Distributed Systems

Peer-to-Peer

Rik Sarkar

University of Edinburgh
Fall 2014

Peer to Peer

- The common perception
  - A system for distributing (sharing?) files
  - Using the computers of common users (instead of servers)
  - A popular file is hosted by one or more users' computers
  - Someone who needs the file can download from one or more users
  - The P2P system provides easy methods to search for files and download them

Peer to Peer

- More generally:
  - Files are not the only things that can be shared
  - Users can share computing power
    - CPU cycles
    - Storage
    - Anonymity (lookup The Onion Router)
- Peer: One that is of equal standing to others in the group
  - Everyone is server *and* a client
  - They provide the service as well as use it

Client – Server model

- The traditional model of internet service is client server
- For a service X (search, email…)
  - There is a specific known server
  - Clients (browsers, email clients) contact the server to get data

Client – Server model (drawbacks)

- Central point of failure
  – When the server fails, entire service goes down
  – If the server does not recover, all data may be lost
- Load management
  – When many clients send requests, everyone gets slow response
  – Popular content gets slower service!
- Addressing: have to “know” the server or search for it

P2P: Motivations

- Tolerance to faults/attacks
- Load balancing
- User participation
- Cost efficiency
- Hard to control

Fault/attack tolerant

- Everyone is a server, serving part of the data store
- Each file has multiple copies
- Failures of few or even many computers does not take down the entire service
- Hard to attack everyone at the same time

Load balanced

- Each file is hosted by multiple users
- If many users want to download, the job gets divided
- Each host handles only a small load, so does not get overloaded
- Each downloader gets faster speed

Participation

- Everyone feels involved
- "I am providing something useful to the entire world!"
- A unique application to inspire user-participation (crowdsourcing). Internet 2.0?
- Previously (say, in 1999), internet used to be a passive experience for most people
  - Except the lucky few who had access to servers and could publish web pages
- Participation is critical to user interest

Cost efficiency

- A file or service can be provided without the expense of a large server
- Popular content is hosted by many users
- Popular content gets better and faster service!
  - Unlikely to be lost due to failure
- Large delivery bandwidth does not require expensive server or infrastructure

Hard to control

- And therefore hard to take down
- No one person has much authority over the system

Some Properties

- Unreliable, uncoordinated, unmanaged
  - No central Authority, peers are independent
  - Increases flexibility of individual peers, but makes overall system (possibly) unreliable
- Resilient to attack, heterogeneous
  - Large number of peers, hard to take down
- Large collection of resources
  - Volunteer participation, global reach

Issues in p2p

- Connecting -- bootstrapping
- Finding content
- Quality of service
- Quality of data
- Hard to control

Issues in p2p

- **Connecting – bootstrapping**
- We first need a network
- Suppose we want to connect to a p2p system
- We need to find some members of the existing system to join the system
  - How can we do that?
- Remember, there is no "server" with fixed address that we can always use to connect

Issues in p2p

- **Finding content**
- Suppose we have managed to find the network somehow
- We now want to find a particular video
- We don't know who has it
- Hard to build a search service, since peers regularly join and leave the system

Issues in p2p

- **Quality of service**
- How fast a download or service works may depend on who is hosting the file/service
- A file/service may be unavailable simply because all the peers hosting it are unavailable
- Hard to rely on it..

Issues in p2p

- **Quality of data**
- You ask for file X
- Node Y claims to have the file
- You download the file, and then find it is something completely different
- We can’t prevent node Y from making false claims

Issues in p2p

- **Quality of data**
- You ask for file X
- Node Y claims to have the file
- You download the file, and then find it is something completely different
- We can’t prevent node Y from making false claims

Issues in p2p

- **Hard to control**
- Therefore hard to guarantee anything
- The service may deteriorate in quality and hard to do anything about it

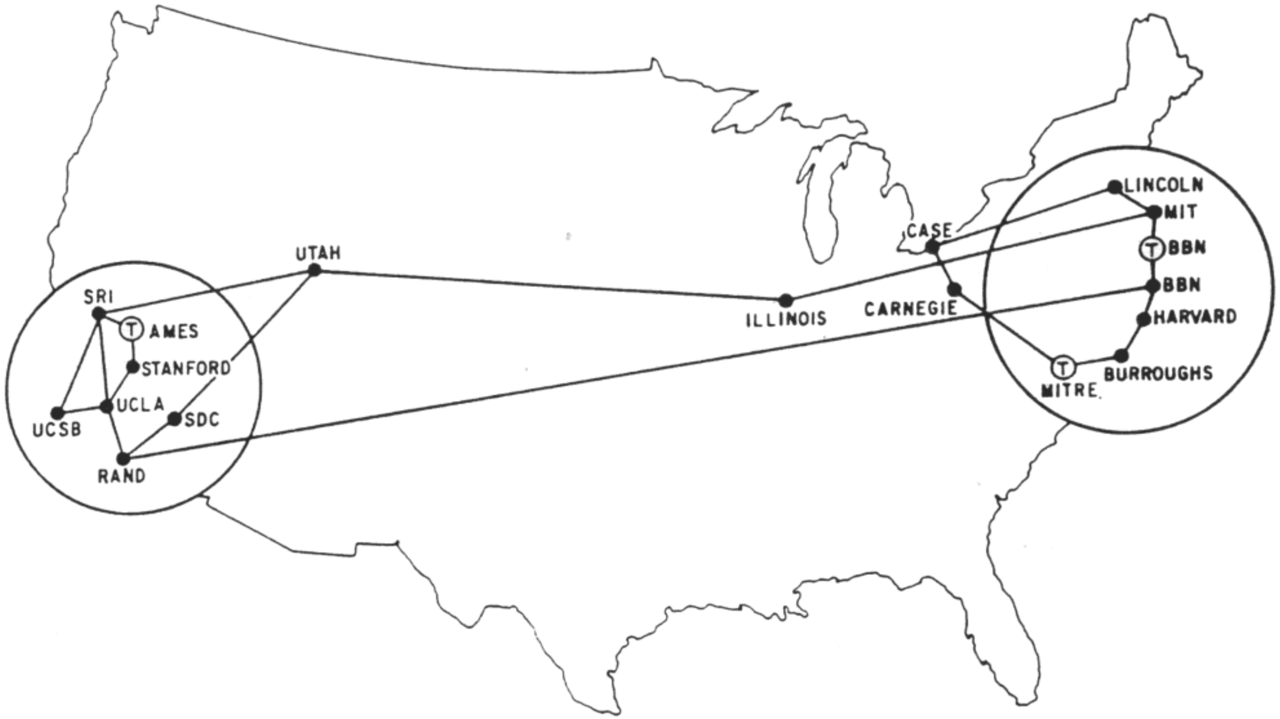
Examples

- Arpanet-Internet
- SETI@Home
- Napster
- Gnutella
- Bittorrent
- Skype

ARPAnet -- internet

- Advanced research project agency of US defense built a network
  - To facilitate communication between few universities working on defense and ARPA projects
  - Each university had a few computers on this network (computers were very expensive)
  - They can send messages using those computers
  - Each computer acted as server as well as client
- This network eventually grew to be the Internet

ARPAnet -- internet



MAP 4 September 1971

ARPAnet -- internet

- Original design of the Internet was with "peers" – all computers on equal footing
- The internet is still fundamentally a peer-based system
- You can have a server on your computer, and the network protocols treat it the same way as any other computer/server
- So we can use our personal computers to host web pages or other service
- (Your ISP may make it difficult, but this is a money issue, not a technology one)

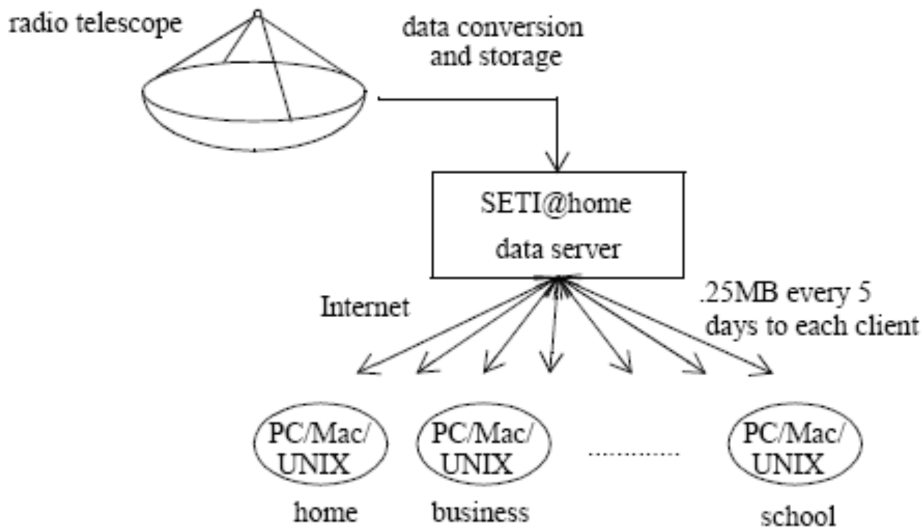
SETI@Home

- Search for extra-terrestrial Intelligence
- Radio signal data from outer space are collected by astronomical telescopes/antennae
- To be analyzed for signs of "artificial signal" structures created by intelligent life in other planets
- The data is split into small chinks for analysis by different computers
- SETI@home volunteers have the software installed on their computers
- The software contacts the UC Berkeley Server and downloads data
- When the computer is not in heavy use, the software analyzes data and sends results back to server

SETI@Home

- Still relies largely on the central server for coordination
- Individual partcipants only do the computation they are asked to
- No communication to peers

- Uses P2P for computation instead of the usual file sharing



Napster

- Music sharing software
- Software makes list of all songs user wants to share
- Uploads list of songs to napster server(s)
  - (large systems need server farms – a distributed system in itself)
- When someone searches for a song, the search goes to server
- Server returns list of peers (IP addresses) that have the song, and it thinks are online
- Song download happens directly from one of the peers

Napster

- Central server based indexing and search
  - Single point of failure
- Connecting to the network is easy – connect to server
- Download is fast – download from peer
- Download from a single peer
- No verification of data correctness

Napster -- History

- Started in 1999
- Popular -- 13 million users in 2001
- Copyright lawsuits throughout
- Millions in fines
- Bankrupt and closed in 2002

- “napster” brand exists as music store

Gnutella

- Trying to address napster's drawbacks
- Completely distributed
  - No server for indexing and searching
  - Open protocol – anyone can build software
- Gnutella used an overlay network for search
  - Every node had a few peers as "neighbors"
  - Choice of neighbors unrelated to underlying network
- Search queries flooded in overlay network to reach all peers
- Any node that has the file responds to search
  - Response routed along the path that the search took to arrive to node
- The file is downloaded from one of the responders
  - The download happens directly from the peer (not through the overlay network)

Gnutella

- Flooding for search was inefficient
  - Cost can be reduced by using TTL and limiting search radius, but still inefficient
- Need the IP address of at least 1 peer to join network
  - Then can connect find other peers through it
  - In practice, some peers were known to be always running (servers)
  - No fully distributed solution to this problem
- No verification of data/content
- More distributed operation than other systems
- No longer active
- Replaced by Kaaza, limewire etc

Bittorrent

- A file/folder shared creates a “torrent” file
  - Acts as a more detailed description than simply the name
  - Contains name
  - Contains list of trackers
    - Trackers are servers that maintain list of peers hosting the file
  - Contains list of chunks & checksums
    - Chunks are parts of the shared file
    - Checksums are hashes to make sure that the correct data has been downloaded

Bittorrent

- Torrent files are found on web sites
  - Bittorrent does not attempt to implement search
- Bittorrent software contacts trackers to get list of peers that have or are downloading file
  - Seeds and leeches
- Contacts them to get lists of chunks they have
- Starts downloading multiple chunks in parallel from different peers
- Randomly, but preferring the more rare chunks

Bittorrent

- Rewards peers for more sharing
  - The more you upload, the better download speeds you get
- Prefers faster peers for download

Skype

- Communication software
- Central server to find IP address or for initial contact to user
- After that, communication occurs directly, server does not see messages
- Means receiver does not get messages until both sender and receiver are online and aware of each-other
- Uses Voice over IP (VoIP) for audio
- Allows phone calls with credit
  - Skype has an office phone line in country X
- When user calls a number in country X
  - The call goes to skype office in X through Internet (free of cost)
  - Then it is routed to the regular phone (cost of a local call)
  - To skype, it costs like a local call
  - User charged a bit more for profit
  - Still cheaper than International call

What is P2P good for?

- In principle, can be used for all sorts of sharing
- Possible to rebuild entire Internet as p2p
  - Everyone participates
  - Any resources can be anywhere, found and delivered through p2p
- Not very practical, hard to do efficiently
- Problem: peers are too dynamic, unreliable
- Adapting to that, makes the system inefficient
  - Think of Gnutella search
- Still some interesting questions remain
  - Can we use it to distribute data better? Ie. What if users stored data in general, and not what they downloaded
  - Can we use it to distribute computation in general?

Some criteria for using p2p design

- Budget – p2p is low budget solution to distribute data/computation
- Resource relevance/popularity – if the item is popular, p2p is useful. Otherwise the few users may go offline..
- Trust – if other users can be trusted, p2p can be a good solution.
  - Can we build a secure network that operates without this assumption?
- Rate of system change – if the system is too dynamic, p2p may not be good. (Imagine peers joining/leaving too fast)

- Rate of content change – p2p is good for static/fixed content. Not good for contents that change regularly, since then all copies have to be updated.

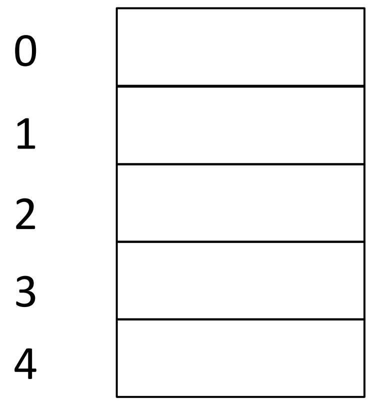
- Criticality – p2p is unreliable, since peers cats independently, may leave/fail any time.
  - P2P is good for applications that are good to have but are not critical to anything urgent

Better p2p design: Some theory

- File transfer in p2p is scalable (efficient even in large systems with many nodes)
  - Occurs directly between peers using Internet
  - Bittorrent like systems can download from multiple peers – more efficiency
- The problem in p2p:
  - Search is inefficient in large systems

Hash tables

- A hash tables has b buckets
  - Any item x is put into bucket h(x)
  - h(x) must be at most b for all x
- Example: a hash table of 5 buckets
  - Any item x is put into bucket x mod 5
  - Insert numbers 3, 5, 12, 116, 211



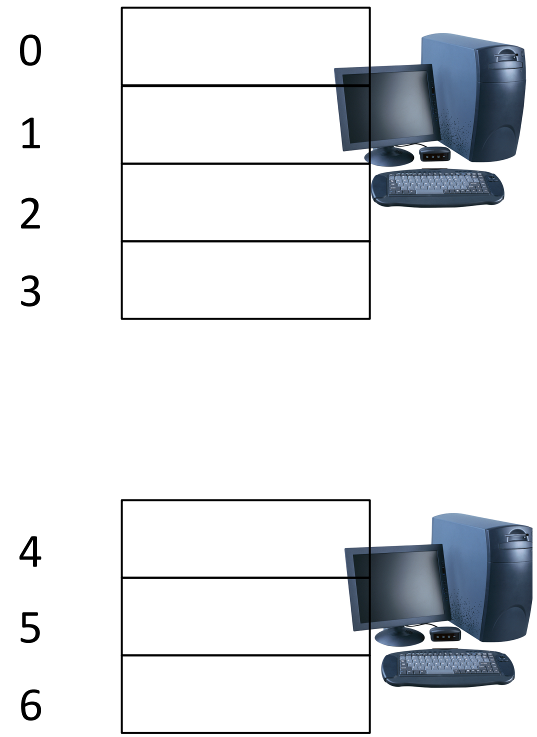
Hash tables

- Hash tables are used to find elements quickly
- Suppose we use hash on the file name "fname"
- Then h("fname") takes us to the bucket containing file fname
- If the bucket has many files, then we will still have to search for the file inside the bucket
- But if our hash table is reasonably large, then usually there will be only a few files in the bucket – easy to search

0	5
1	116, 211
2	2
3	3
4	

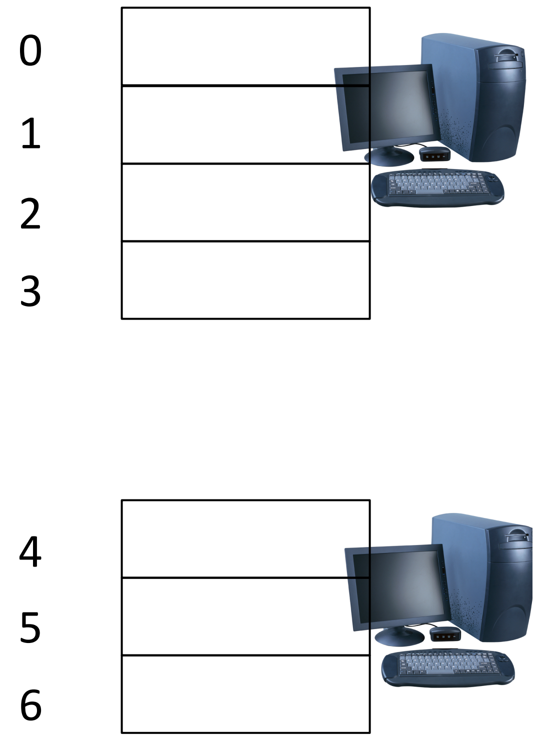
Distributed hash tables

- Each computer knows the hash function
- Each computer is responsible for some of the hash buckets
- Different parts of the data are stored in different computers



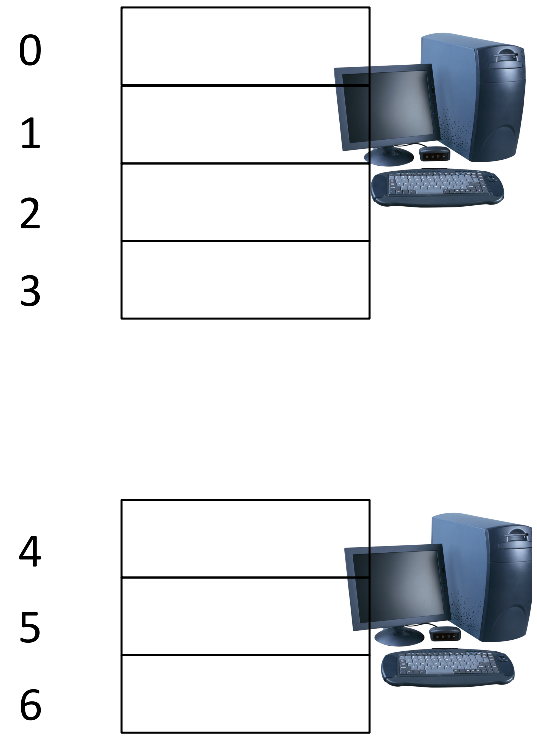
Distributed hash tables

- Elements can be inserted/ retrieved as usual to the corresponding bucket
  - But need to ask the computer responsible for that bucket
- Need efficient mechanism to find the responsible node
  - Using communication between nodes



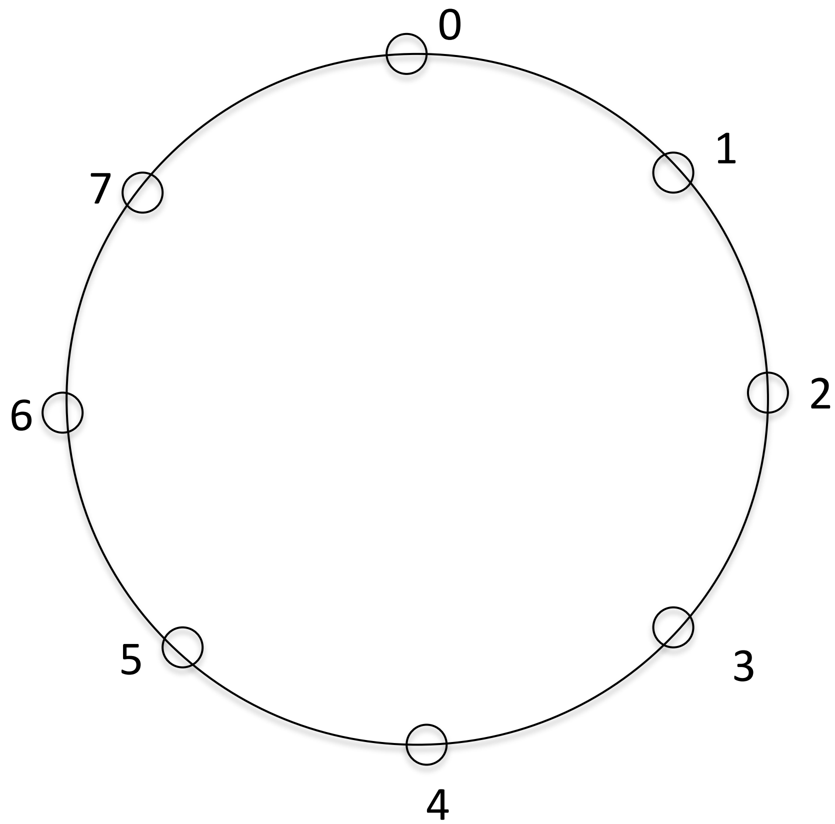
Distributed hash tables

- P2p systems are dynamic
  - Nodes join/leave all the time
  - Need a mechanism to shift responsibilities with change



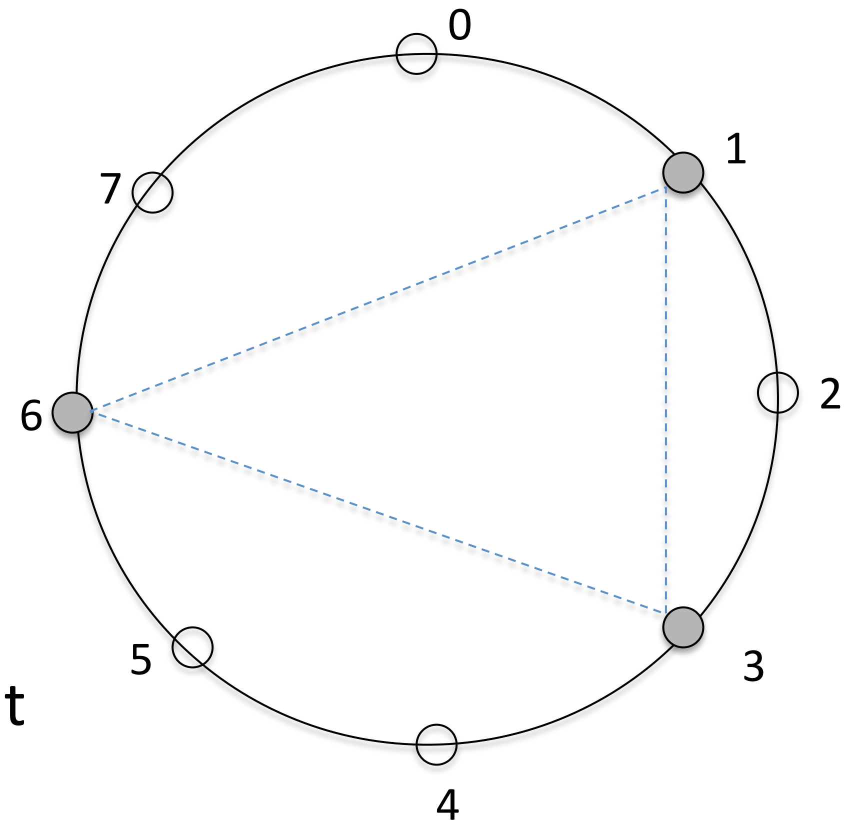
Example system: Chord

- P2P system from MIT (2001)
- Operates using a ring overlay for the set of node ids
- Each id has a *slot* in the overlay
  - Each slot may not be occupied



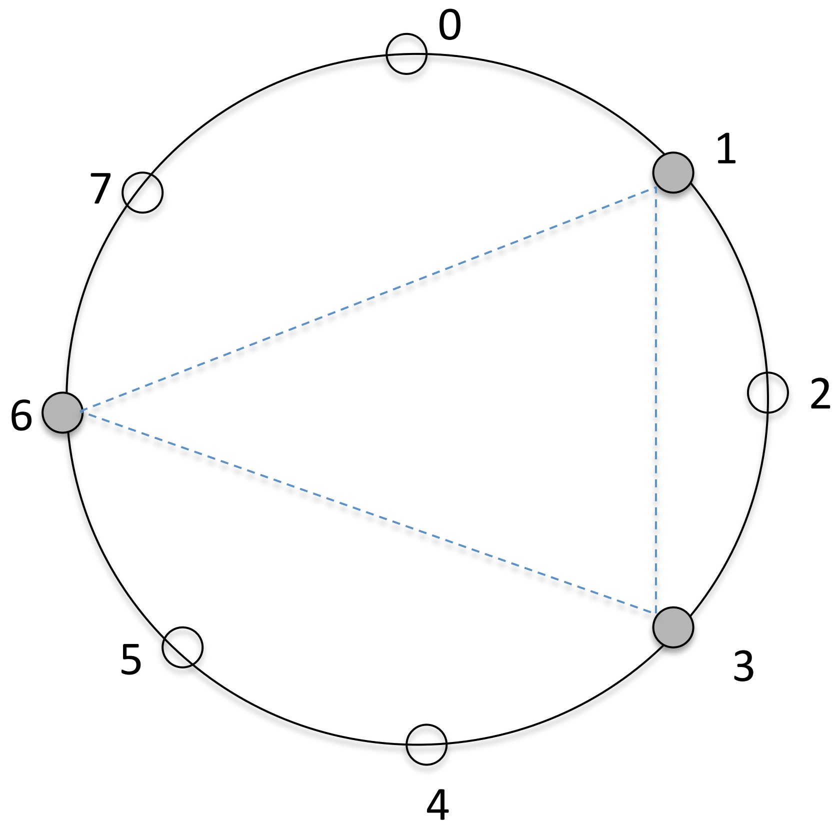
Example system: Chord

- Each node knows the *next* and *previous* occupied slots in the ring
- Storage using hash tables
- To store/retrieve data, forward message to *next* until reaching the node with the bucket
- If the slot is not occupied, (for example, 5 in the figure), store it at the next occupied slot (eg. 6)



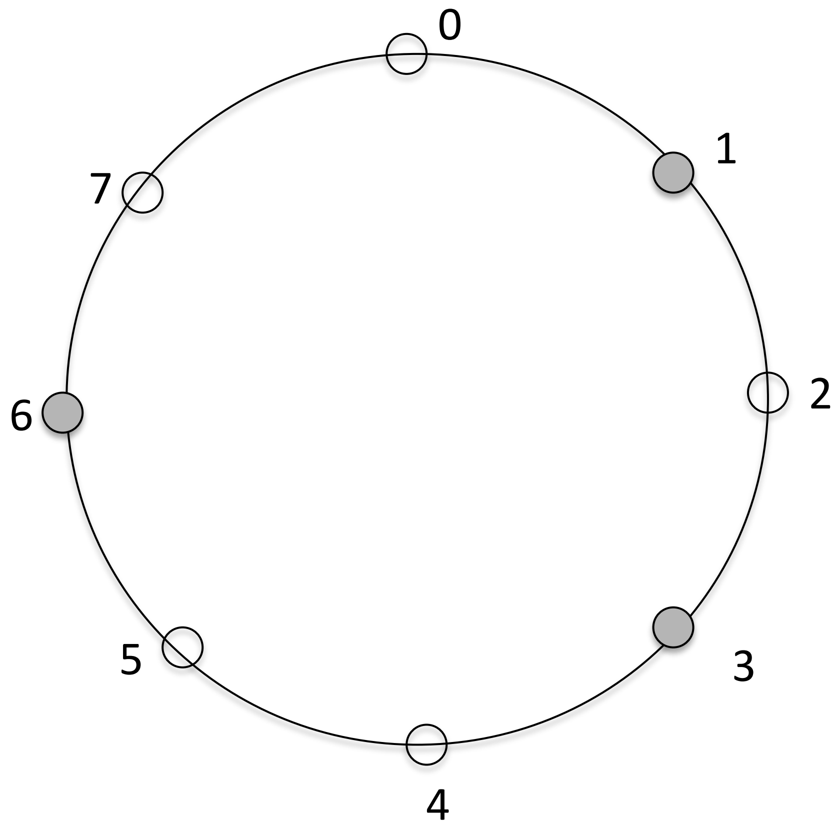
Example system: Chord

- When a node wants to join, it finds occupied slots just before/after itself
- Example: 5 wants to join
  - 5 has to know at least one node already in system, say node 1.
  - 5 sends search message for itself to 1
  - The message gets forwarded using *next* pointers
  - Node 3 and 6 realize that they are neighbors of 5
  - Message sent back to 5



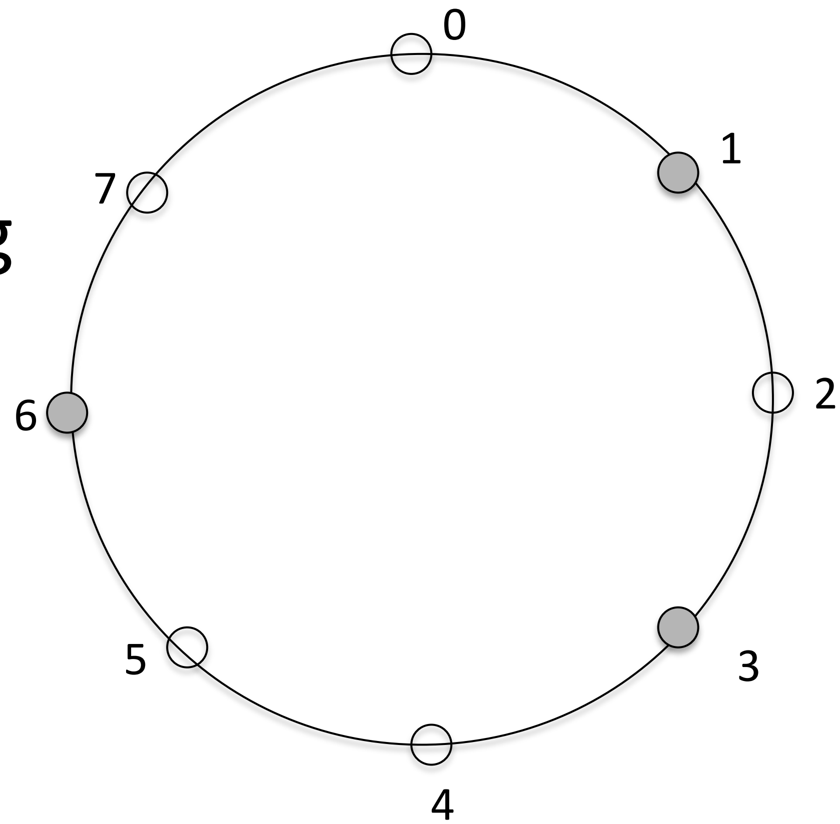
Example system: Chord

- 6 can send 5's hash table to 5
- Each node replicates all the data for several nodes before/after itself
- If a node fails, its data is still preserved



Example system: Chord

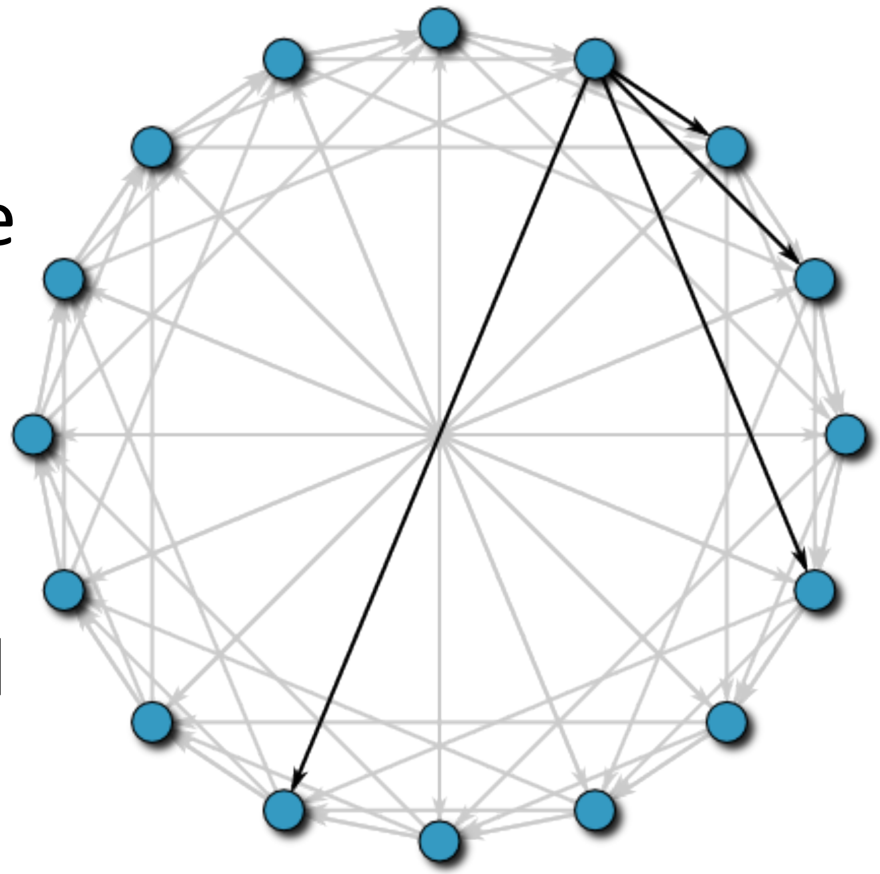
- Problem: search is still inefficient
- It goes sequentially along the ring
- Cost: O(n)
- Now imagine a ring with a million nodes!



Chord: more efficient search

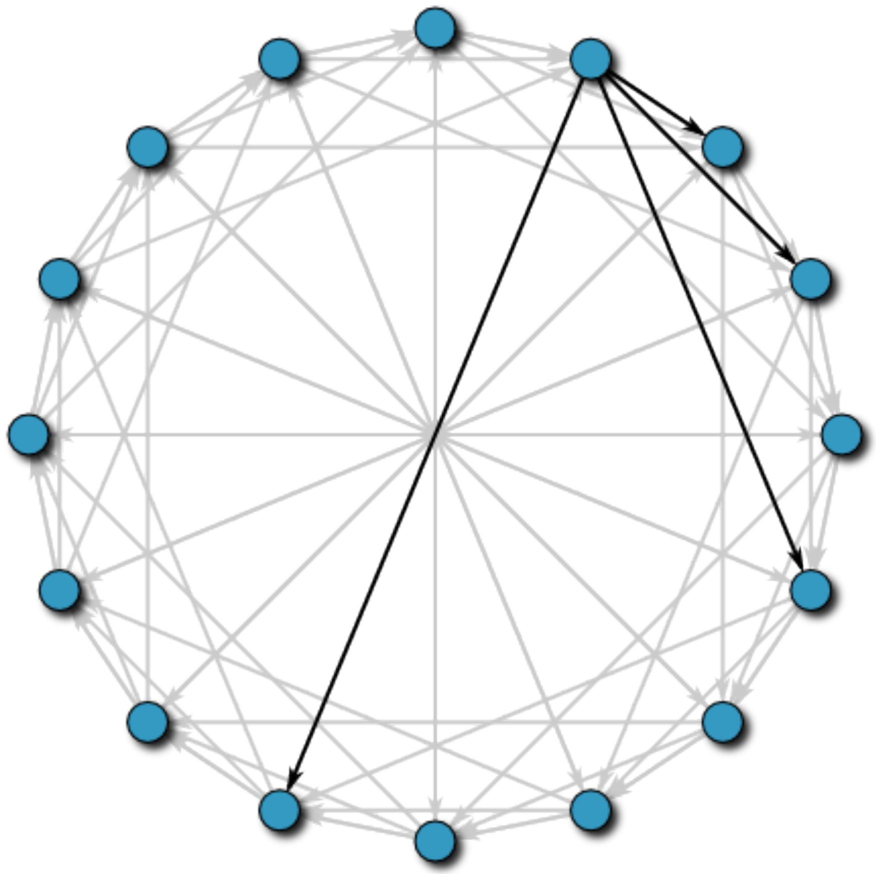
- Add some extra links in the overlay graph
- To find node x, go to the neighbor that is nearest to the destination

- Which extra links to add to the network?



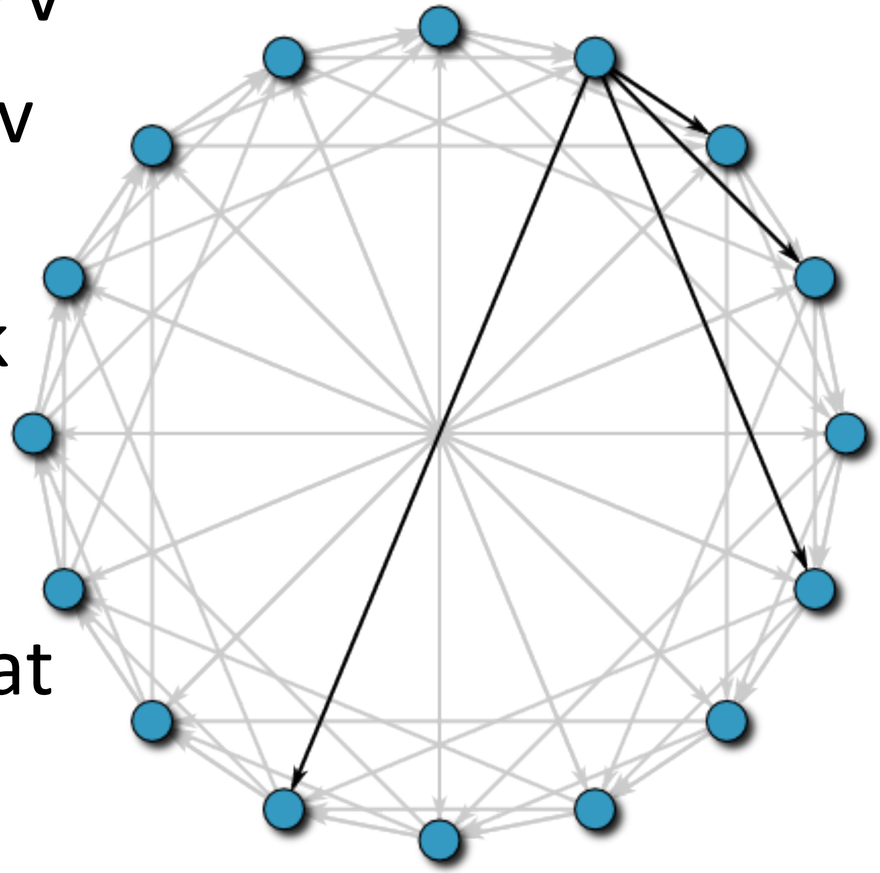
Chord: more efficient search

- At node v, add links to
  - $(2^i+v) \bmod n$
  - Or the first occupied slot after
- Each node has log n additional links
  - O(log n) storage
- Search is efficient



Chord: more efficient search

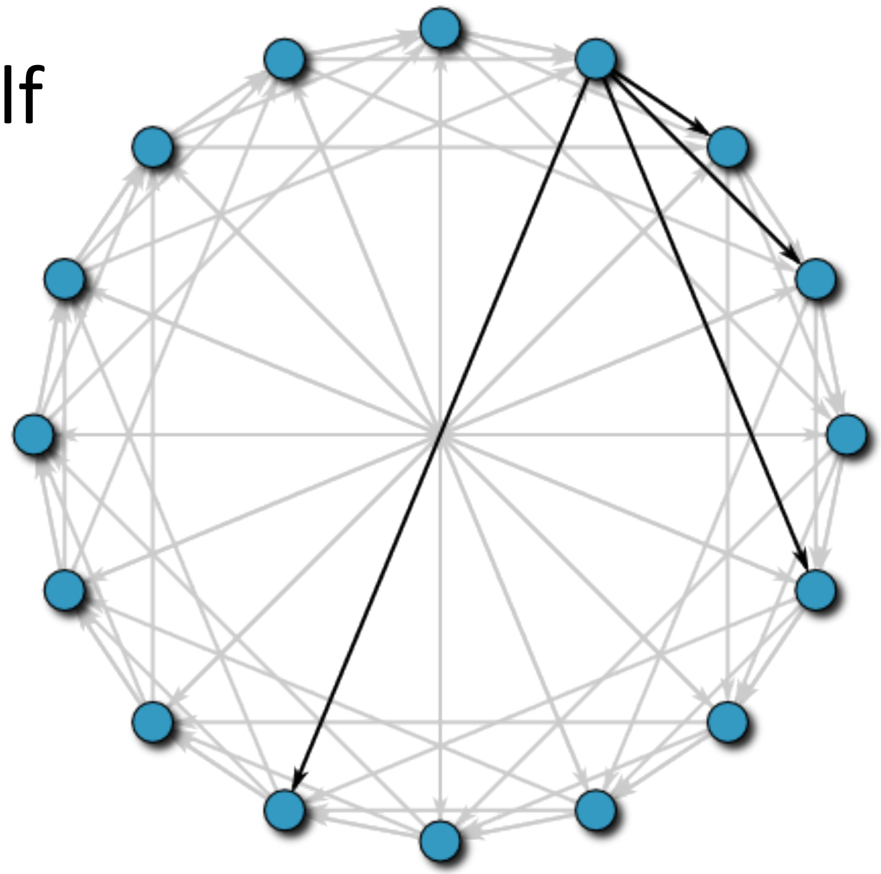
- Suppose we are at node v
- And searching for node v + x
- There is at least one link to a node between v + x/2 and v+x
- The message goes to that node



Chord: more efficient search

- The distance to the destination becomes half in each step

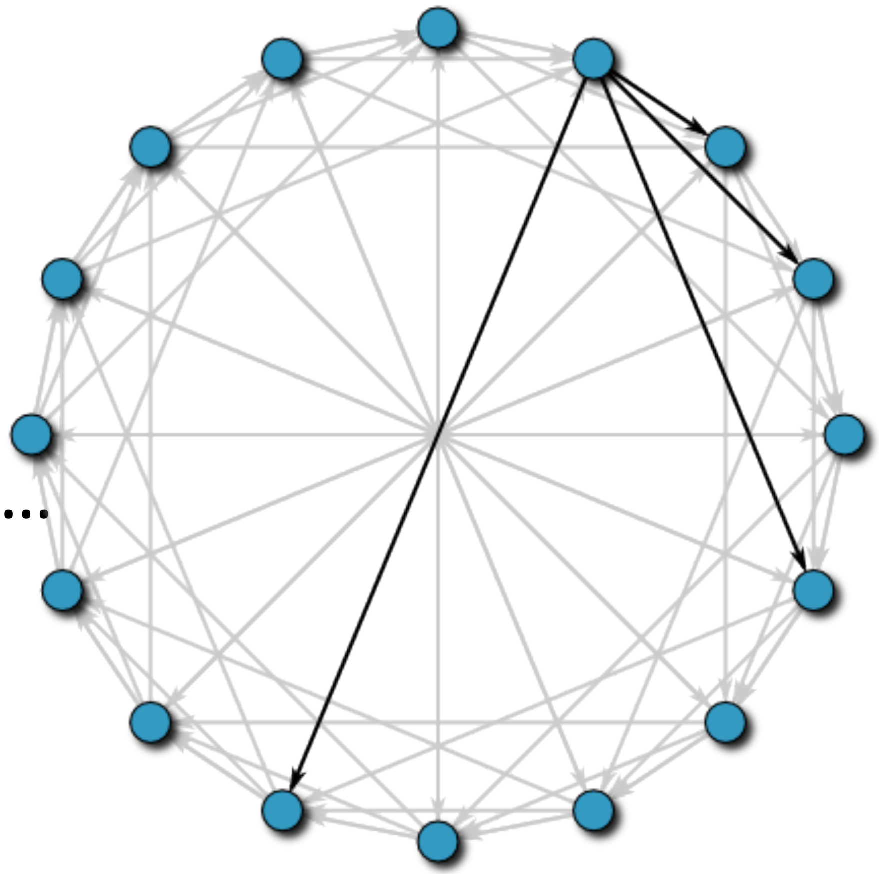
- How many steps does it take?



Chord: more efficient search

- The distance d to the destination becomes half or less in each step
- How many steps does it take?

- The sequence d, d/2, d/4 … converges to 1
- In O(lg n) steps
  – (since d<=n)



Magnet links

- Instead of a .torrent or other descriptor file, use a “link” which eventually gets the file or equivalent data
  – Can be used in any system, currently popular in bittorrent
- Can be of different types
  – Some links direct to the “trackers”, and give the hash of the file
  – Other links lead into a DHT, to find .torrent file/info
    - Assumes the user agent knows how to enter and find content in the overlay network of the DHT
    - Several slightly different formats for magnet links
- Overall, bittorrent is moving toward using DHT magnet links
- But the formats/protocols are not yet standardized or well documented

P2P – Some thoughts

- File sharing has been studied a lot
- Other things much less
- Most p2p designs are old
- Things have changed a lot in recent years
  - More mobile, portable devices
  - Faster networks
  - Bluetooth, nfc, social networks
  - Locations!
- What are good p2p designs in the new environments?

P2P – Can you..

- Design a system for personal storage?
  - Not just copies
  - Needs to be reliable
    - No use if my data is not available when someone else is offline
    - Need multiple replicas
    - Need to keep these replicas updated
  - What other properties?