Distributed Systems
Peer to Peer

Alessandro Sivieri

Dipartimento di Elettronica e Informazione
Politecnico di Milano, Italy

sivieri@elet.polimi.it

http://corsi.dei.polimi.it/distsys

Slides based on previous works by Alessandro Margara

Contents

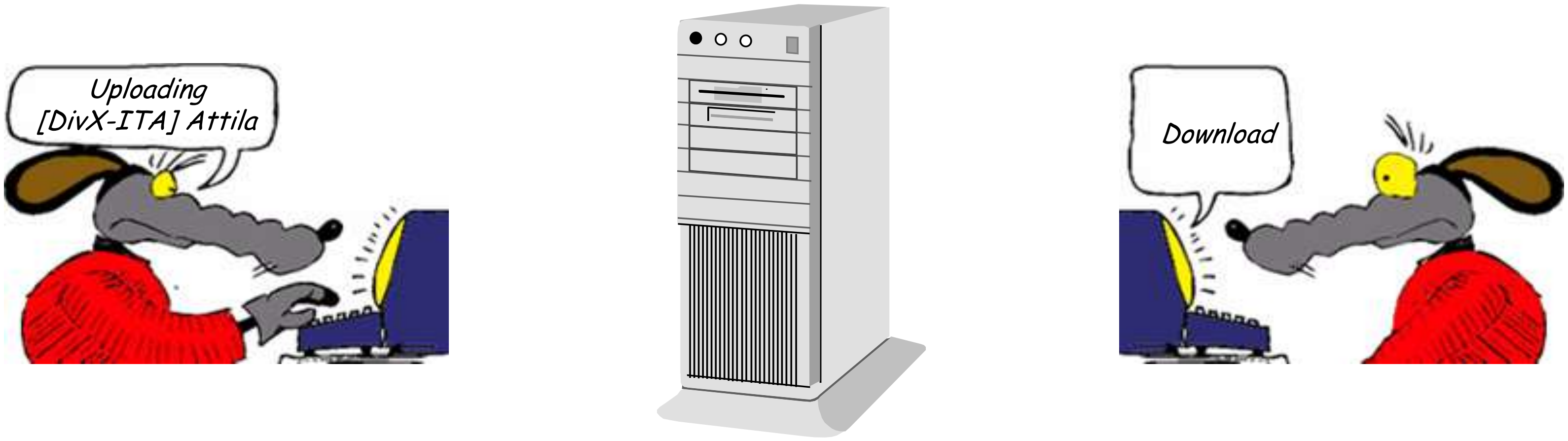
- Introduction to P2P
- Centralized Database
  - Napster
- Query Flooding
  - Gnutella
- Intelligent Query Flooding
  - KaZaA
- Swarming
  - BitTorrent
- Unstructured Overlay Routing
  - Freenet
- Structured Overlay Routing
  - Distributed Hash Tables (Chord, Kademlia, CAN, PASTRY)

Why P2P

- Quickly grown in popularity
  - Hundreds of file sharing applications
  - About 31% of all Internet traffic is due to BitTorrent (2004)
  - Audio/Video transfer now dominates traffic on the Internet
    - In Jan 2006 overall p2p traffic (mostly audio/video) was 70% of all Internet traffic
- But what is P2P?
  - Searching or location? -- DNS, Google!
  - Computers “Peering”? -- Server Clusters, IRC Networks, Internet Routing!
  - Clients with no servers? -- Doom, Quake!
- First, let’s say what is not...

or at least, it is not only that

Why P2P is different



- Fundamental difference:

> *"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 C/S to P2P

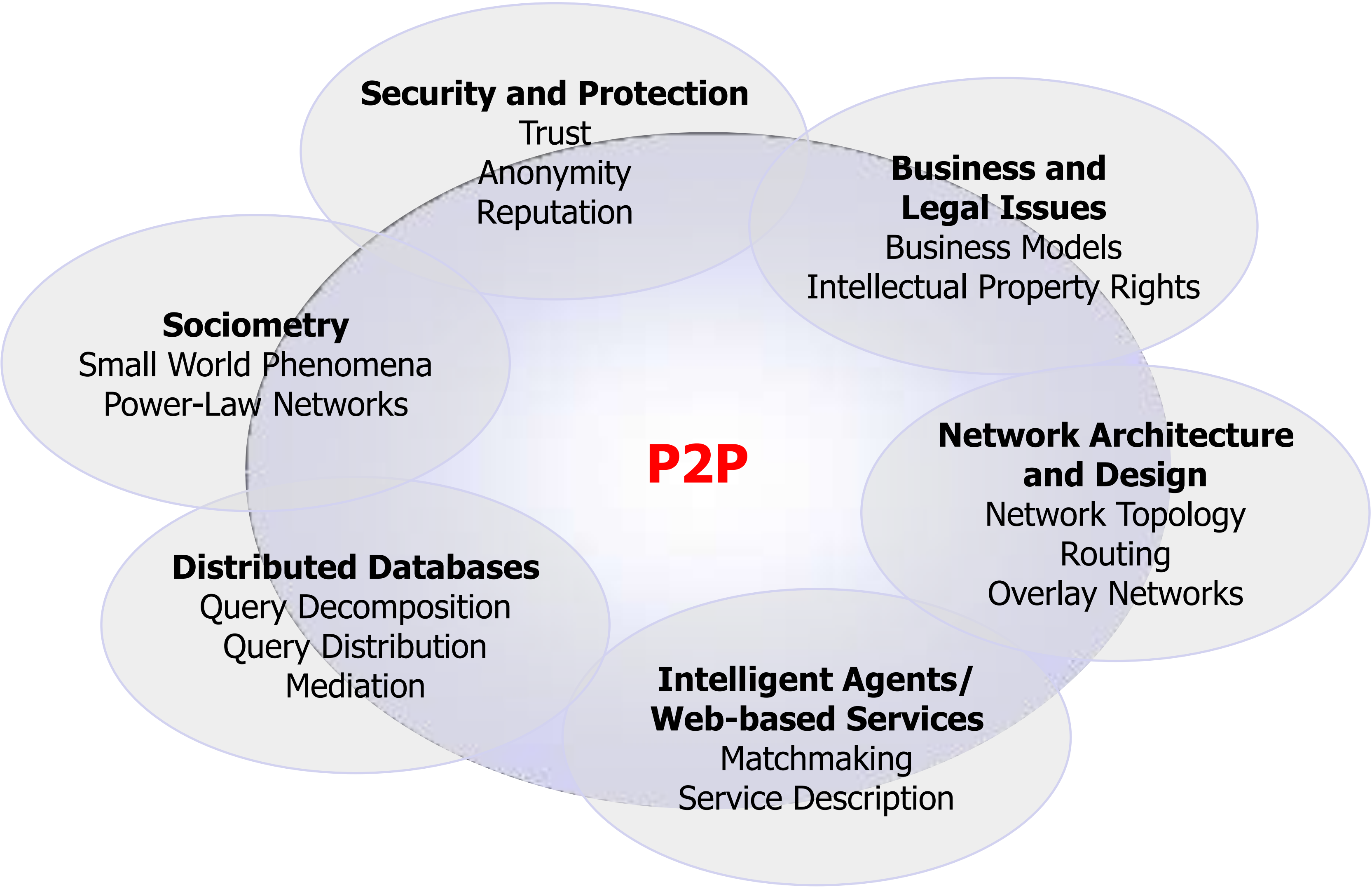
- C/S
  - Most commonly used paradigm in today’s Internet
  - A client requests data or a service, a server satisfies the request
  - Successful: Web, FTP, Web Services
  - However:
    - Hard to scale,
    - Presents single point of failure
    - Requires administration
    - Leaves some resources unused

- P2P
  - It is a paradigm: It specifies how, not what
  - It promotes the sharing of resources and services through direct exchange between peers
  - Resources can be:
    - Processing cycles (SETI@home)
    - Collaborative work (ICQ, Skype, Waste)
    - Storage space (Freenet)
    - Network bandwidth (ad hoc networking, Internet)
    - Data (most of the rest)

Characteristics of peer-to-peer

- 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 is Internet-wide
  - No global view of the system
  - Resources are geographically distributed

P2P Issues



Security and Protection
Trust
Anonymity
Reputation

Business and Legal Issues
Business Models
Intellectual Property Rights

Sociometry
Small World Phenomena
Power-Law Networks

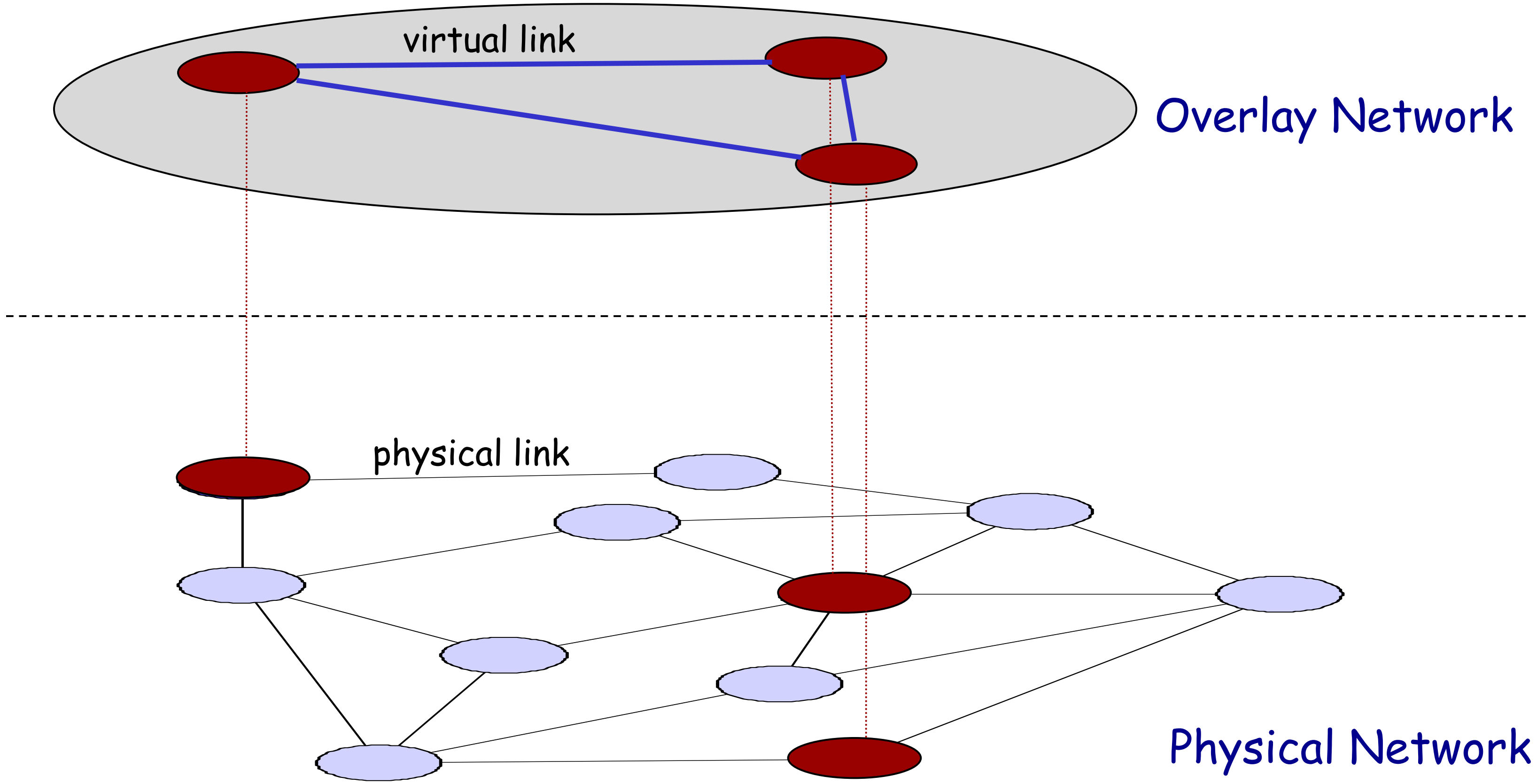
P2P

Network Architecture and Design
Network Topology
Routing
Overlay Networks

Distributed Databases
Query Decomposition
Query Distribution
Mediation

Intelligent Agents/ Web-based Services
Matchmaking
Service Description

Overlay network



What we will concentrate on

- Retrieving resources is a fundamental issue in peer-to-peer systems due to their inherent geographical distribution
- The problem is to direct queries towards nodes that can answer them in the most efficient way

Data retrieval: Lookup and search

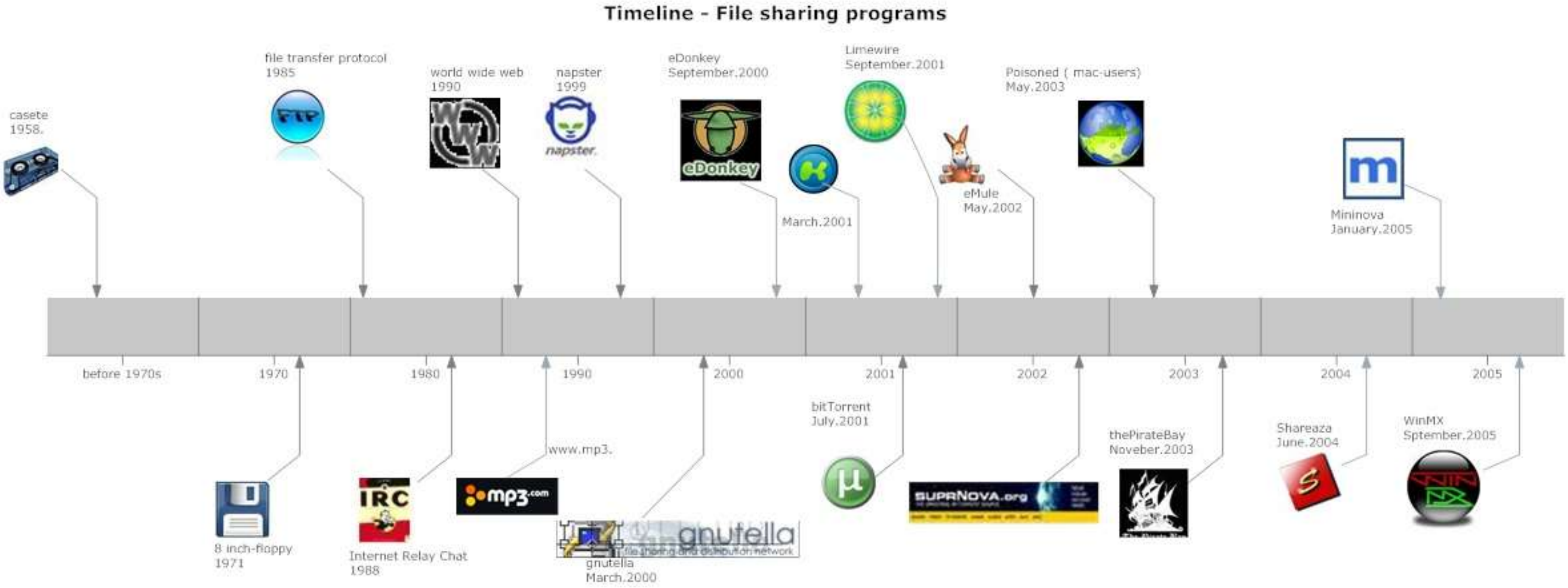
- We can distinguish two forms of retrieval operations that can be performed on a data repository
  - Search for something
    - Locate all documents on 'Networking'
    - Locate all papers which contain the expression 'Peer-to-Peer'
    - …
  - Lookup a specific item
    - Locate a copy of 'RFC 3268'

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

Politecnico
di Milano

Brief timeline (up to 2005)



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
- In 2000, 50M users downloaded Napster client. Napster had a peak of 7TB traffic a day with 1.5 million simultaneous users

http://www.napster.com

Napster History

- 1987: MP3 format developed by Karlheinz Brandenburg of Fraunhofer Gesellschaft. “CD ripping” now feasible
- 1999: Shawn Fanning develops Napster, believing he has “bypassed” copyright law. Napster has >25M users in its first year
- Dec., 1999: RIAA sues Napster for “contributory and vicarious” copyright infringement
- April, 2000: Metallica sues Napster, Yale, Indiana Univ., and USC. (Yale bans the use of Napster within a week.)

Napster History (2)

- July, 2000: US District Judge Patel grants RIAA's request for an injunction. The injunction is temporarily stayed soon thereafter
- October, 2000: Napster announces a partnership with Bertlesmann AG (one of the "major labels" in the industry whose trade association is suing it!)
- January, 2001: Napster and Bertlesmann say that they will roll out a "subscription service" by "early summer" and will use "DRM technology"
- February, 2001: Ninth Circuit upholds lower court's findings that Napster is guilty of contributory and vicarious infringement

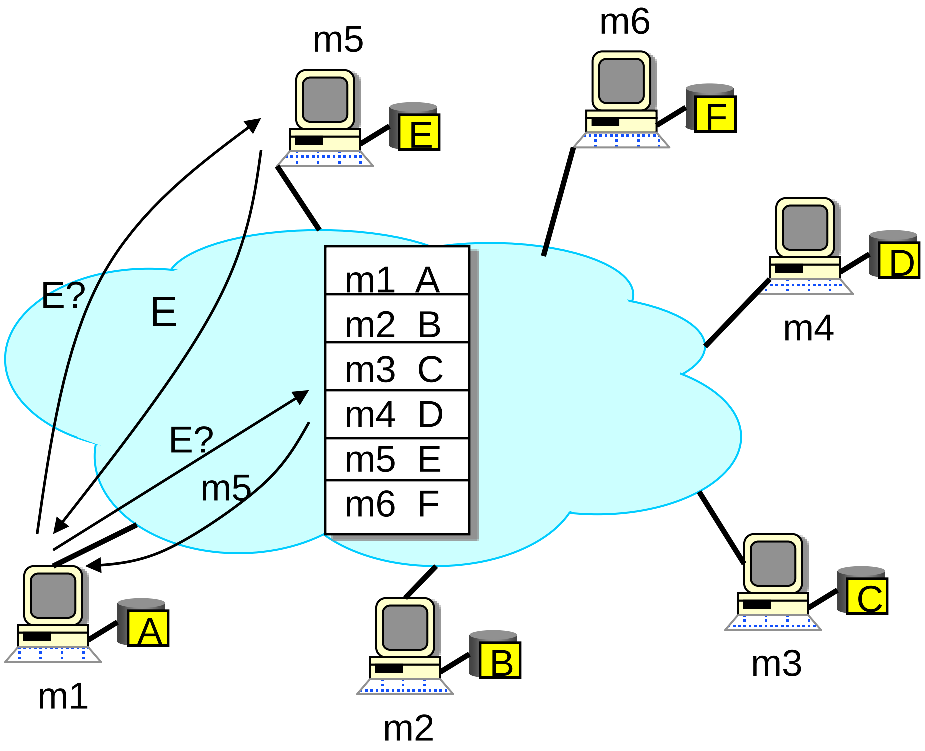
Napster History (3)

- Summer, 2001: Napster and Bertelsmann fail to roll out subscription service
- September, 2001: Napster reaches a settlement with music publishers (but not with RIAA record labels). However, CNET.com reports the number of users has “dropped from tens of millions…to almost zero”

Napster, R.I.P. !

Napster

- Join: On startup, client contacts central server
- Publish: Reports list of files to central server
- Search: Query the server => return someone that stores the requested file
- Fetch: Get the file directly from peer



m1	A
m2	B
m3	C
m4	D
m5	E
m6	F

A pure P2P system ?

- Many researchers argued that Napster is not a peer-to-peer system (or at least not a pure one) since it depends on server availability
- IMHO, they confuse topology with function
- Napster is a P2P system since allows small computers on edges to contribute
  - All peers are active participants as service provider not only as consumer

Napster: Pro and cons

- ☑ PROs:
  - ☑ Simple
  - ☑ Search scope is O(1)
- ☒ CONs:
  - ☒ Server maintains O(N) State
  - ☒ Server does all search processing
  - ☒ Single point of failure
  - ☒ Napster "in control" (freedom is an illusion)

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 HopToLive field in the messages

http://www.gnutella.com

Gnutella History

- Gnutella was written by Justin Frankel, the 21-year-old founder of Nullsoft, in just 14 days
- Nullsoft (maker of WinAmp) posted Gnutella on the Web, March 1999
- Nullsoft acquired by AOL, June 1999
- A day later AOL yanked Gnutella, at the request of Time Warner
- People had already downloaded and shared the program
- Gnutella continues today, run by independent programmers

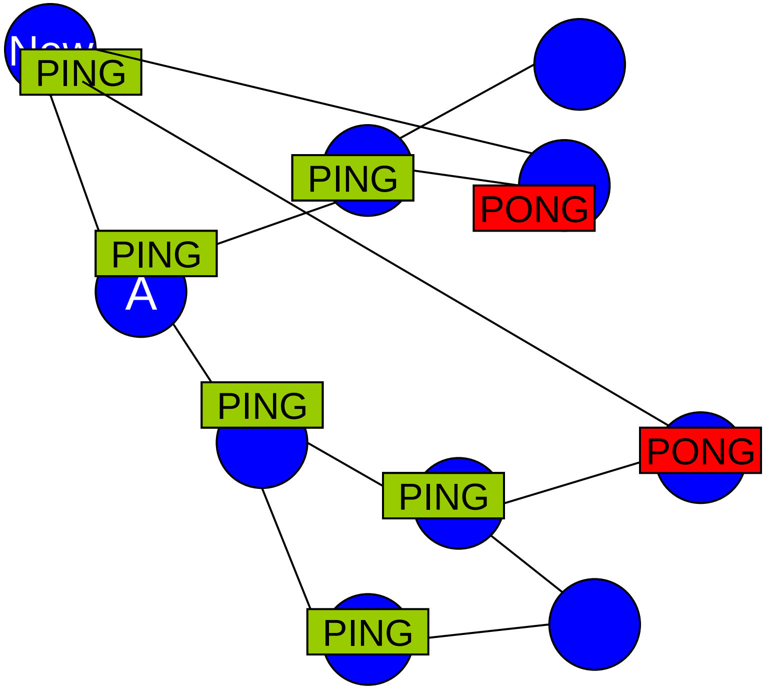
Gnutella: Overview

- Query Flooding:
  - **Join**: On startup, client contacts a few other nodes; these become its “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

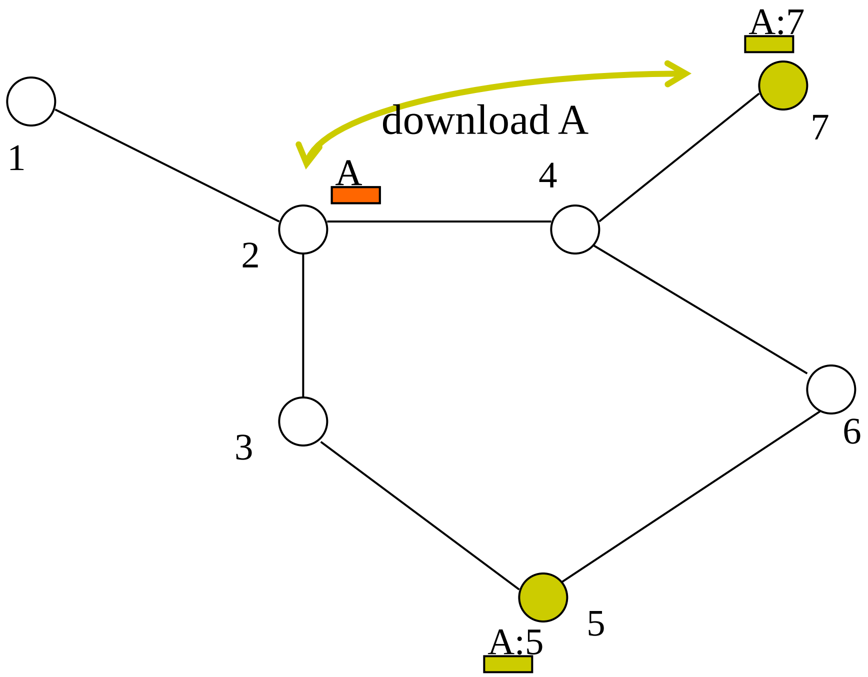
Politecnico
di Milano

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



Gnutella search mechanism



Gnutella: Pro and cons

- ☑ PROs:
  - ☑ Fully de-centralized (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 TTL 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

- Peers connect directly to each other; content is distributed and there is no central server
- Search requests are sent to the “nearest” supernode, which tries to locate the content; if it fails, the request is sent to other supernodes
- Any node running Kazaa with a good Internet connection can become a supernode
  - Other Kazaa users upload lists of shared files to neighboring supernodes
  - Supernodes facilitate search but do not host content; peers connect directly to download files
  - Supernode status is controlled by the software based on user settings (permission to become a supernode, bandwidth restrictions, etc.)

http://www.kazaa.com

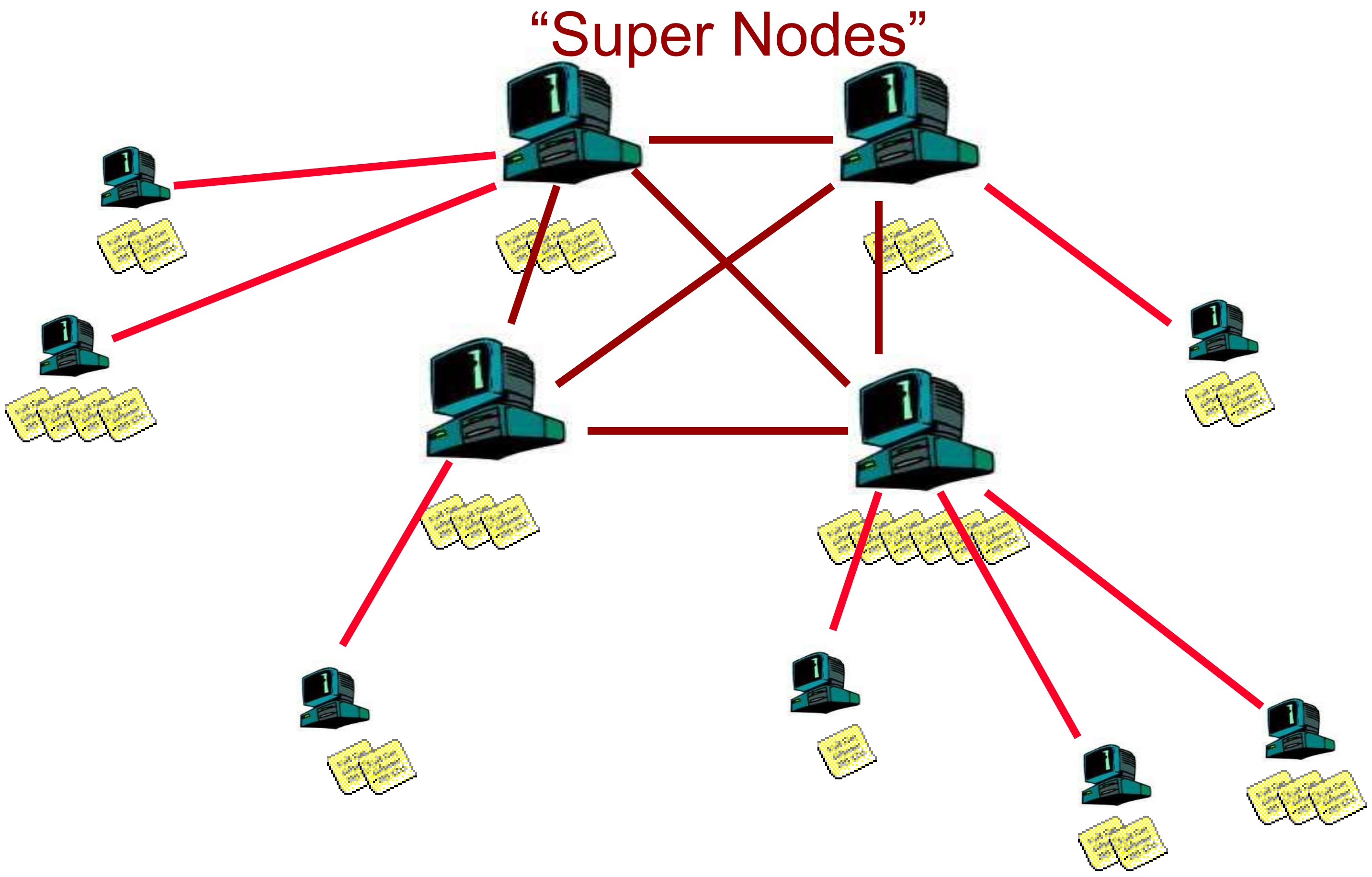
KaZaA: History

- In 2001, KaZaA created by Dutch company Kazaa BV
- Kazaa Media Desktop now is produced by Sharman Networks, Ltd., “a consortium of private investors with multimedia interests” (see company website). Based in Australia with offices in Europe
- Single network called FastTrack used by other clients as well: Morpheus, giFT, etc.
- Eventually protocol changed so other clients could no longer talk to it
- Most popular file sharing network up until 2004

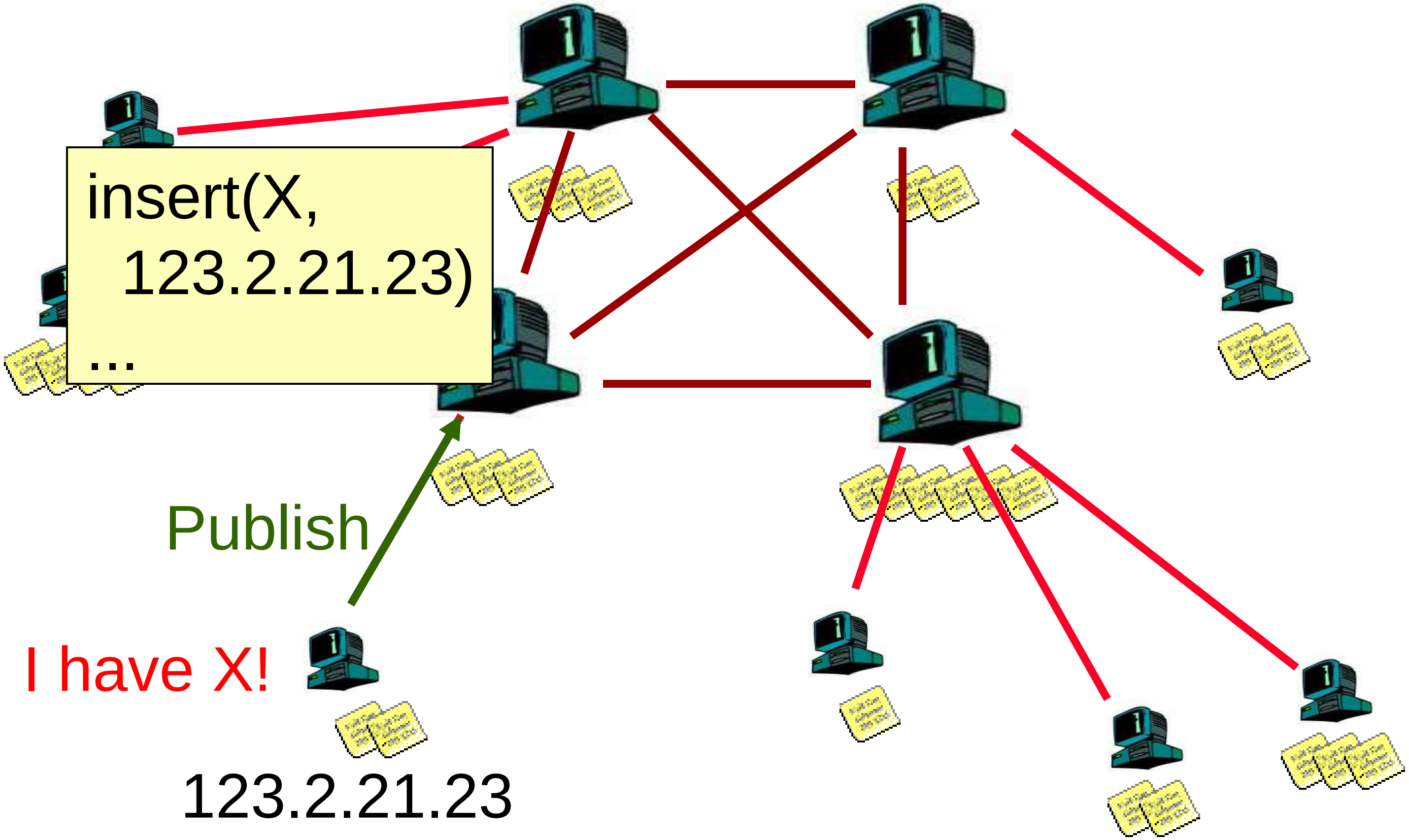
KaZaA: Overview

- “Smart” Query Flooding:
  - **Join**: On startup, client contacts a “supernode” ... may at some point become one itself
  - **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

KaZaA: Network Design



KaZaA: File insert



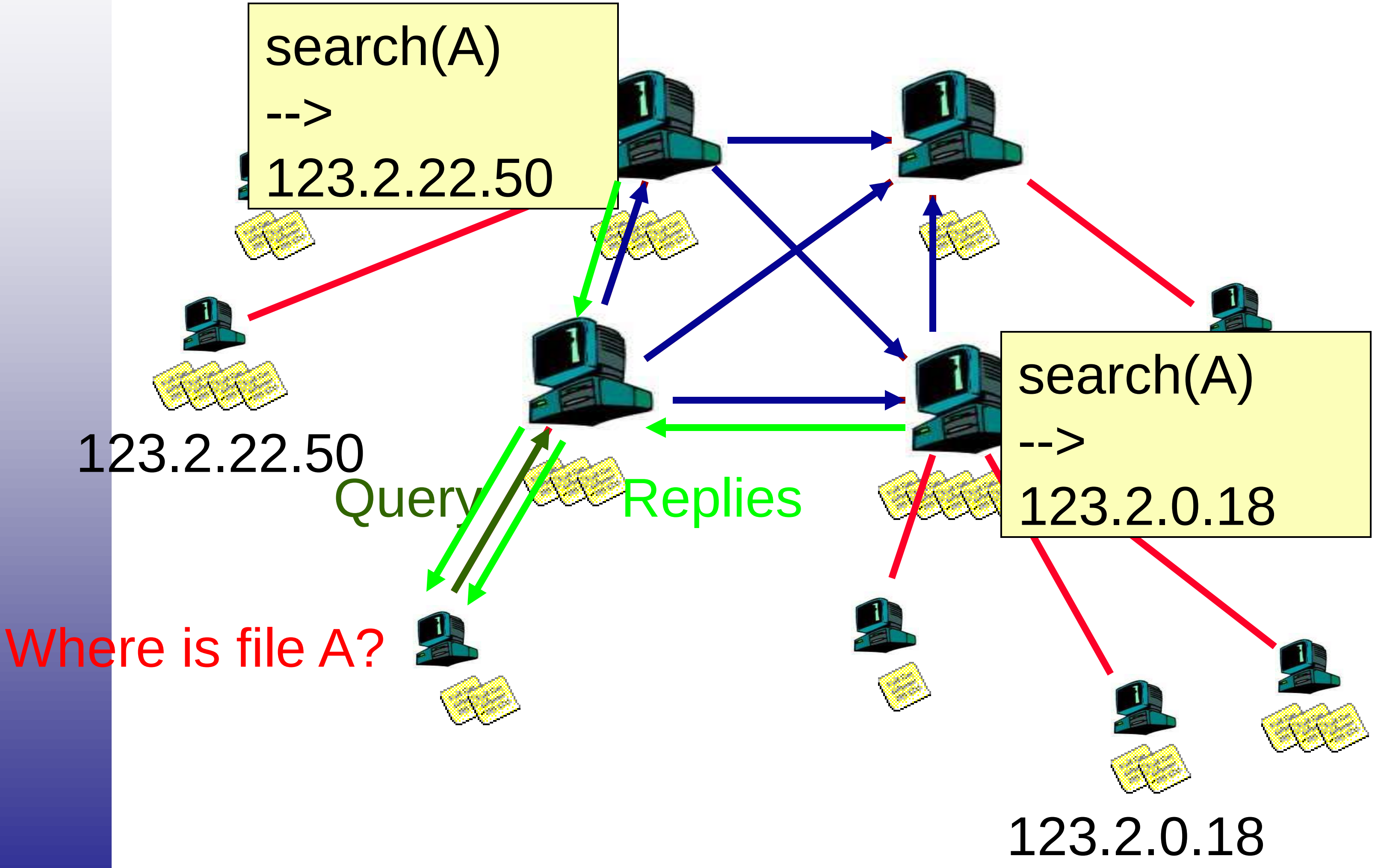
insert(X,
 123.2.21.23)
...

Publish

I have X!

123.2.21.23

KaZaA: File search



search(A)
-->
123.2.22.50

search(A)
-->
123.2.0.18

123.2.22.50

Query

Replies

Where is file A?

123.2.0.18

KaZaA: Fetching

- More than one node may have requested file...
- How to tell?
  - Must be able to distinguish identical files
  - Not necessarily same filename
  - Same filename not necessarily same file
- Use hash of file
  - KaZaA uses UUHash: Fast, but not secure (easy to change large part of the file without changing the hash)
  - Alternatives: MD5, SHA-1
- How to fetch?
  - Get bytes [0..1000] from A, [1001...2000] from B

KaZaA: Pros and cons

- ☑ Pros:
  - ☑ Tries to take into account node heterogeneity:
    - ☑ Bandwidth
    - ☑ Host Computational Resources
    - ☑ Host Availability (?)
  - ☑ Rumored to take into account network locality
- ☒ Cons:
  - ☒ Mechanisms easy to circumvent
  - ☒ Still no real guarantees on search scope or search time
  - ☒ Proprietary Protocol

Gnutella reloaded

- Originally, Gnutella peers connected to other peers randomly. This was ok with broadband users but not modem users
- Solution: provide a hierarchy to the network
  - Ultrapeer
    - Acts as a proxy
    - Reduces the number of nodes in the network involved in message handling and routing
    - Forwards a query to the leaf only if it believes the leaf can answer it
    - Connencted to other Ultrapeers and other normal Gnutella hosts
  - Leaf
    - Normal Gnutella hosts
    - Leaves a small number of connections open (usually to ultrapeers)
    - Usually connects to 3 ultrapeers at a time

eDonkey network

- Decentralized network, no central server
- Network servers are used as communication hubs
  - Clients update server lists regularly
- The network protocol is not documented
  - It is described in practice by the interaction between current clients and servers
- *eserver* is proprietary freeware
- *eDonkey2000* was the original freeware client
- *eMule* is the current open source, cross platform client
- Most popular network up until 2007

eDonkey history

- eDonkey2000 has been developed by US company MetaMachine and released in 2000
- Better than Napster: users could download chunks of files from different peers, incrementing performances
- MetaMachine sued by RIAA in 2005
  - So they discontinued the program
- In 2006 they reached an agreement with RIAA and converted their clients to a “legal” system…
- … but it was too late!

eDonkey network internals

- Identification of files based on hashes
  - In particular, MD4 is computed for each chunk of files (chunks being of fixed size of 9500 KB)
  - Each file has then a hash obtained by concatenating partial hashes and hashing the result
    - Collisions are believed to be quite rare
- Search based on file names or file characteristics
  - Plus the possibility of using a eD2k link
- For avoiding having to use central servers, eDonkey network started to support other standards, like KAD

Free riders

- Free riders only download without contributing to the network.
- First attempts in Napster were user-based
  - *"I refuse to upload you if you don't share anything worth"*
- Direct Connect++ requires you explicitly but, of course, it does not guarantee, share something valuable (unless hub administrator takes care of)
- EMule has a credits system, which stores the amount of data each client has uploaded and downloaded from every other client and gives clients that have more credits a higher priority in the queues
- Kazaa in 2002 introduces the Participation Level that increases when you upload and decrease when you download. Then when you upload a file to someone else the person with the highest Participation Level gets it first, then they upload it on to the person with the next highest Participation Level, and so on.
  - Unfortunately the system adopted by KaZaA is flawed as it relies on the client accurately reporting their Participation Level and therefore it is easy to cheat with the many "unofficial" clients

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)

http://bittorrent.org

BitTorrent: History

- In 2002, B. Cohen debuted BitTorrent (written in Python)
- Key Motivation:
  - Popularity exhibits temporal locality (Flash Crowds)
  - E.g., Slashdot effect, CNN on 9/11, new movie/game release
- Focused on *Efficient Fetching*, not *Searching*:
  - Distribute the *same* file to all peers
  - Single publisher, multiple downloaders
- Has some “real” publishers:
  - Blizzard Entertainment using it to distribute the beta of their new game
  - Linux Distros

BitTorrent: Overview

- Swarming:
  - **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.
  A *.torrent* file holds:
  - 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 provide 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
- 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 unchokes a fixed number of peers (default = 4)
  - 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 unchoke*', which is uploaded regardless of the current download rate from it. This peer rotates every 30s
  - This allows to discover currently unused connections that are better than the ones being used

BitTorrent: 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
  - Similar to cooperative move in Prisoner's Dilemma
- 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

The Freenet Project

- Freenet is a P2P application designed to ensure true freedom of communication over the Internet
- It allows anybody to publish and read information with complete anonymity
- Importance of free flow of information, communication & knowledge
  - Democracy assumes a well-informed population
  - Censorship restricts freedom
  - Anonymity protects freedom of speech

http://freenet.org

Freenet: Goals

The Freenet Project

- 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: History

The Freenet Project

- Originally, it was developed in 1999 by Ian Clarke while a student at the University of Edinburgh, now it is still supervised by Ian Clarke, though many other people contribute to the project
- ~60 known "Freesites"
- Third-party applications under active use and development
- ~3,000 downloads/day
- Convenient installation on Windows and Linux

It's a small world after all

- The Milgram Study
  - Letters given to 150 random starting people in Omaha, NB, and Wichita, KA
  - Letters had name and address of a target person in Cambridge, MA
  - Participants instructed to send letter to a friend they know on a first-basis most likely to know target person
  - Participants asked to write their name on the letter to prevent loops and track the letter's path

Results of Milgram study

- Median of 5 intermediate friends to get from starters to target
  - Range of 2 to 10
- Points to note:
  - Very scalable: 5 "hops" in a country of 287 million people
  - Based only on local knowledge, no centralized organization required
  - Robust: even if someone deliberately misrouted the letter, it would, at worst, restart the search

Politecnico di Milano

Freenet: Overview

The Freenet Project

- Routed Queries:
  - Join: On startup, client contacts a few other nodes it knows about; gets 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

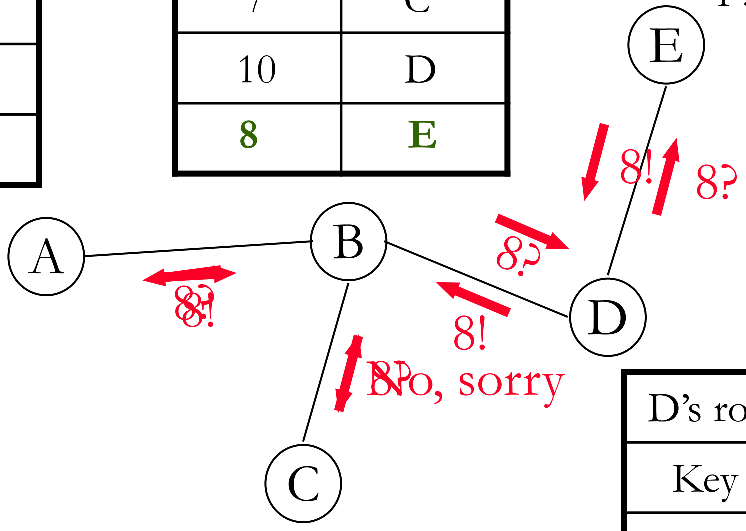
- Insertion of new data can be handled similarly
  - Inserted data is routed in the same way as a request would
    - Search for the id of the data to insert
    - If the id is found the data is not reinserted (it was already present)
    - Otherwise the data is sent along the same path as the request. 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 add entry to routing table associating the key and the data source (can be random decided)

Freenet: Example

A's routing table	
Key	Pointer
8	**E**

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

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"
  - The majority of nodes has relatively few local connections but a significant small number of nodes have large number of neighbors
  - Well known nodes tend to see more requests and become even better connected ("the rich get richer")
- Consequence:
  - Most queries only traverse a small number of hops to find the file

Freenet: Caching properties

The Freenet Project

- 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

The Freenet Project

- Anonymity
  - Messages are forwarded back and forth; nodes can't tell where a message originated
  - Randomly modify source of packet as it traverses the network
- Security & Censorship resistance
  - No constraints on how to choose *ids* for files => easy to have two files collide, creating "denial of service" (censorship)
  - Solution 1: Use SHA-1 so that the id of a file is directly related to its content. It may be easily checked at each hop. Hard to change the file content to tamper with it
  - Solution 2: Have a *id* type that requires a private key signature that is verified when updating the file
  - Cache file on the reverse path of queries/publications => Attempt to "replace" file with bogus data will just cause the file to be replicated more!

Freenet: Cryptography

- 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

Network evolution

- Adding nodes:
  - Announce public key and physical address (e.g. IP) to an existing node (found by out-of-band means).
  - Announcement is recursively forwarded to random nodes
  - Nodes in the chain then collectively assign the new node a random GUID
- Route training:
  - Nodes on the way back may cache copies
  - As more requests are processed, nodes should specialize in handling a few parts of the key space

Freenet: Pros and cons

The Freenet Project

- ☑ 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

Skype

- First p2p IP telephony company
- Decentralized user directory
- Node hierarchy (notice **how much** this concept has been used so far)
- Proprietary protocol and applications
- Channel encryption

Skype: history

- Founded in 2003 in Sweden
  - By, among others, the author of Kazaa
- Acquired in October 2005 by eBay
  - In the same year, videotelephony has been introduced
- 100 million users in April 2006
- … many related services released along the way…
- By the end of 2009, eBay sells the majority of the company to a consortium…
- … and in May 2011 Microsoft acquires the whole thing (new tech bubble?)

Skype: hierarchy

- Three types of nodes
  - Supernode: any client can become one of these, if resources are powerful enough (e.g. no NAT)
    - NAT traversal («hole punching»)
    - Relay communications transparently
  - Node
  - Login server: maintained by Skype owners

Skype: protocol

- SC generates 192bit session key, encrypts with LS 1536bit RSA key
- SC generates 1024bit RSA key pair (login info hashed with MD5 is the shared secret)
- SC hashes the session key into a 256bit AES key, which encrypts the public RSA key and the shared secret
- Session key and RSA pair (encrypted) are sent to the login server

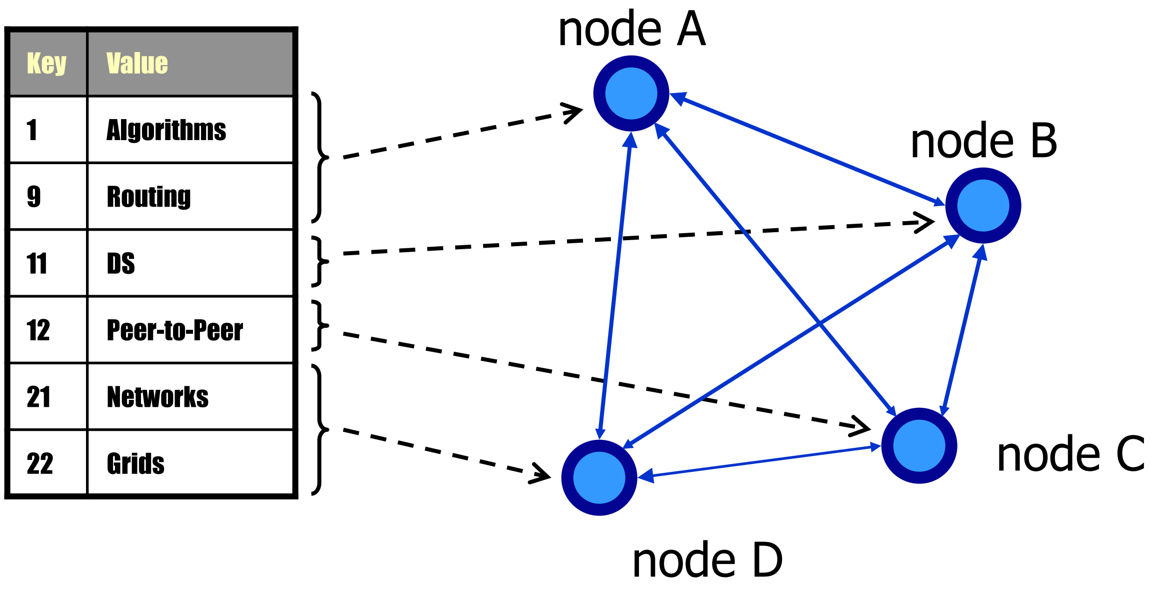
Skype: protocol

- SS opens the session key
- SS opens then the RSA public key and the shared secret
- SS checks the identity and then signs RSA public key with its private key
- SS sends the signed data to the supernodes
- When SC will search for a person to contact, RSA public key will be sent to him to establish a connection

DHT: History

- In 2000-2001, academic researchers said "we want to play too!"
- Motivation:
  - Frustrated by popularity of all these "half-baked" P2P apps :)
  - Guaranteed lookup success for files in system
  - Provable bounds on search time
  - Provable scalability to millions of node
- Hot Topic in networking ever since

DHT: Overview



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

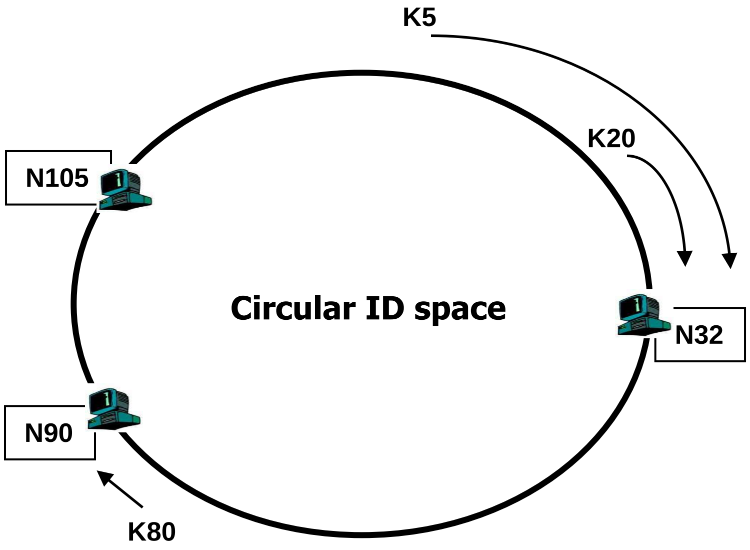
- **Abstraction**: A distributed "hash-table" (DHT) data structure:
  - put(id, item);
    - Actually, item represents only a reference to the real file
  - item = get(id);
- **Implementation**: Nodes in the system form a distributed data structure
  - Can be Ring, Tree, Hypercube, Skip List, Butterfly Network, ...

DHT: Overview

- Structured Overlay Routing:
  - **Join**: On startup, contact a “bootstrap” node and integrate yourself 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. Data structure guarantees that query will meet the publication
  - **Fetch**: Two options:
    - Publication contains actual file => fetch from where query stops
    - Publication says “I have file X” => query tells you 128.2.1.3 has X, use IP routing to get X from 128.2.1.3

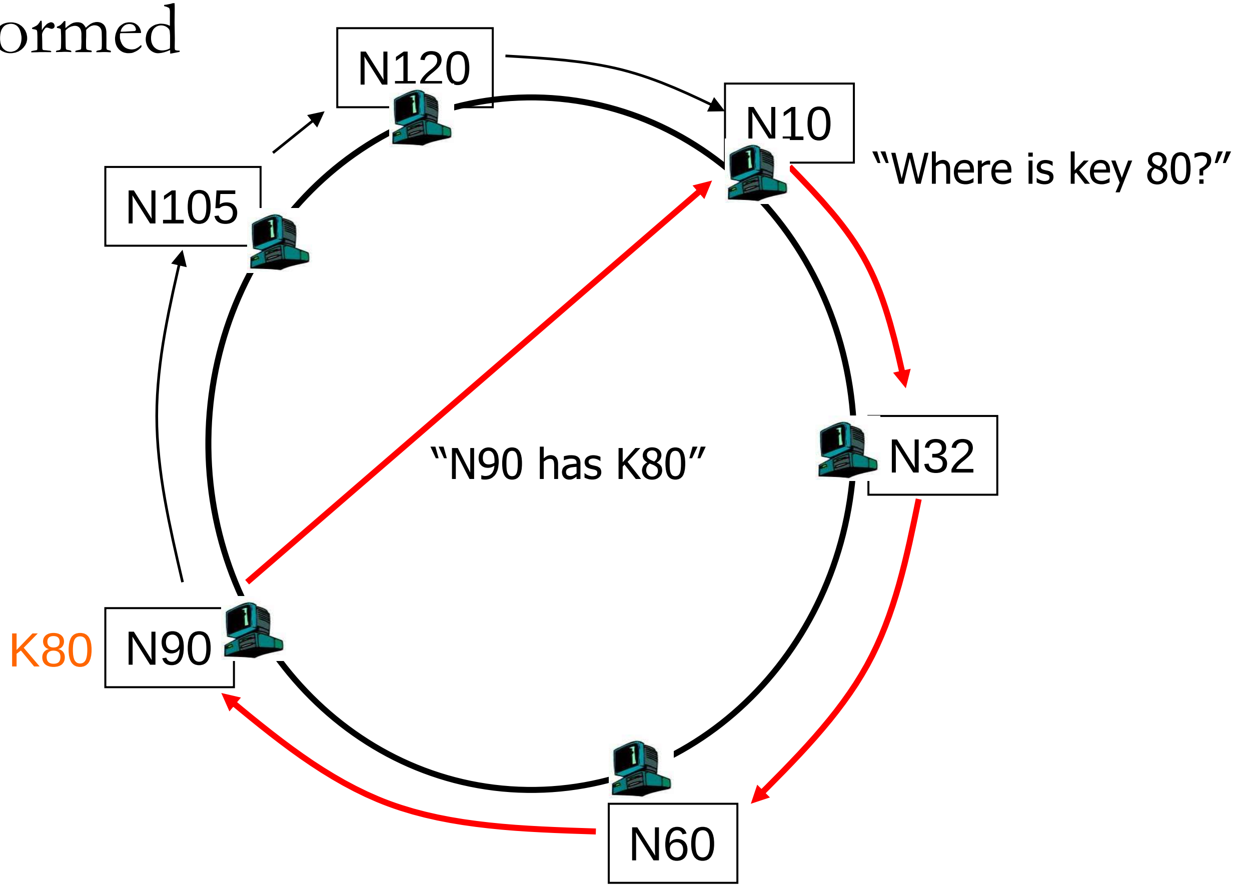
DHT example: Chord

- 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* ≥ *k* (the *successor*)



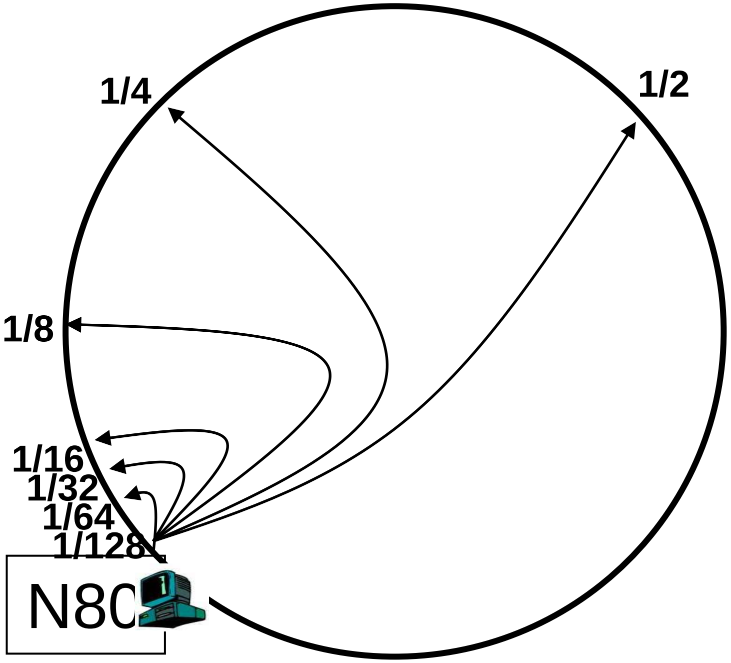
Chord: Basic lookup

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



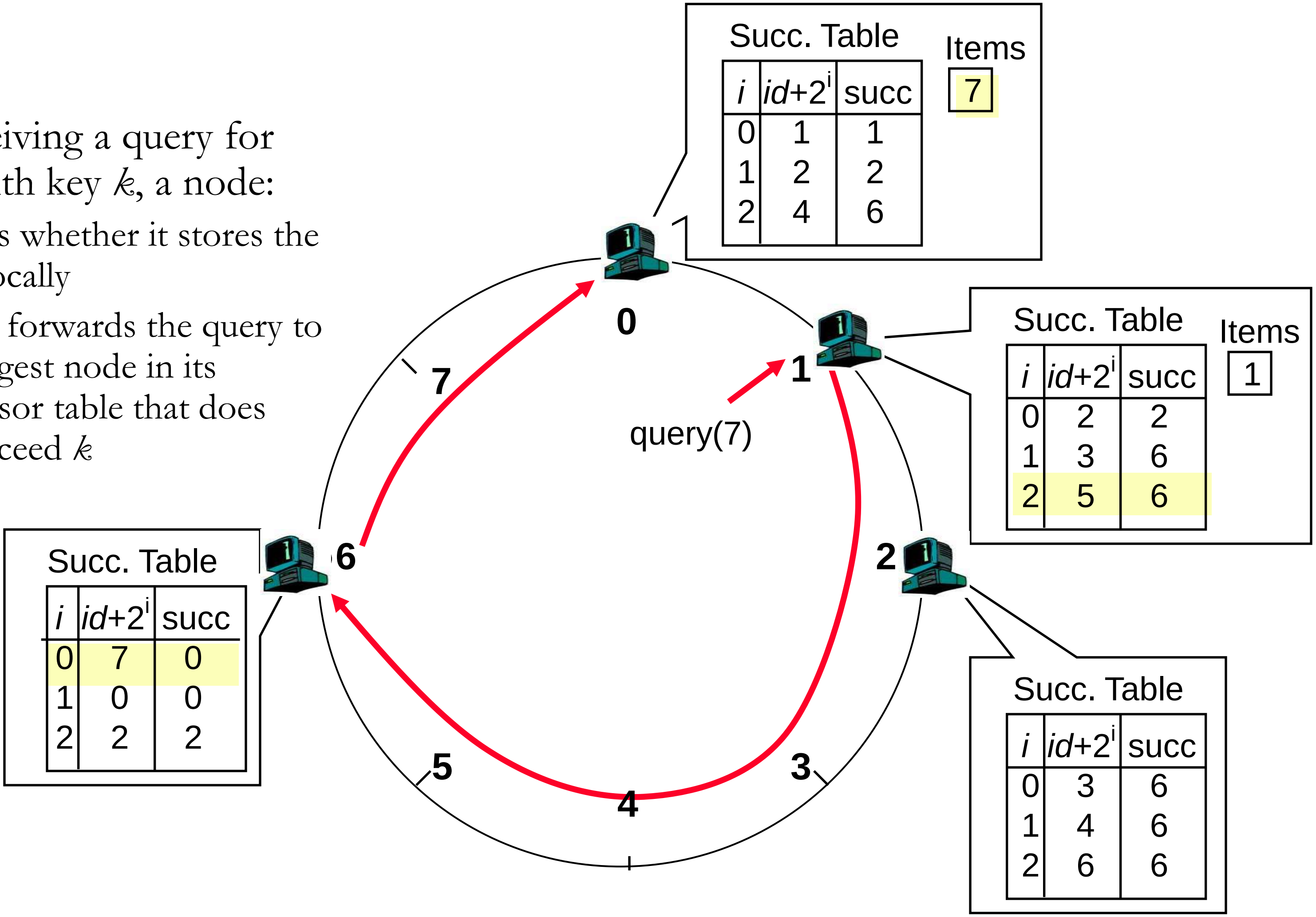
Chord "finger table"

- 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 $n + 2^i$ $(i = 0...m\text{-}1)$
  - In other words, the ith finger points $1/2^{m\text{-}i}$ 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 largest node in its successor table that does not exceed *k*



Succ. Table

i	$id+2^i$	succ
0	1	1
1	2	2
2	4	6

Items

7

Succ. Table

i	$id+2^i$	succ
0	2	2
1	3	6
2	5	6

Items

1

Succ. Table

i	$id+2^i$	succ
0	7	0
1	0	0
2	2	2

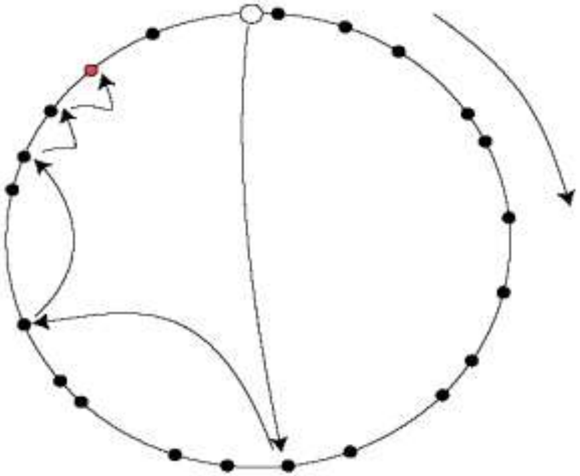
Succ. Table

i	$id+2^i$	succ
0	3	6
1	4	6
2	6	6

Chord: Joining

- Each node keeps also 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:
  - Initialize the predecessor and fingers of node *n*
  - Update the fingers and predecessors of existing nodes to reflect the addition of *n*
- We assume *n* knows another node *n'* already into the system
  - It uses *n'* to initialize its predecessor and fingers
- To update fingers we may observe that:
  - Node *n* will become the i^th^ finger of node *p* iff
    - *p* precedes *n* by at least 2^i
    - the i^th^ finger of *p* succeeds *n*
  - The first node *p* that can meet these two conditions is the immediate predecessor of node $n-2^i$

Chord summary

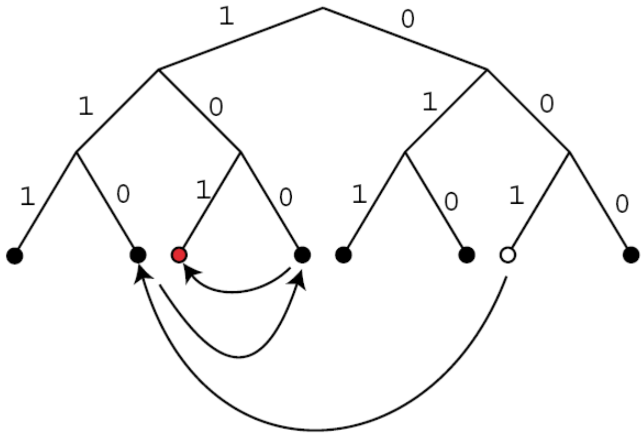


- Routing table size (with $N=2^m$ nodes)?
  - *Log N* fingers
- Routing time?
  - Each hop expects to 1/2 the distance to the desired id => *expect O(log N) hops*
- Joining time?
  - *With high probability expect O(log^2 N) hops*

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 take into account physical topology

Other DHTs: Kademlia

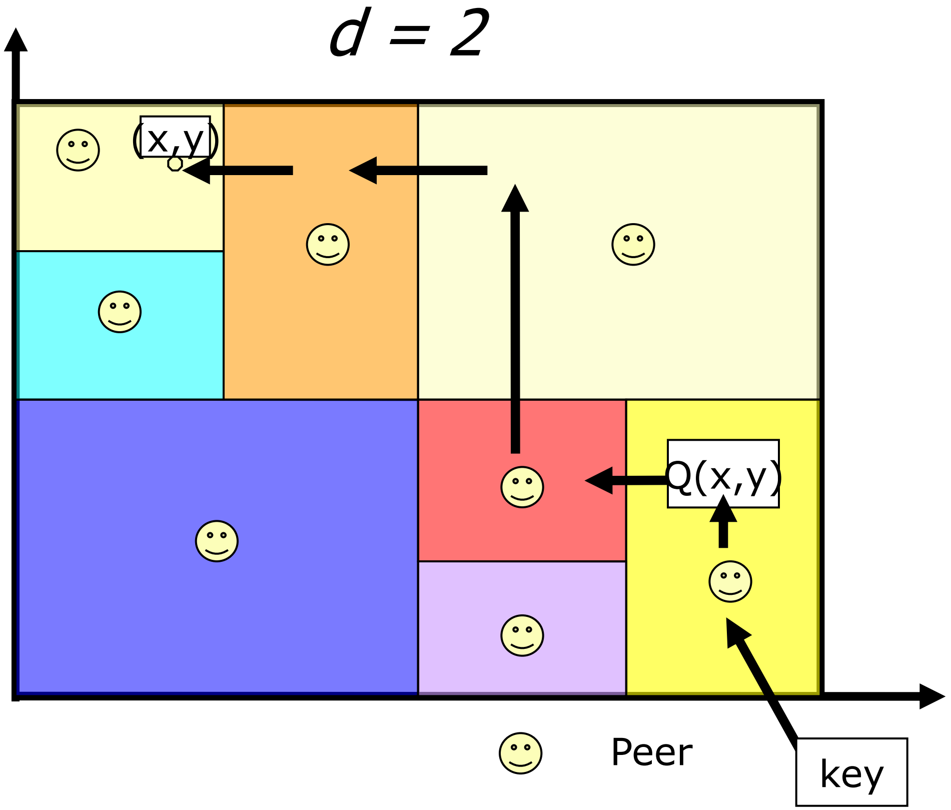


- The Kademlia algorithm is based on the calculation of the "distance" between two nodes
- When searching for some key, the algorithm explores the network in several steps, each step approaching closer to the searched-for key, until the contacted node returns the value, or no more closer nodes are found
- Currently, it is implemented in the new version of EDonkey (a.k.a Overnet) and a Kad-network is available in EMule as well

Politecnico di Milano

Other DHTs: CAN

- Internet-scale distributed hash table
- Uses a d-dimensional coordinate space to store (key, value) pairs to nodes
- Each CAN node holds an IP address and virtual coordinate zone of each of its neighbors in the space



Routing state O(d), Hops to find data o($dN^{1/d}$)

(however if d=(log N) / 2, we get Chord's performance)

Other DHTs: Pastry

- Each node assigned a 128-bit node identifier (nodeID)
- FileID computed as an hash of file's name + owner's public key + randomly chosen salt
- Message routed towards node whose nodeID is numerically closest to fileID
- Physical topology is taken into account

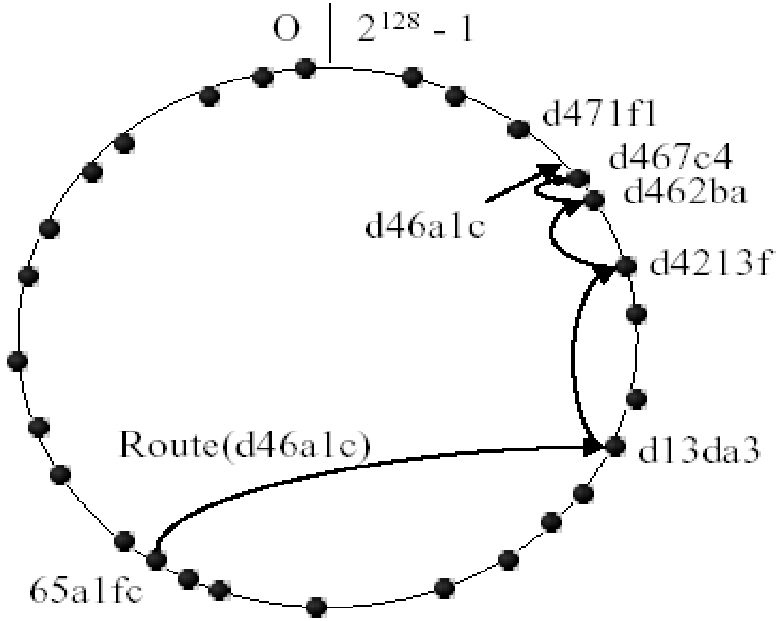


Figure 2: Routing a message from node *65a1fc* with key *d46a1c*. The dots depict live nodes in Pastry's circular namespace.

*Routing state O(*log *n)*
*Hops to find data O(*log *n)*

Politecnico di Milano

DHT - References

- Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan, *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*, ACM SIGCOMM 2001, San Diego, CA, August 2001, pp. 149-160
- P. Maymounkov and D. Mazieres. *Kademlia: A peer-to-peer information system based on the xor metric*. In Proceedings of the 1st International Workshop on Peer-to-Peer Systems, Mar. 2002
- Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. *A Scalable Content-Addressable Network*. In Proceedings of the ACM SIGCOMM Symposium on Communication, Architecture, and Protocols, pages 161--172, San Diego, CA, U.S.A., August 2001
- A. Rowstron and P. Druschel.*Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems*. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, pages 329-350, November, 2001

P2P: Summary

- Peer-to-Peer is a paradigm while file-sharing is just one of the possible applications exploiting it
- Many different styles; remember pros and cons of each
  - Centralized, flooding, swarming, unstructured and structured routing
- Lessons learned:
  - Single points of failure are very bad
  - Flooding messages to everyone is bad
  - Underlying network topology is important
  - Not all nodes are equal
  - Need incentives to discourage freeloading
  - Privacy and security are important
  - Structure can provide theoretical bounds and guarantees
- Open Issue:
  - How to implement an efficient search?