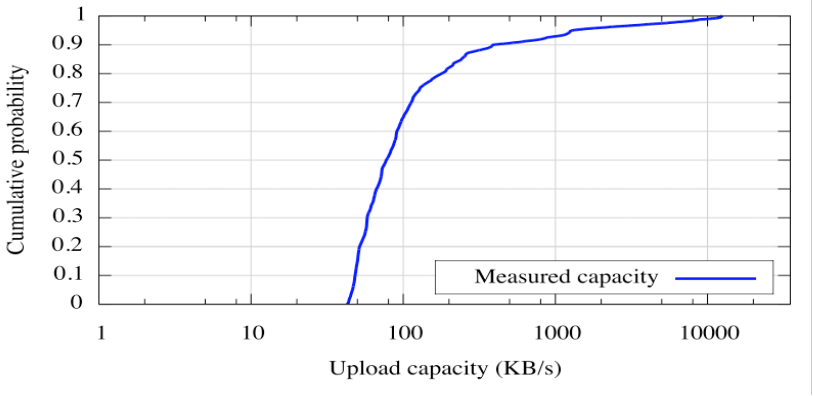
Study BitTorrent's Incentives

- First, construct a model to predict *unreciprocated* altruism
  - Measure large number of popular swarms
  - Estimate fairness, altruism, and reciprocation behavior

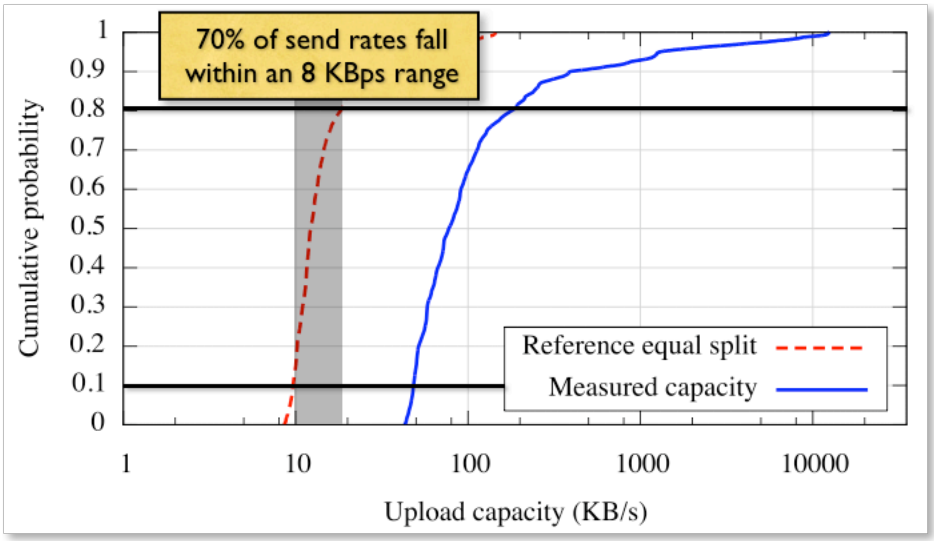
Fairness



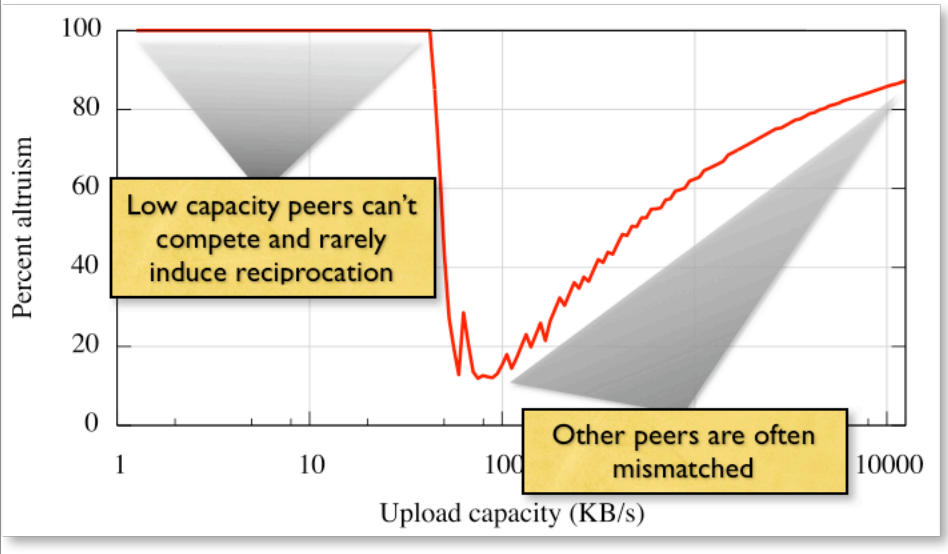
End-host capacities



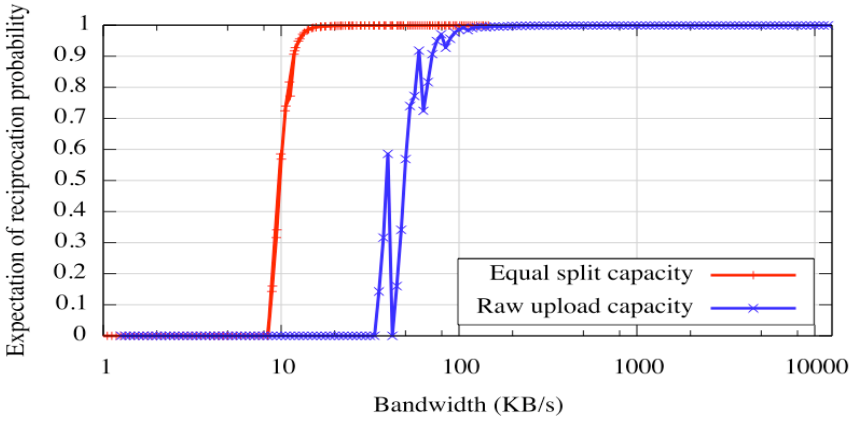
Per-Peer Send Rates



Altruism



Reciprocation Probability



Methodology

- First, construct a model to predict *unreciprocated* altruism
  - Measure large number of popular swarms
  - Estimate fairness, altruism, and reciprocation behavior
- Second, develop a strategic client: BitTyrant

BitTyrant: Strategic Peer Selection

Select peers and rates to maximize "return-on-investment"

Each round, rank order each peer p by the ratio d_p/u_p, and choose those of top rank until the local upload capacity is reached.

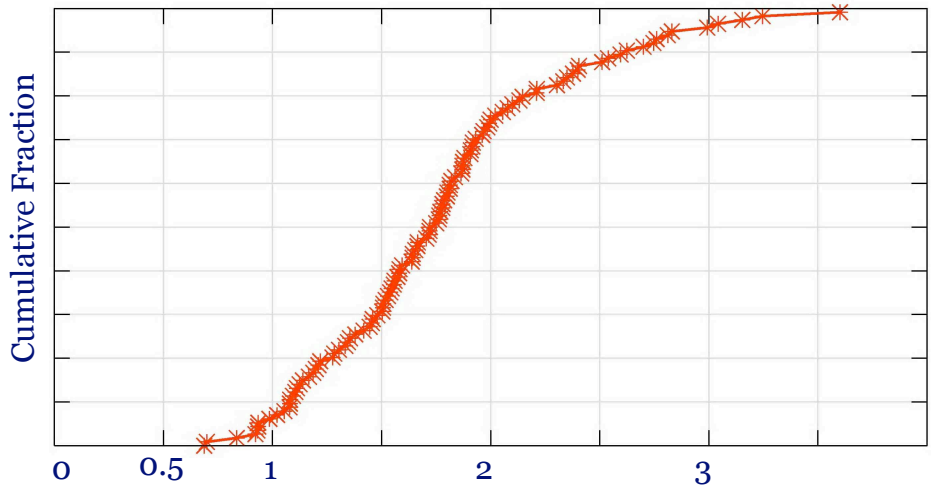
$$\underbrace{\frac{d_0}{u_0}, \frac{d_1}{u_1}, \frac{d_2}{u_2}, \frac{d_3}{u_3}, \frac{d_4}{u_4}}_{\text{choose } k \mid \sum_{i=0}^{k} u_i \leq \text{capacity}} , \ldots$$

At the end of each round for each unchoked peer:

If peer p does not send data: increase cost estimate, u_p.

If peer p has unchoked us for the last minute: reduce cost estimate, u_p.

BitTyrant Performance



BitTorrent Today

- Well designed system with *some* incentives
- Significant fraction of Internet traffic
  - Estimated at 30%
  - Though this is hard to measure
- Problem of incomplete downloads
  - Peers leave the system when done
  - Many file downloads never complete
  - Especially a problem for less popular content
- Still lots of legal questions remains
- Further need for incentives

53

Distributed Hash Tables (DHT): History

- In 2000-2001, academic researchers jumped on to the P2P bandwagon
- Motivation:
  - Guaranteed lookup success for files in system (the search problem that BitTorrent doesn't address)
  - Provable bounds on search time
  - Provable scalability to millions of node
- Hot topic in networking ever since

DHT: Overview

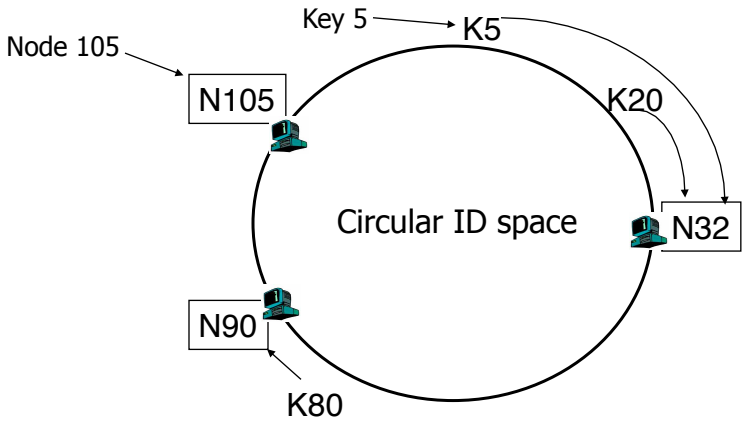
- **Abstraction**: a distributed "hash-table" (DHT) data structure:
  - put(id, item);
  - item = get(id);
- **Implementation**: nodes in system form an interconnection network
  - Can be Ring, Tree, Hypercube, Butterfly Network, ...

DHT: Example - Chord

- Associate with each node and file a unique *id* in an *uni*-dimensional space (a Ring)
  - E.g., pick from the range $[0...2^m]$
  - Usually the hash of the file or IP address
- Properties:
  - Routing table size is O(log *N*) , where *N* is the total number of nodes
  - Guarantees that a file is found in O(log *N*) hops

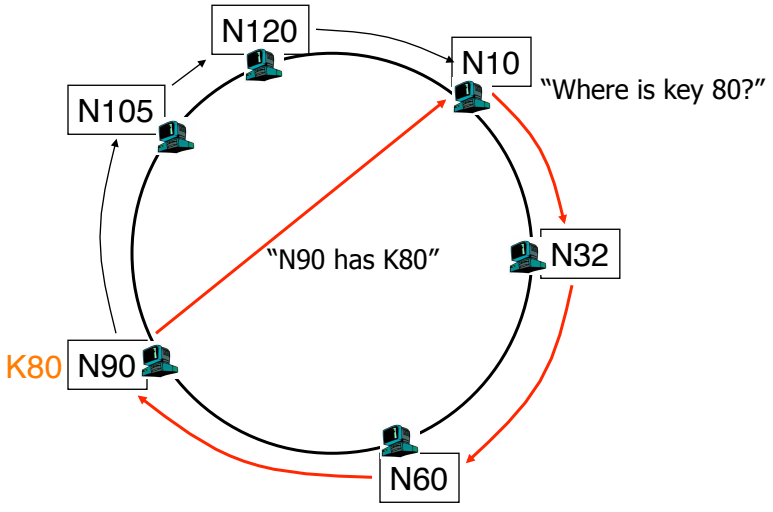
from MIT in 2001

DHT: Consistent Hashing

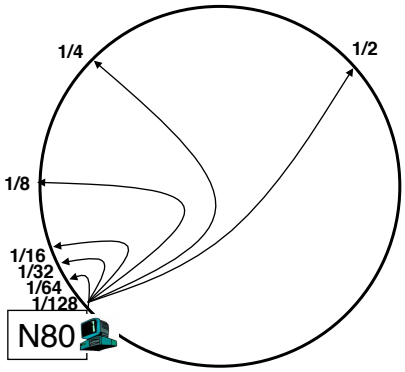


A key is stored at its successor: node with next higher ID

DHT: Chord Basic Lookup



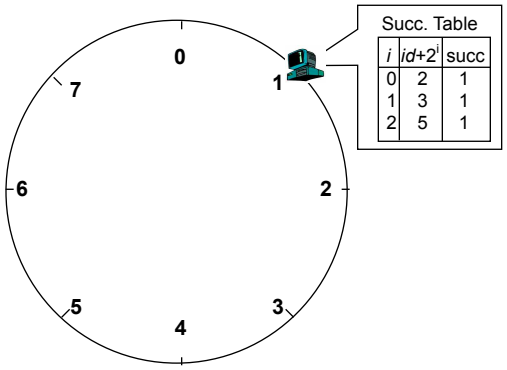
DHT: Chord "Finger Table"



- Entry *i* in the finger table of node *n* is the first node that succeeds or equals $n + 2^i$
- In other words, the i^{th} finger points $1/2^{n-i}$ way around the ring

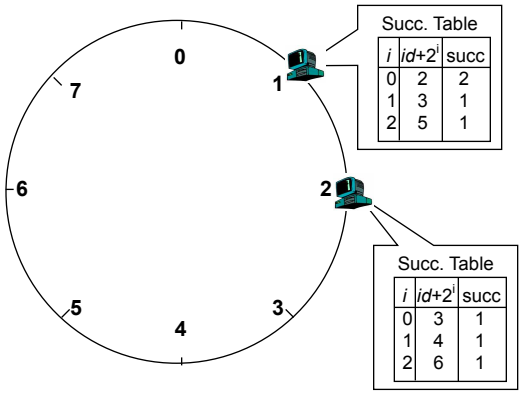
DHT: Chord Join

- Assume an identifier space [0..8]
- Node n1 joins



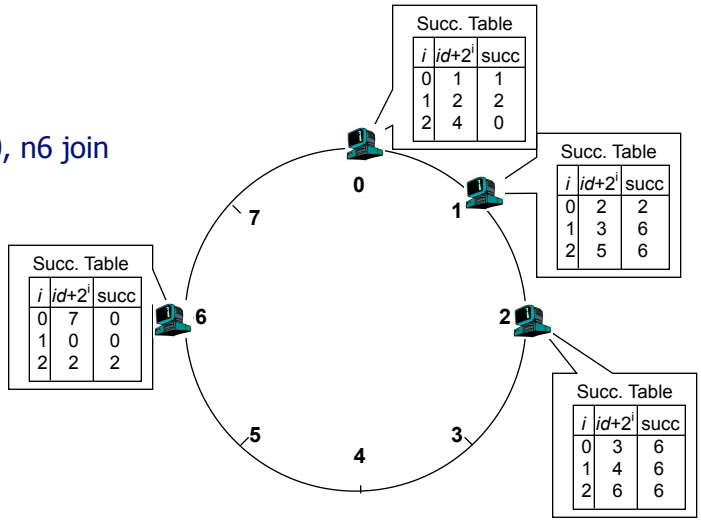
DHT: Chord Join

- Node n2 joins



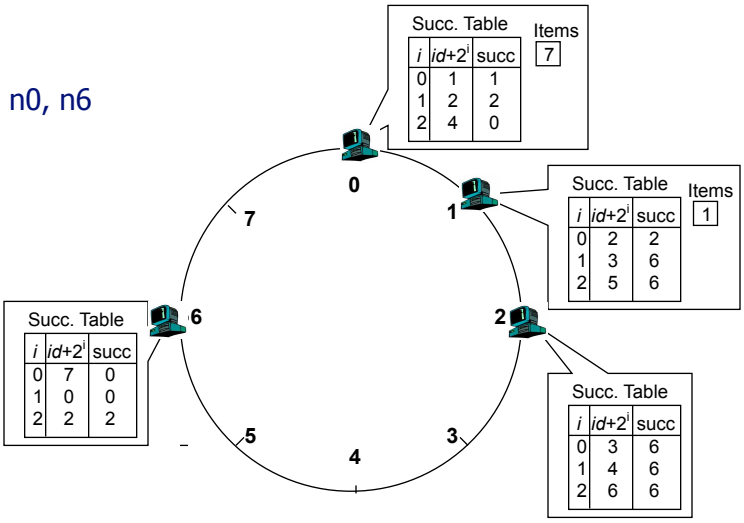
DHT: Chord Join

- Nodes n0, n6 join



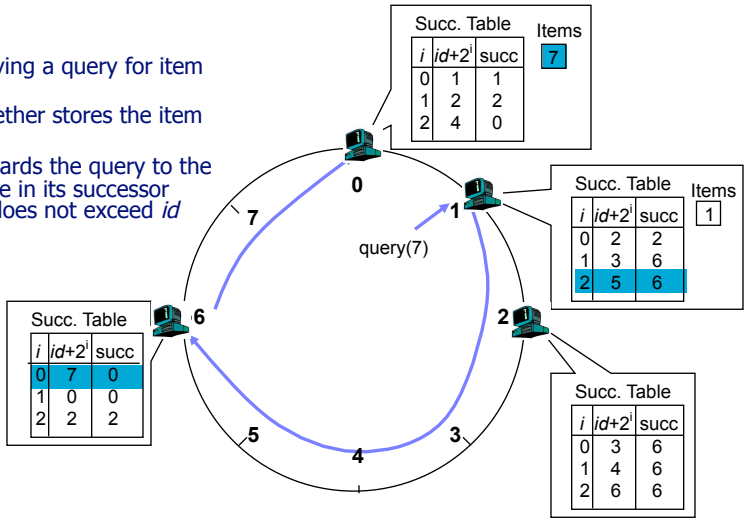
DHT: Chord Join

- Nodes:
  n1, n2, n0, n6
- Items:
  f7, f1



DHT: Chord Routing

- Upon receiving a query for item *id*, a node:
- Checks whether stores the item locally
- If not, forwards the query to the largest node in its successor table that does not exceed *id*



DHT: Chord Summary

- Routing table size?
  - Log *N* fingers
- Routing time?
  - Each hop expects to 1/2 the distance to the desired id => expect O(log *N*) hops.
- What is good/bad about Chord?