POLITECNICO
MILANO 1863

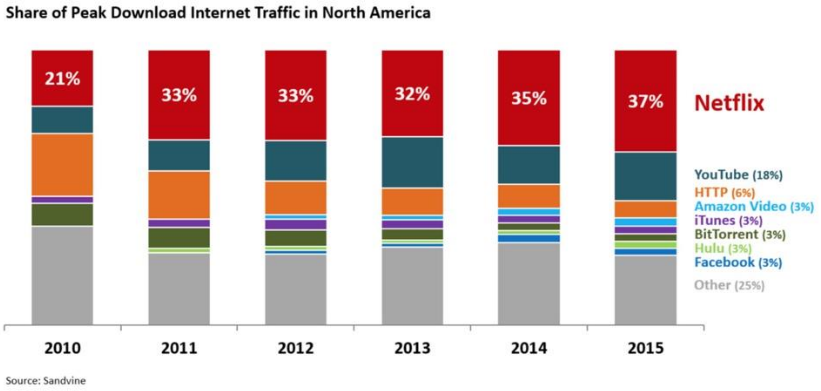
Peer-to-Peer

Alessandro Margara
alessandro.margara@polimi.it
https://margara.faculty.polimi.it

Peer-to-peer

- Architectural paradigm to design distributed systems

- Very popular in the early 2000s
  - Hundreds of file sharing applications developed
  - According to various sources, in 2006 they accounted for more than 2/3 of the entire Internet traffic!
  - Declined over the years due to streaming platforms



Peer-to-peer

- Fundamental difference with respect to architectures based on a client-server approach

"Take advantage of resources at the edges of the network"

(Clay Shirky, O'Reilly)

- What's changed
  - End-host resources have increased dramatically
  - Broadband connectivity now common

From client-server to P2P

Client-server

- Paradigm: a client requests data or a service, a server satisfies the request
- Successful: Web, FTP, Web services
- However
  - Hard to scale
  - Presents a single point of failure
  - Requires administration
  - Leaves some resources unused

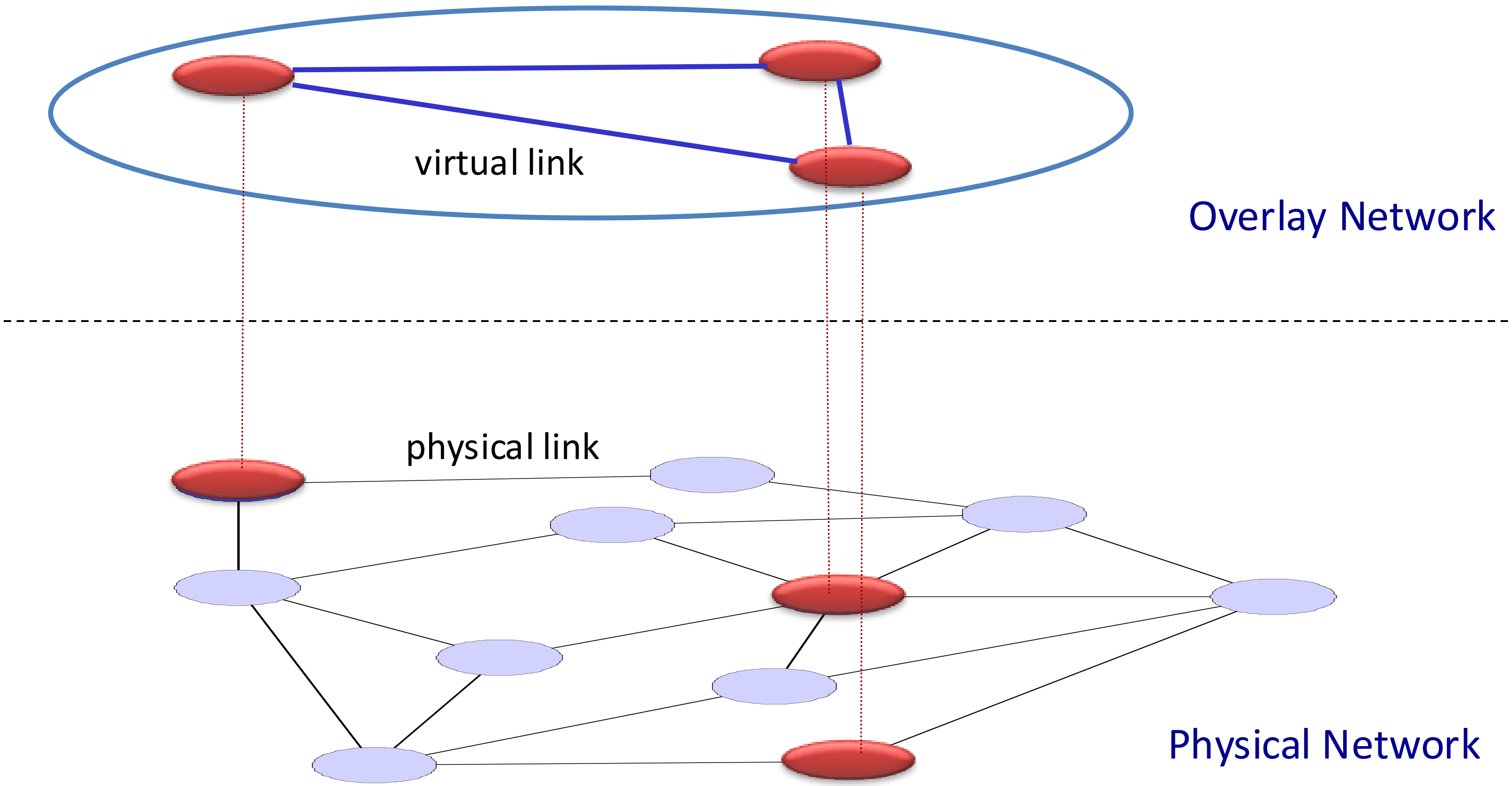
P2P

- Alternative paradigm that promotes the sharing of resources and services through direct exchange between peers
  - Network bandwidth (ad hoc networking, Internet)
  - Processing cycles (Bitcoin)
  - Storage space (Freenet)
  - Data (most of the rest)

Characteristics of P2P

- All nodes are potential users of a service and potential providers of a service
  - Nodes act as servers, clients, as well as routers (ad-hoc communication)
- Each node is independent of the other: no central administration is needed
- Nodes are dynamic: they come and go unpredictably
- Capabilities of nodes are highly variable
- The scale of the system can be Internet-wide
  - No global view of the system
  - Resources are geographically distributed

Overlay network



Our focus: retrieve resources

- Retrieving resources is a fundamental issue in P2P systems due to their inherent geographical distribution
  - Problem: direct requests towards nodes that can answer them in the most efficient way

- We can distinguish two forms of retrieval operations that can be performed on a data repository
  1. Search for something
     - Locate all documents on "Distributed system"
  2. Lookup a specific item
     - Locate a copy of 'RFC 3268'

What to retrieve

- The actual data
  - It can become a burden if query results are routed through the overlay network
  - It is only meaningful in lookup operations
    - Search operations return multiple items

- A reference to the location from where the data can be retrieved
  - It is used both in lookup and in search

Napster

CENTRALIZED SEARCH

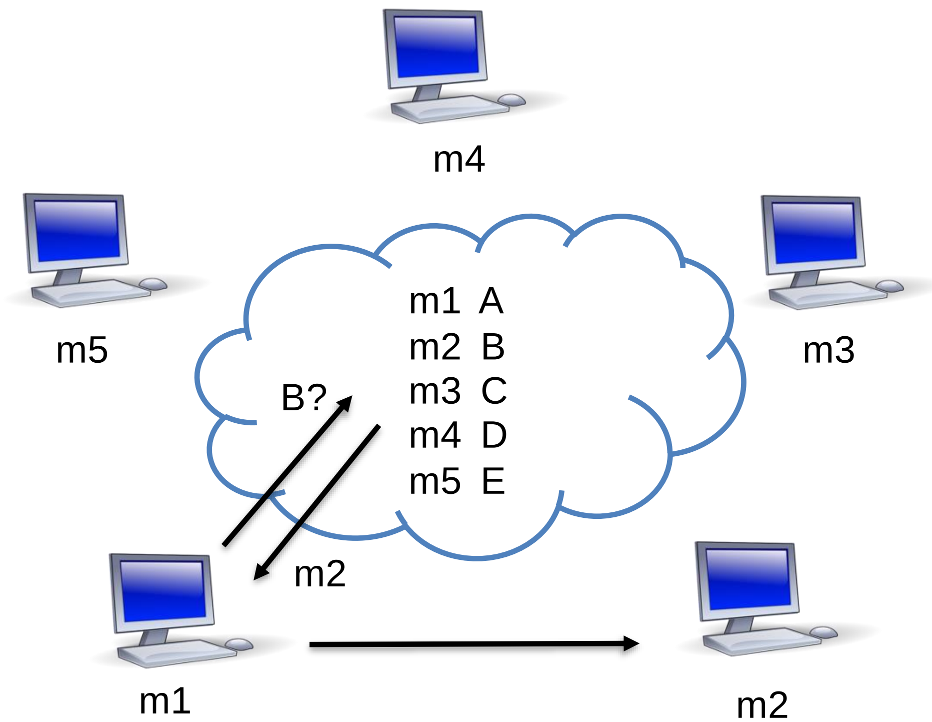
Napster

- It was the first p2p file sharing application
  - A killer application: free music over the Internet

- Key idea: share the storage and bandwidth of individual (home) users

Centralized search: Napster

- Join: clients contact a central server
- Publish: submit list of files to central server
- Search: query the server for someone owning the requested file
- Fetch: get the file directly from peer



A pure P2P system?

- Many researchers argued that Napster is not a P2P system (or at least not a pure one) since it depends on server availability

- Still, Napster allows small computers on edges to contribute
  - All peers are active participants as service provider not only as consumer
  - Even if they rely on a centralized server for lookup

Centralized search: pros and cons

- PROs:
  - Simple
  - Search scope is O(1)

- CONs:
  - Server maintains O(N) State
  - Server does all search processing
  - Single point of failure
  - Single point of “control”

Gnutella

QUERY FLOODING

Query flooding: Gnutella

- No central authority
  - Need to find a connection point in the network

- Gnutella employs the most basic search algorithm
  - Flooding

- Each query is forwarded to all neighbors

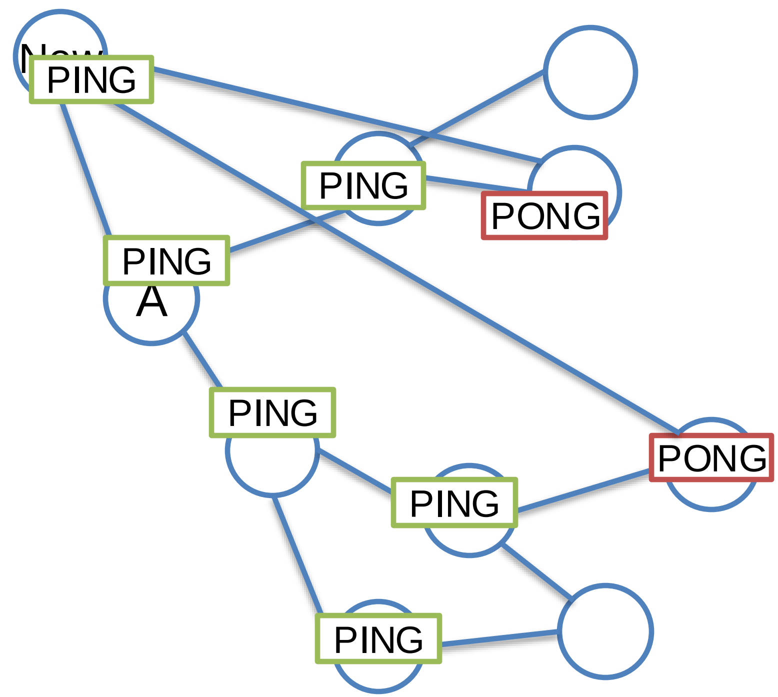
- Propagation is limited by a HopsToLive field in the messages

Query flooding: Gnutella

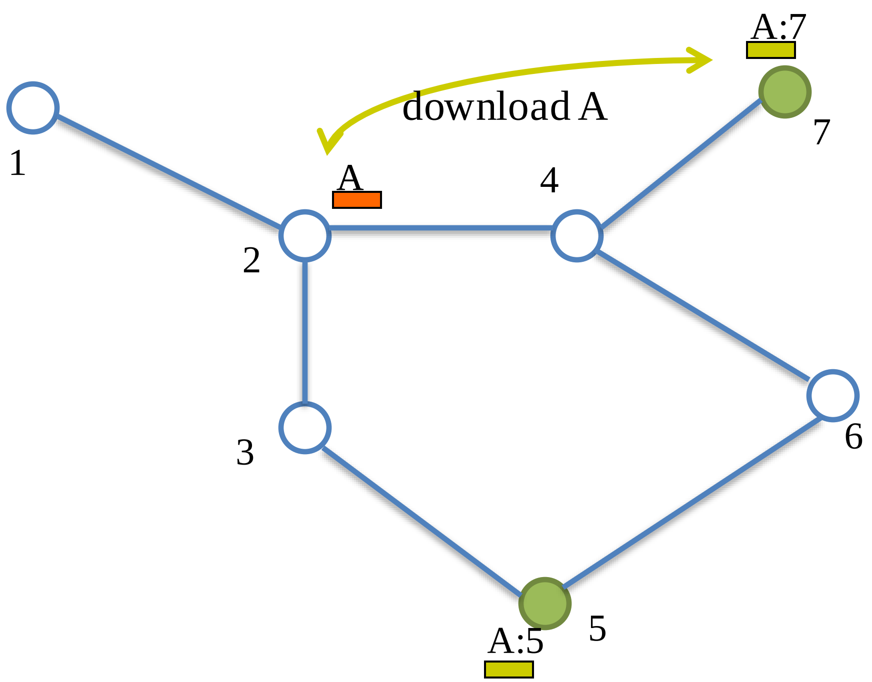
- Join: clients contact a few other nodes
  - They become “neighbors”
- Publish: no need
- Search: ask neighbors, who ask their neighbors, and so on … when/if found, reply to sender
- Fetch: get the file directly from peer

Gnutella: joining the network

- The new node connects to a well known “anchor” node
- Then sends a PING message to discover other nodes
- PONG messages are sent in reply from hosts offering new connections with the new node
- Direct connections are then made to the newly discovered nodes



Query flooding



Query flooding: pros and cons

- PROs:
  - Fully decentralized (no central coordination required)
  - Search cost distributed
  - "Search for S" can be done in many ways, e.g., structured database search, simple text matching, "fuzzy" text matching, etc.

- CONs:
  - Flood" of Requests. If average number of neighbors is C and average HTL is D, each search can cause C×D request messages
  - Search scope is O(N)
  - Search time is O(2D)
  - Nodes leave often, network unstable

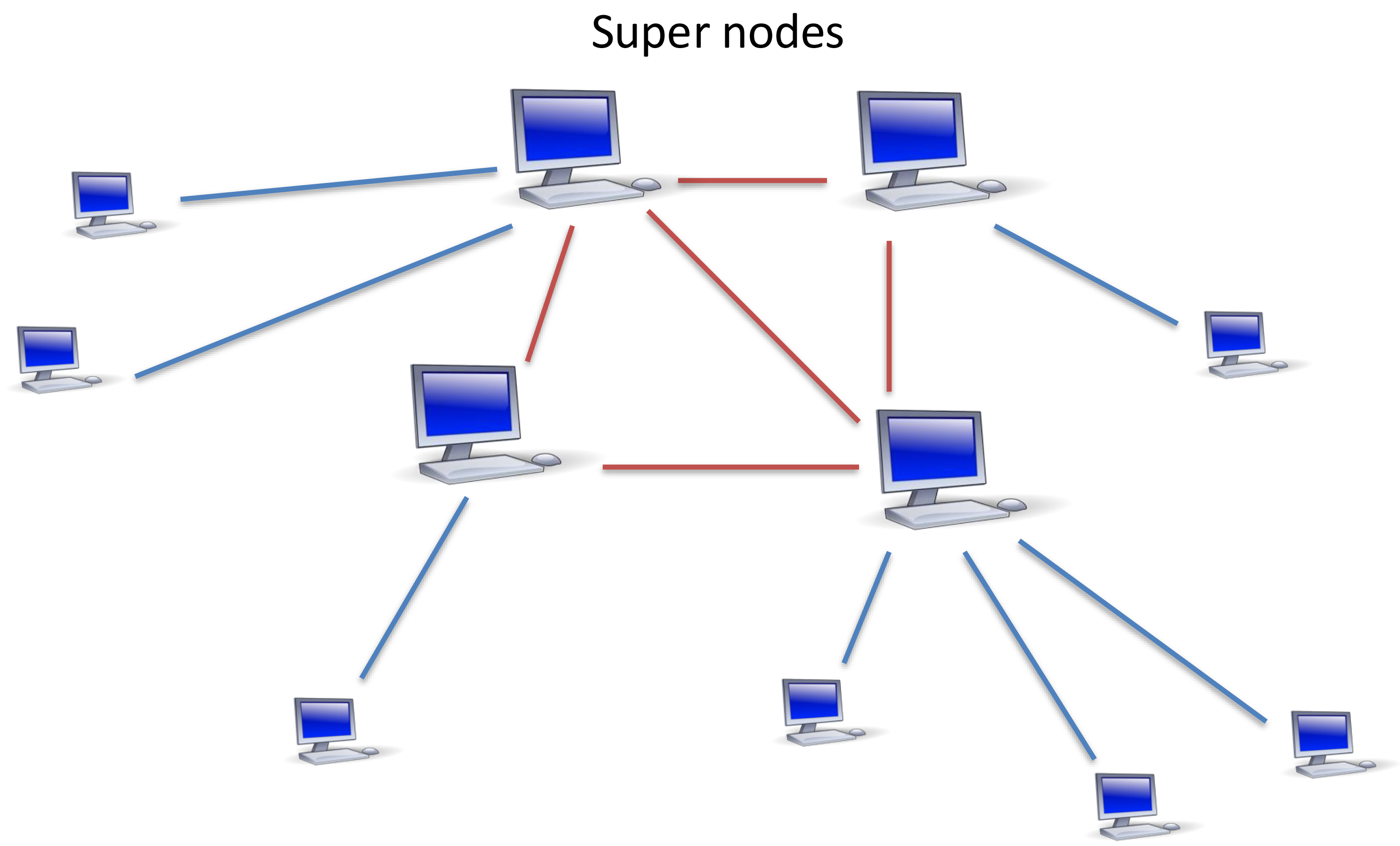
Kazaa

HIERARCHICAL TOPOLOGY

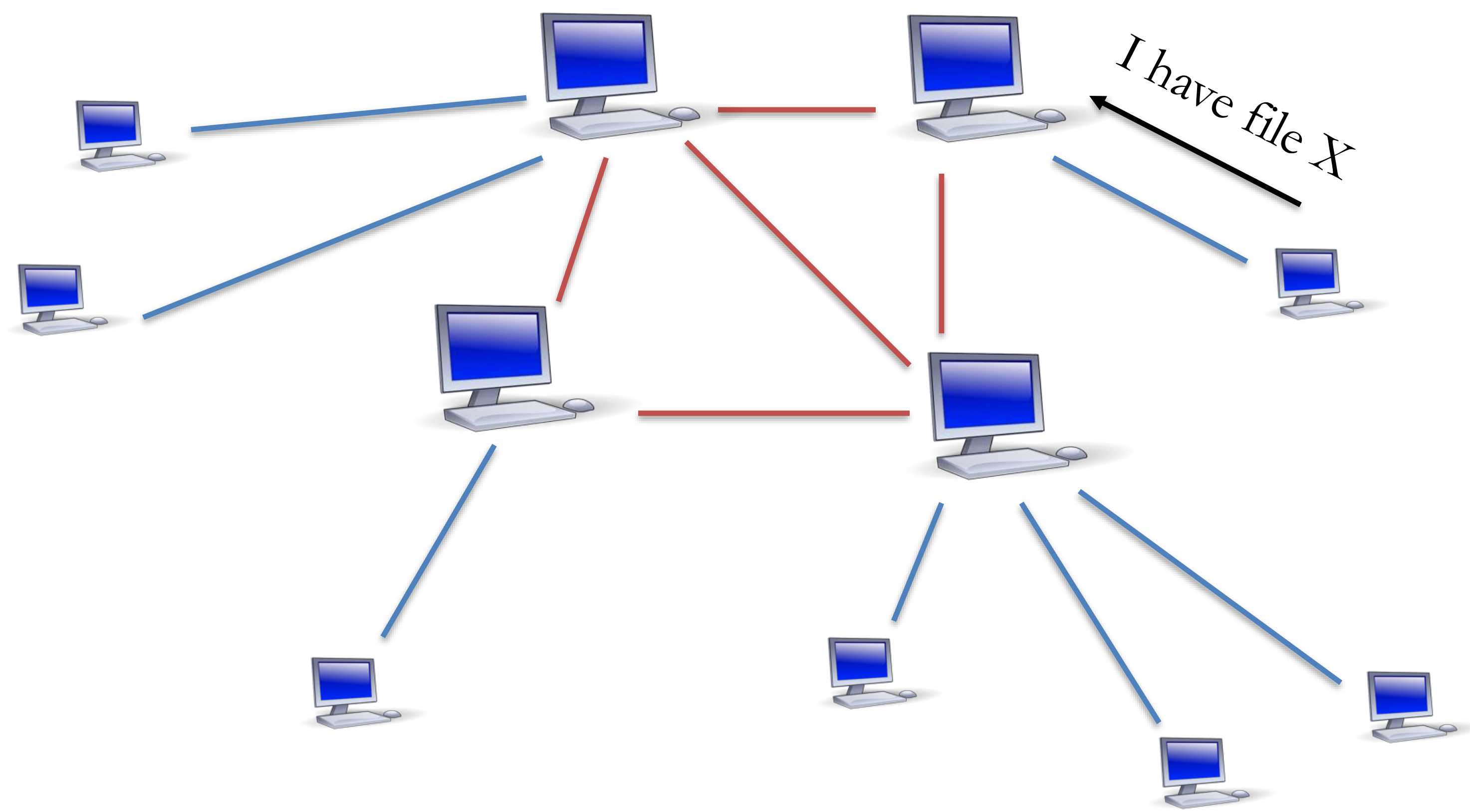
Hierarchical topology: Kazaa

- Distinction between normal nodes and "supernodes"
  - Query flooded among supernodes
  - Normal nodes contact supernodes to perform search
- Join: clients contact a supernode
  - May at some point become supernode themselves
- Publish: send list of files to supernode
- Search: send query to supernode, supernodes flood query amongst themselves
- Fetch: get the file directly from peer(s); can fetch simultaneously from multiple peers

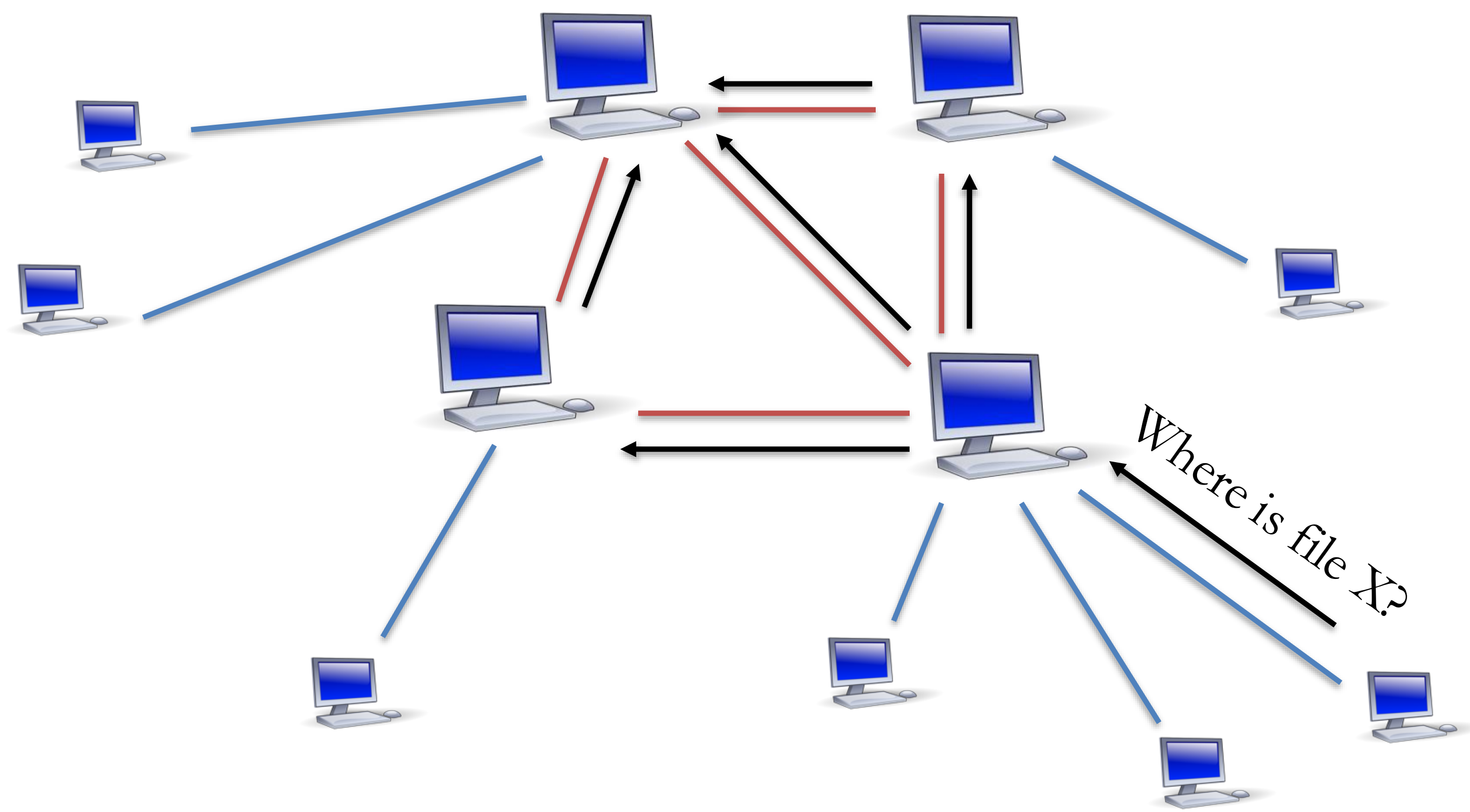
Hierarchical topology



Hierarchical topology: publish



Hierarchical topology: search



KaZaA: pros and cons

- Pros:
  - Tries to consider node heterogeneity
    - Bandwidth
    - Host computational resources
    - Host availability
  - Kazaa rumored to consider network locality

- Cons:
  - Still no real guarantees on search scope or search time

BitTorrent

COLLABORATIVE SYSTEMS

Problem: free riders

- Free riders only download without contributing to the network
- First attempts in Napster were user-based
  - "I refuse to upload if you don't share anything worth"
- Direct Connect required you to share something
  - But no guarantees that you share something valuable
- EMule / KaZaA had credit systems to control the priority of clients
  - Proved easy to circumvent with unofficial clients

Collaborative systems: BitTorrent

- BitTorrent allows many people to download the same file without slowing down everyone else's download

- It does this by having downloaders swap portions of a file with one another, instead of all downloading from a single server
  - This way, each new downloader not only uses up bandwidth but also contributes bandwidth back to the swarm

- Such contributions are encouraged because every client trying to upload to other clients gets the fastest downloads

- It currently is the most used file sharing network (since 2007)

Collaborative systems: BitTorrent

- Join: contact centralized "tracker" server, get a list of peers
- Publish: run a tracker server
- Search: out-of-band
  - E.g., use Google to find a tracker for the file you want
- Fetch: download chunks of the file from your peers
  - Upload chunks you have to them

BitTorrent: terminology

- Torrent: a meta-data file describing the file(s) to be shared
  - Names of the file(s)
  - Size(s)
  - Checksum of all blocks (file is split in fixed-size blocks)
  - Address of the tracker
  - Address of peers
- Seed: a peer that has the complete file and still offers it for upload
- Leech: a peer that has incomplete download
- Swarm: all seeders/leeches together make a swarm
- Tracker: a server that keeps track of seeds and peers in the swarm and gathers statistics
  - When a new peer enters the network, it queries the tracker to obtain a list of peers

BitTorrent: content distribution

- Breaks the file down into smaller fragments (usually 256KB in size)
  - The .torrent holds the SHA1 hash of each fragment to verify data integrity
- Peers contact the tracker to have a list of the peers
- Peers download missing fragments from each other and upload to those who don't have it
- The fragments are not downloaded in sequential order and need to be assembled by the receiving machine
  - When a client needs to choose which segment to request first, it usually adopts a "rarest-first" approach, by identifying the fragment held by the fewest of its peers
  - This tends to keep the number of sources for each segment as high as possible, spreading load

BitTorrent: content distribution

- Clients start uploading what they already have (small fragments) before the whole download is finished

- Once a peer finishes downloading the whole file, it should keep the upload running and become an additional seed in the network

- Everyone can eventually get the complete file as long as there is "one distributed copy" of the file in the network, even if there are no seeds

Where all the magic happens ...

- Choking is a temporal refusal to upload
  - Choking evaluation is performed every 10 seconds
  - Each peer un-chokes a fixed number of peers
  - The decision on which peers to un/choke is based solely on download rate, which is evaluated on a rolling, 20-second average
    - The more I downloaded from you the higher chances are that I upload to you

- A BitTorrent peer has also a (single) “optimistic un-choke”, which is uploaded regardless of the current download rate from it
  - The selected peer rotates every 30s
  - This allows to discover currently unused connections that are better than the ones being used

Game theory

- Employ a "tit-for-tat" sharing strategy
  - "I'll share with you if you share with me"
  - Be optimistic: occasionally let freeloaders download
    - Otherwise, no one would ever start!
    - Also allows you to discover better peers to download from when they reciprocate

- Approximates Pareto efficiency
  - If two peers get poor download rates for the uploads they are providing, they can start uploading to each other and get better download rates than before

BitTorrent: pros and cons

- Pros:
  - Works reasonably well in practice
  - Gives peers incentive to share resources; avoids free-riders

- Cons:
  - Pareto efficiency is a relatively weak condition
  - Central tracker server needed to bootstrap swarm

Freenet

SECURE STORAGE

Freenet

- Freenet (now hyphanet) is a P2P application designed to ensure true freedom of communication over the Internet

- It allows anybody to publish and read information with reasonable anonymity

- Importance of free flow of information, communication & knowledge
  - Democracy assumes a well-informed population
  - Censorship restricts freedom
  - Anonymity protects freedom of speech

Freenet: goals

- Allow practical one-to-many publishing of information
- Provide reasonable anonymity for producers and consumers of information
- Rely on no centralized control or network administration
- Be scalable from tens to hundreds of thousands of users
- Be robust against node failure or malicious attack

Freenet

- Join: clients contact a few other nodes they know about; get a unique node id
- Publish: route file contents toward the node which stores other files whose id is closest to file id
- Search: route query for file id using a steepest-ascent hill-climbing search with backtracking
- Fetch: when query reaches a node containing file id, it returns the file to the sender

Freenet: routing protocol

- Each node in the network stores some information locally
- Nodes also have approximate knowledge of what their neighbors store too
- Request is forwarded to node's "best guess" neighbor unless it has the information locally
- If the information is found within the request's "hops to live", it is passed back through this chain of nodes to the original requestor
- The intermediate nodes store the information in their LRU (least recently used) cache as it passes through

Freenet: publishing

- Insertion of new data can be handled similarly
  - Inserted data is routed in the same way as a request would
    - Search for the id of the data to insert
    - If the id is found, the data is not reinserted (it was already present)
    - Otherwise, the data is sent along the same path as the request
      - This ensures that data is inserted into the network in the same place as requests will look for it

- During searches data is cached along the way
  - This guarantees that data migrates towards interested parties

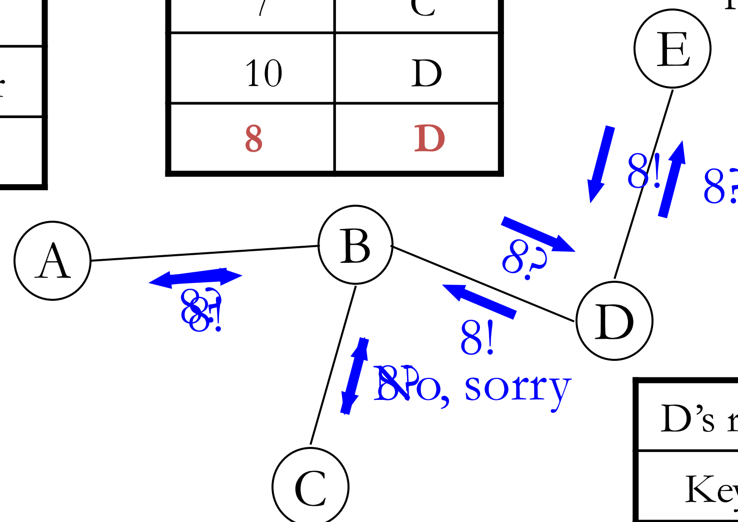
- Each node adds entry to routing table associating the key and the data source (can be random decided)

Freenet: example

A's routing table	
Key	Pointer
8	B

B's routing table	
Key	Pointer
7	C
10	D
8	D

E owns a copy of 8



D's routing table	
Key	Pointer
8	E
...	...

Freenet: routing properties

- "Close" file ids tend to be stored on the same node
  - Why? Publications of similar file ids route toward the same place

- Network tend to be a "small world"
  - Most nodes have relatively few local connections, but a few nodes have many neighbors
  - Well known nodes tend to see more requests and become even better connected

- Consequently, most queries only traverse a small number of hops to find the file

Freenet: caching properties

- Information will tend to migrate towards areas of demand
- Popular information will be more widely cached
- Files prioritized according to popularity
  - Unpopular files deleted when node disk space runs out
- Unrequested information may be lost from the network

Freenet: anonymity & security

- Anonymity
  - Messages (content) are forwarded back and forth
    - Nodes can't tell where a message originated
- Security & censorship resistance
  - The ids of two files should not collide
    - Otherwise, that could be used to create "denial of service" (censorship)
  - Solution: use robust hashing so that the id of a file is directly related to its content

Freenet: anonymity and security

- Link-level encryption
  - Prevents snooping of inter-node messages
  - Messages are quantized to hinder traffic analysis
  - Traffic analysis still possible, but would be a huge task
- Document encryption
  - Prevents node operators from knowing what data they are caching
- Document verification
  - Allow documents in Freenet to be authenticated
  - Facilitate secure date-based publishing
    - The sender publishes a new version

Freenet: pros and cons

- Pros:
  - Intelligent routing makes queries relatively short
  - Search scope small
    - Only nodes along search path involved
    - No flooding
  - Anonymity properties may give you "plausible deniability"

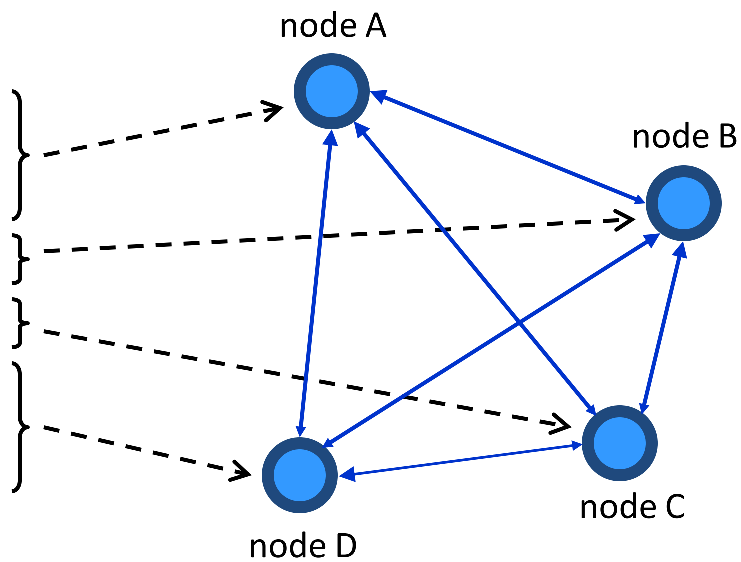
- Cons:
  - Still no provable guarantees!
  - Anonymity features make it hard to measure, debug

Distributed Hash Tables (DHTs)

STRUCTURED TOPOLOGY

DHT: Overview

Key	Value
1	Algorithms
9	Routing
11	DS
12	Peer-to-Peer
21	Networks
22	Grids



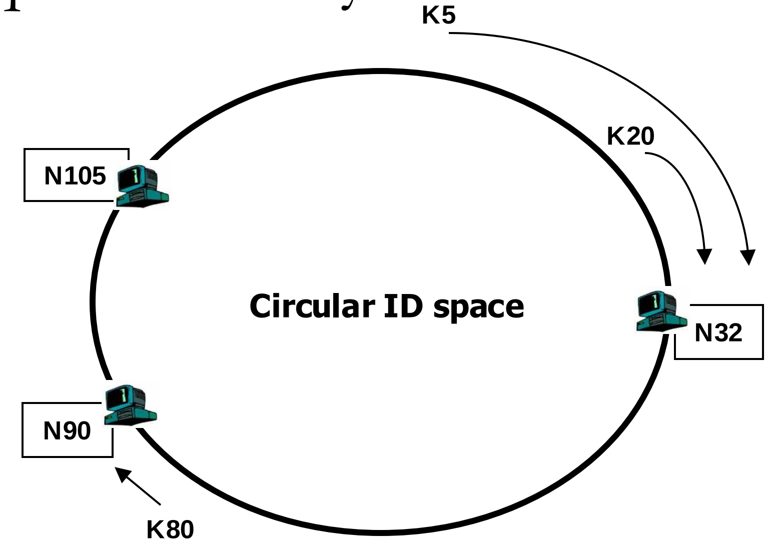
- Abstraction: a distributed "hash-table" (DHT) data structure
  - put(id, item)
  - item = get(id)
    - item can be any resource, such as a reference to a file

DHT: structured overlay routing

- Join: clients contact a “bootstrap” node and integrate into the distributed data structure; get a node id
- Publish: route publication for file id toward a close node id along the data structure
- Search: route a query for file id toward a close node id
  - The data structure guarantees that query will meet the publication
- Fetch:
  - Publication contains actual file → fetch from where query stops
  - Publication contains a reference → fetch from the location indicated in the reference

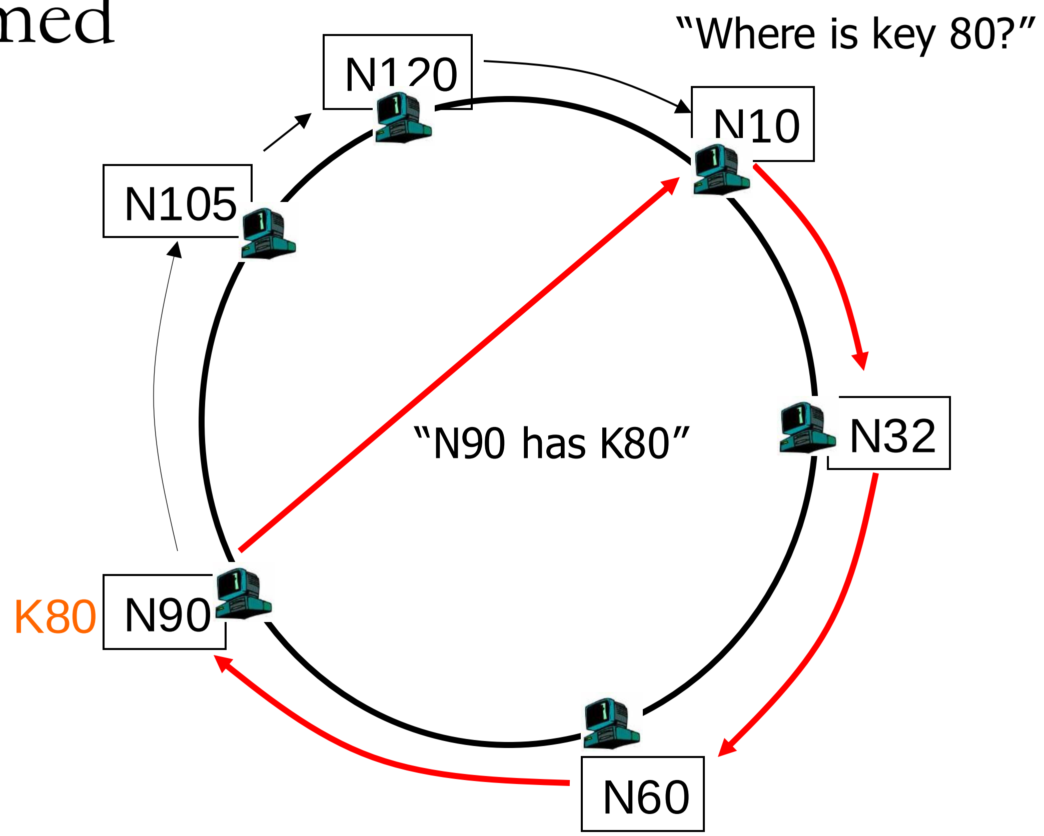
DHT example: Chord

- Nodes and keys are organized in a *logical ring*
  - Each node is assigned a unique *m*-bit identifier
    - Usually, the hash of the IP address
  - Every item is assigned a unique *m*-bit key
    - Usually, the hash of the item
  - The item with key *k* is managed (e.g., stored) by the node with the smallest $id \geq k$ (the *successor*)



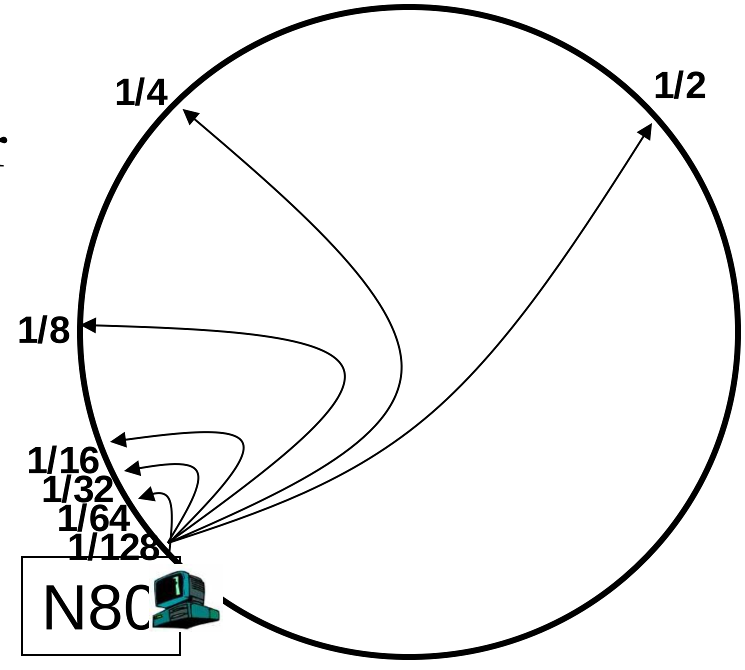
Chord: basic lookup

- Each node keeps track of its successor
- Search is performed linearly



Chord "finger table"

- Each node maintains a finger table with m entries
- Entry i in the finger table of node n is the first node whose id is higher or equal than $n + 2^i$
  - $i = 0 \ldots m-1$
  - In other words, the i^{th} finger points $1/2^{m-i}$ way around the ring



Chord: routing

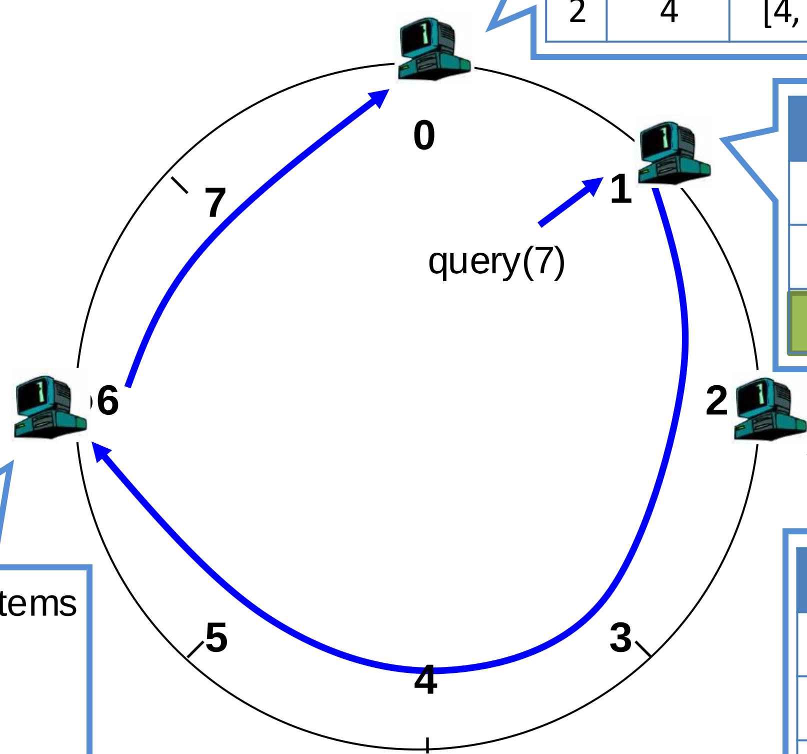
- Upon receiving a query for an item with key k, a node
  - Checks whether it stores the item locally
  - If not, forwards the query to the node in its successor table that is responsible for the interval of keys including k

Chord: routing

i	id+2^i	int.	succ.
0	1	[1, 2)	1
1	2	[2, 4)	2
2	4	[4, 0)	6

Items

7



i	id+2^i	int.	succ.
0	2	[2, 3)	2
1	3	[3, 5)	6
2	5	[5, 1)	6

Items

1

i	id+2^i	int.	succ.
0	7	[7, 0)	0
1	0	[0, 2)	0
2	2	[2, 6)	2

Items

i	id+2^i	int.	succ.
0	3	[3, 4)	6
1	4	[4, 6)	6
2	6	[6, 2)	6

Items

Chord: joining

- Each node also keeps track of its predecessor
  - To allow counter-clockwise routing useful to manage join operations

- When a new node n joins, the following actions must be performed
  1. Initialize the predecessor and fingers of node n
  2. Update the fingers and predecessors of existing nodes to reflect the addition of n

Chord: joining

- To initialize its predecessor and fingers, we assume n knows another node n’ already into the system
  - It uses n’ to initialize its fingers
  - Finger i = successor of node $n+2^i$

Chord: joining

- To update fingers of other nodes, we observe that node n will become the i-th finger of node p if and only if
  1. p precedes n by at least 2^{i-1}
  2. The i-th finger of p succeeds n

- The first node p that meets these two conditions is the immediate predecessor of node $n-2^{i-1}$
  - Search for this node and update it

- We need to do this for each finger i
  - log(N) fingers
  - For each of them we need to perform a lookup that costs log(N)
  - The total cost for joining will be $\log^2(N)$

Chord: stabilization

- The correctness of Chord relies on the correctness of successors pointers
  - This may not be preserved in the case of multiple nodes joining and leaving simultaneously
  - Chord periodically runs a stabilization procedure to ensure this invariant

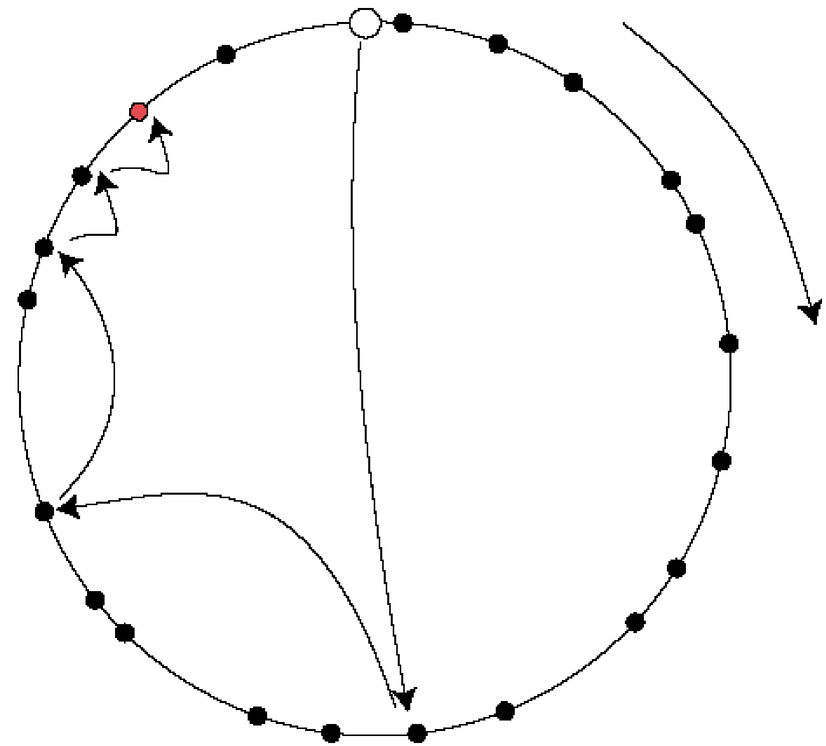
- To stabilize routing information, Chord uses periodic procedures to update successor and finger tables
  - Each node n, for each finger i, periodically lookup for the successor of node $n+2^{i-1}$

Chord: failure and replication

- To increase robustness, each Chord node maintains a list of successors of size R
  - Contains the node's first R successors
  - If the node's immediate successor does not respond, the node contacts the next in the list
  - Resilient even if R-1 successors fail simulataneously

Chord summary

- Routing table size?
  - Log $_N$ fingers
  - With N= 2^m nodes
- Routing time?
  - Each hop expects to 1/2 the distance to the desired id => expect O(log N) hops
- Joining time?
  - With high probability expect O($\log^2$ N) hops



Chord: pros and cons

- Pros:
  - Guaranteed Lookup
  - O(log N) per node state and search scope

- Cons:
  - No one uses them? (only one file sharing app)
  - It is more fragile than unstructured networks
  - Supporting non-exact match search is hard
  - It does not consider physical topology

DHTs

- Chord is just an example, other structures exist
  - Different trade-offs between search scope, information to store, cost to maintain the structure when nodes join and leave

- Examples
  - Kademlia: tree-based structure
  - CAN: d-dimensional space: the number of dimensions can be changed based to better adapt the system to the application at hand

DHTs

- DHTs power the Interplanetary File System (IPFS) project (https://ipfs.tech)
  - Address-based Internet tends to be centralized and vulnerable to single points of failures
  - Idea: use a DHT to create a content-based service/infrastructure

Psaras et al. "Why the Internet needs the interplanetary file system: P2P file sharing would make the Internet more efficient". IEEE Spectrum, 2022.