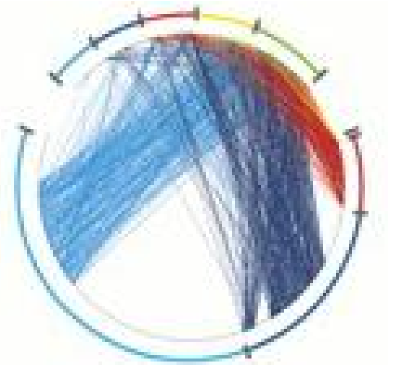
Models and Algorithms for Complex Networks

Searching in P2P Networks



What is P2P?

- “the sharing of computer resources and services by direct exchange of information”

What is P2P?

- “P2P is a class of applications that take advantage of resources – storage, cycles, content, human presence – available at the edges of the Internet. Because accessing these decentralized resources means operating in an environment of unstable and unpredictable IP addresses P2P nodes must operate outside the DNS system and have significant, or total autonomy from central servers”

What is P2P?

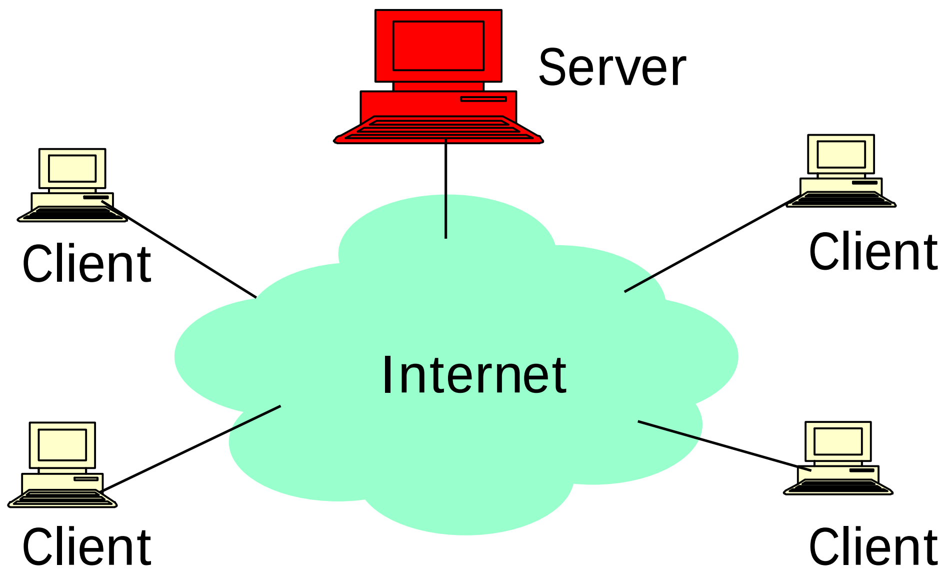
- “A distributed network architecture may be called a P2P network if the participants share a part of their own resources. These shared resources are necessary to provide the service offered by the network. The participants of such a network are both resource providers and resource consumers”

What is P2P?

- Various definitions seem to agree on
  - sharing of resources
  - direct communication between equals (peers)
  - no centralized control

Client/Server Architecture

- Well known, powerful, reliable server is a data source
- Clients request data from server
- Very successful model
  - WWW (HTTP), FTP, Web services, etc.



* Figure from http://project-iris.net/talks/dht-toronto-03.ppt

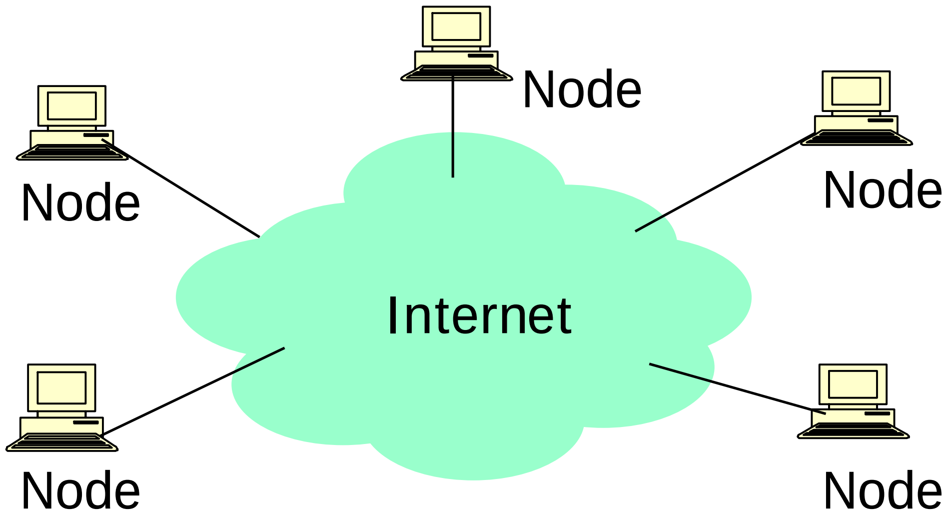
Client/Server Limitations

- Scalability is hard to achieve
- Presents a single point of failure
- Requires administration
- Unused resources at the network edge

- P2P systems try to address these limitations

P2P Architecture

- Clients are also servers and routers
  - Nodes contribute content, storage, memory, CPU
- Nodes are autonomous (no administrative authority)
- Network is dynamic: nodes enter and leave the network “frequently”
- Nodes collaborate directly with each other (not through well-known servers)
- Nodes have widely varying capabilities



* Content from http://project-iris.net/talks/dht-toronto-03.ppt

P2P Goals and Benefits

- Efficient use of resources
  - Unused bandwidth, storage, processing power at the “edge of the network”
- Scalability
  - No central information, communication and computation bottleneck
  - Aggregate resources grow naturally with utilization
- Reliability
  - Replicas
  - Geographic distribution
  - No single point of failure
- Ease of administration
  - Nodes self-organize
  - Built-in fault tolerance, replication, and load balancing
  - Increased autonomy
- Anonymity – Privacy
  - not easy in a centralized system
- Dynamism
  - highly dynamic environment
  - ad-hoc communication and collaboration

P2P Applications

- Are these P2P systems?
  - File sharing (Napster, Gnutella, Kazaa)
  - Multiplayer games (Unreal Tournament, DOOM)
  - Collaborative applications (ICQ, shared whiteboard)
  - Distributed computation (Seti@home)
  - Ad-hoc networks
- We will focus on information sharing P2P systems

Information sharing P2P systems

- The resource to be shared is information (e.g. files)
- The participants create an overlay network over a physical network (e.g. the Internet)
- P2P search problem: locate the requested information in the overlay network efficiently
  - small number of messages and hops
  - low latency
  - load balance
  - easy to update in a highly dynamic setting

Popular file sharing P2P Systems

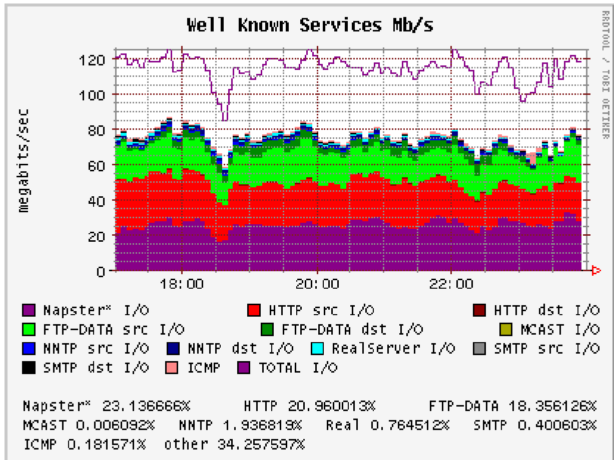
- Napster, Gnutella, Kazaa, Freenet

- Large scale sharing of files.
  - User A makes files (music, video, etc.) on their computer available to others
  - User B connects to the network, searches for files and downloads files *directly* from user A

- Issues of copyright infringement

Napster

- program for sharing files over the Internet
- a “disruptive” application/technology?
- history:
  - 5/99: Shawn Fanning (freshman, Northeasten U.) founds Napster Online music service
  - 12/99: first lawsuit
  - 3/00: 25% UWisc traffic Napster
  - 2000: est. 60M users
  - 2/01: US Circuit Court of Appeals: Napster knew users violating copyright laws
  - 7/01: # simultaneous online users: Napster 160K, Gnutella: 40K, Morpheus: 300K



Napster: how does it work

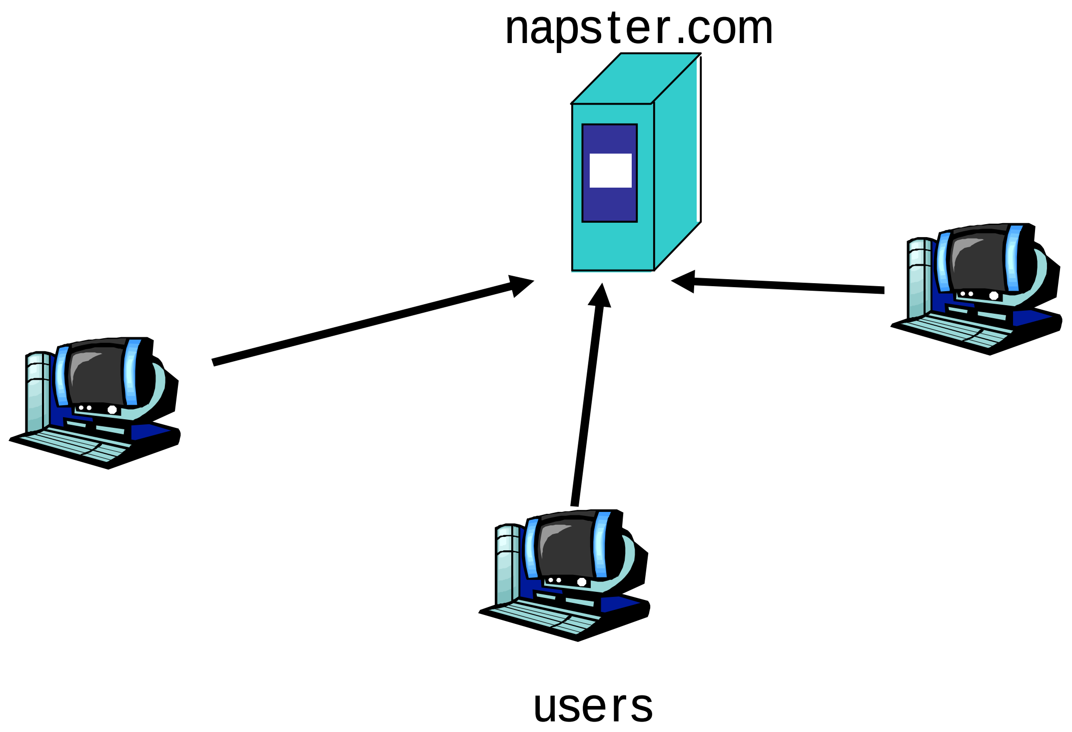
Application-level, client-server protocol over point-to-point TCP

Four steps:

- Connect to Napster server
- Upload your list of files (push) to server.
- Give server keywords to search the full list with.
- Select “best” of correct answers. (pings)

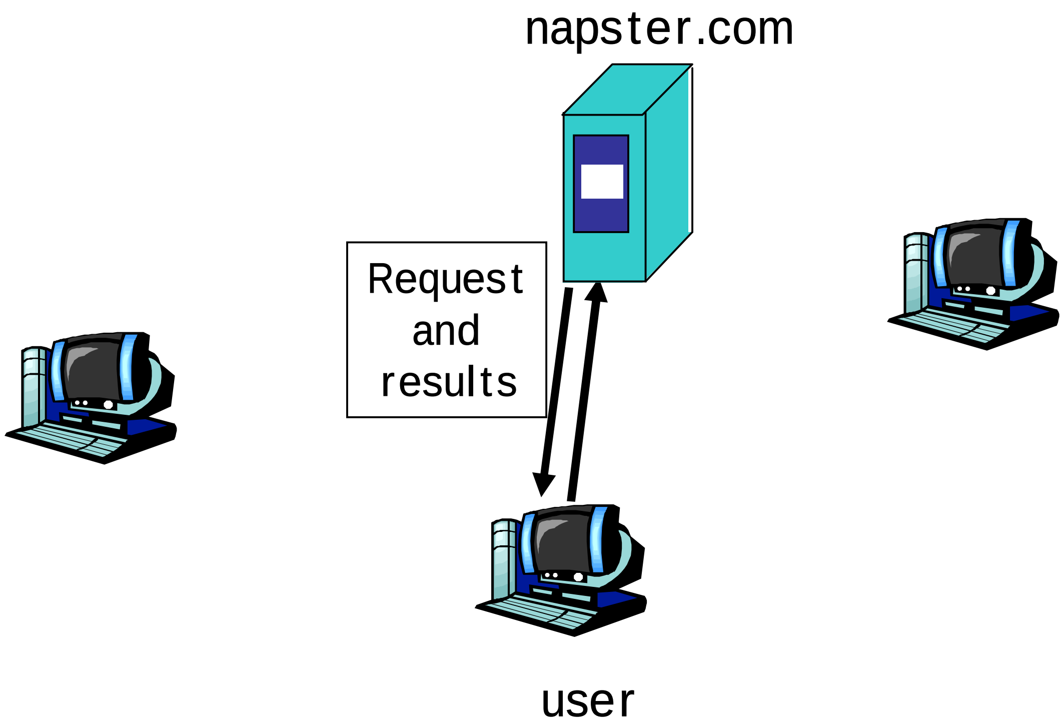
Napster

1. File list is uploaded



Napster

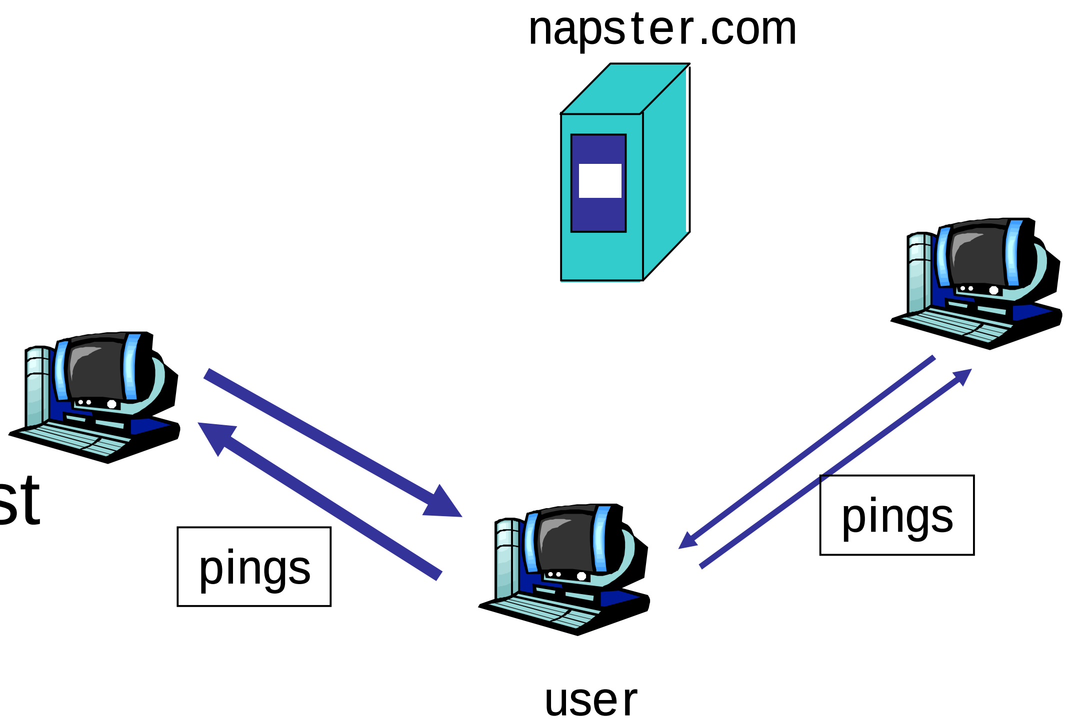
2. User requests search at server.



Napster

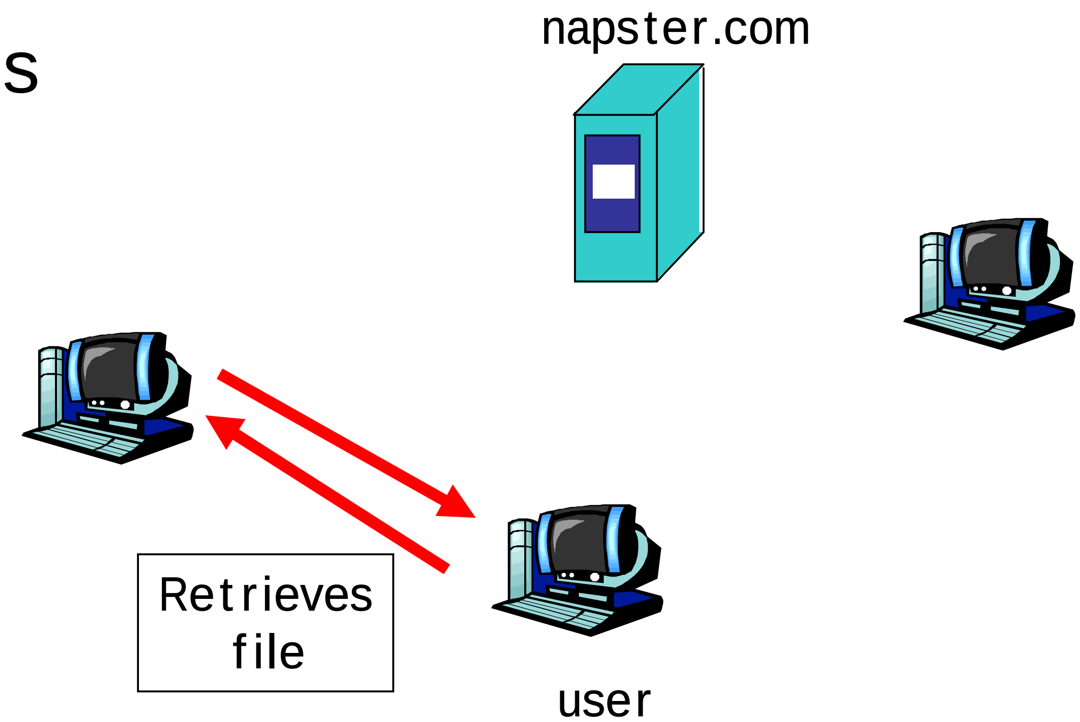
3. User pings hosts that apparently have data.

   Looks for best transfer rate.



Napster

4. User retrieves file



Napster

- § Central Napster server
  - þ Can ensure correct results
  - þ Fast search

  - ý Bottleneck for scalability
  - ý Single point of failure
  - ý Susceptible to denial of service
    - Malicious users
    - Lawsuits, legislation

- § Hybrid P2P system – “all peers are equal but some are more equal than others”
  - § Search is centralized
  - § File transfer is direct (peer-to-peer)

Gnutella

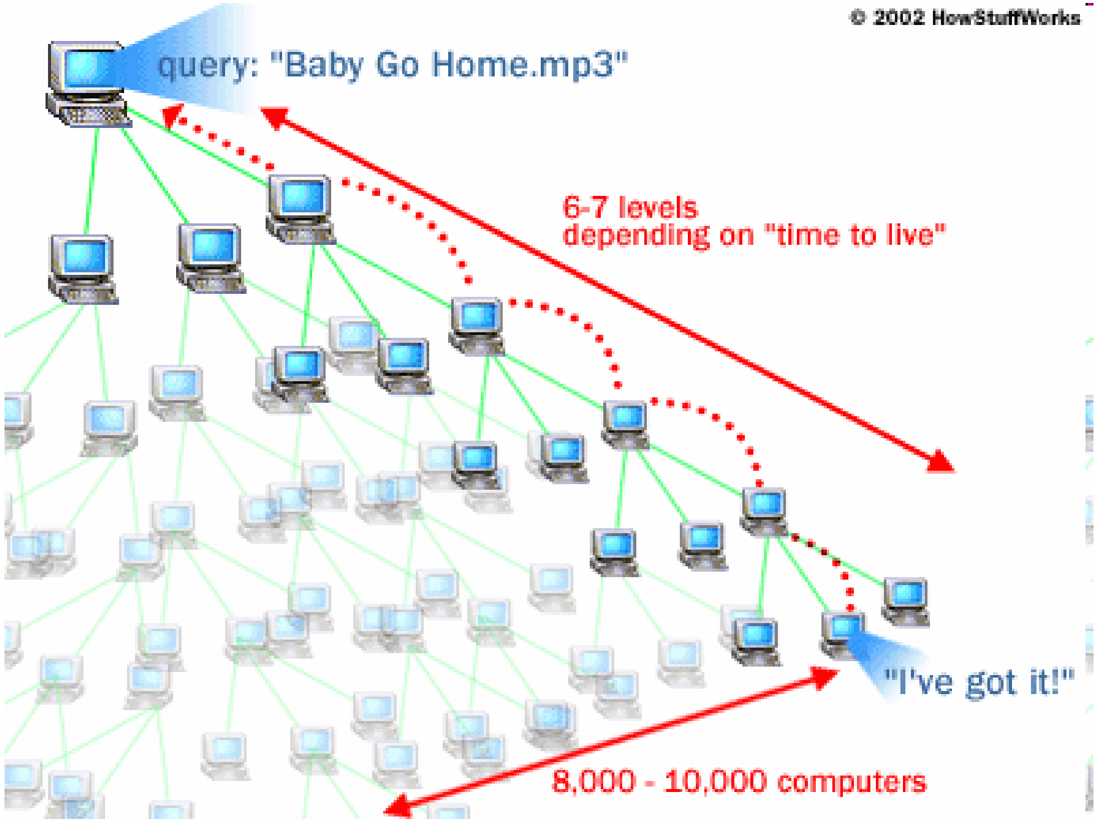
- Share any type of files (not just music)
- Completely decentralized method of searching for files
  - applications connect to peer applications

- each application instance serves to:
  - store selected files
  - route queries (file searches) from and to its neighboring peers
  - respond to queries (serve file) if file stored locally
- Gnutella history:
  - 3/14/00: release by AOL, almost immediately withdrawn
  - too late: 23K users on Gnutella at 8 am this AM
  - reverse engineered. many iterations to fix poor initial design

Gnutella

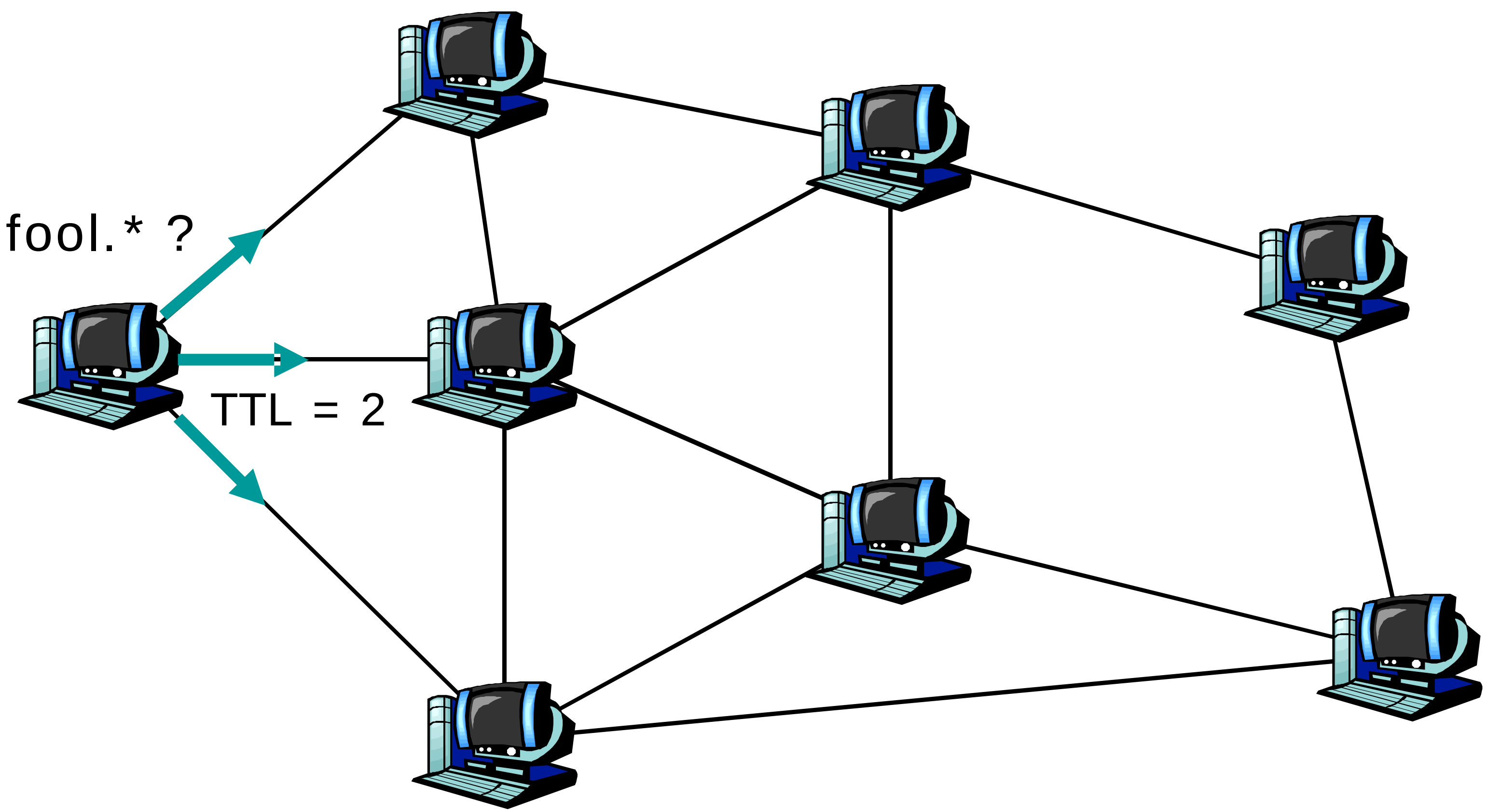
Searching by flooding:

- If you don't have the file you want, query 7 of your neighbors.
- If they don't have it, they contact 7 of their neighbors, for a maximum hop count of 10.
- Requests are flooded, but there is no tree structure.
- No looping but packets may be received twice.
- Reverse path forwarding

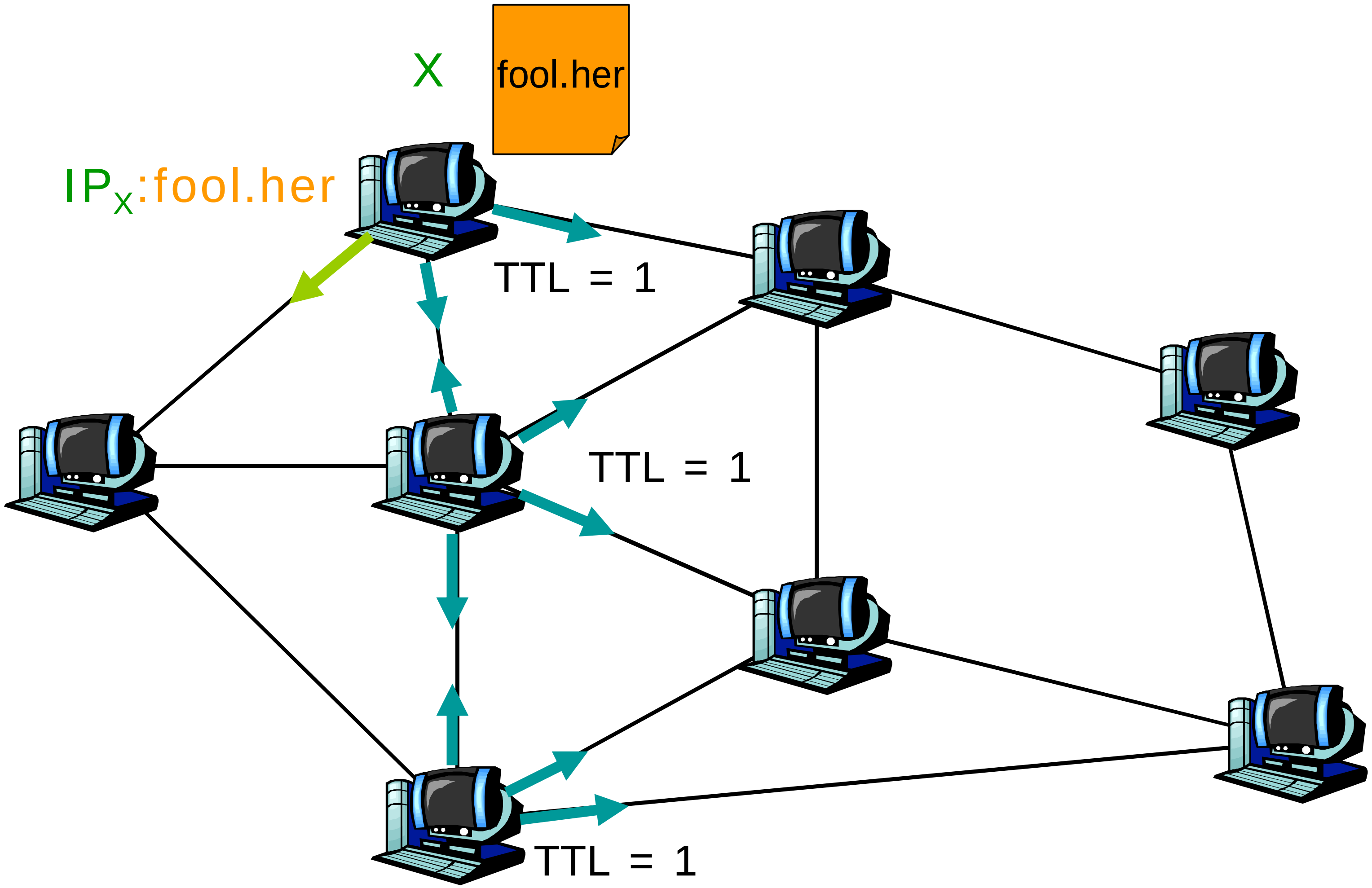


* Figure from http://computer.howstuffworks.com/file-sharing.htm

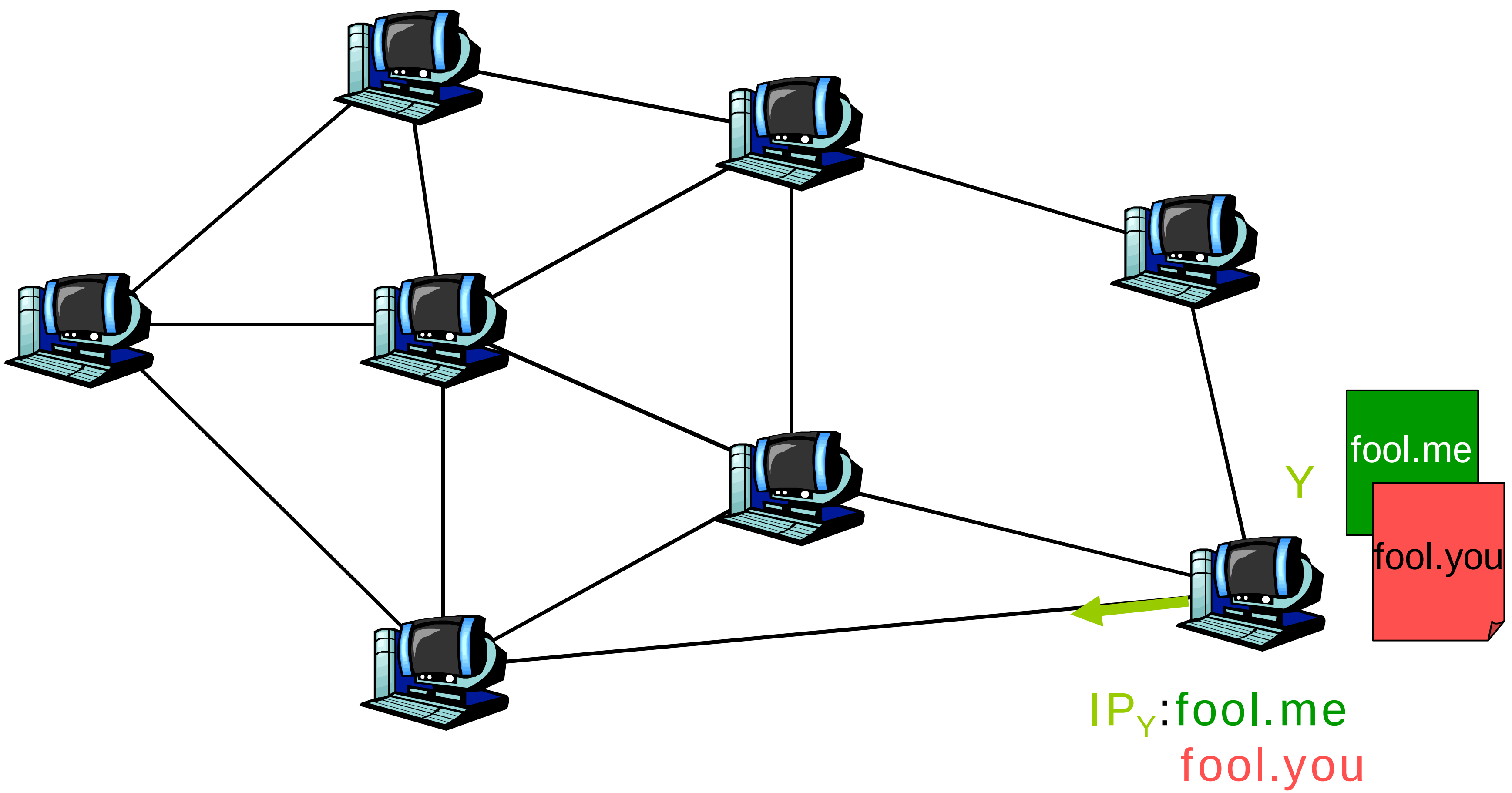
Gnutella



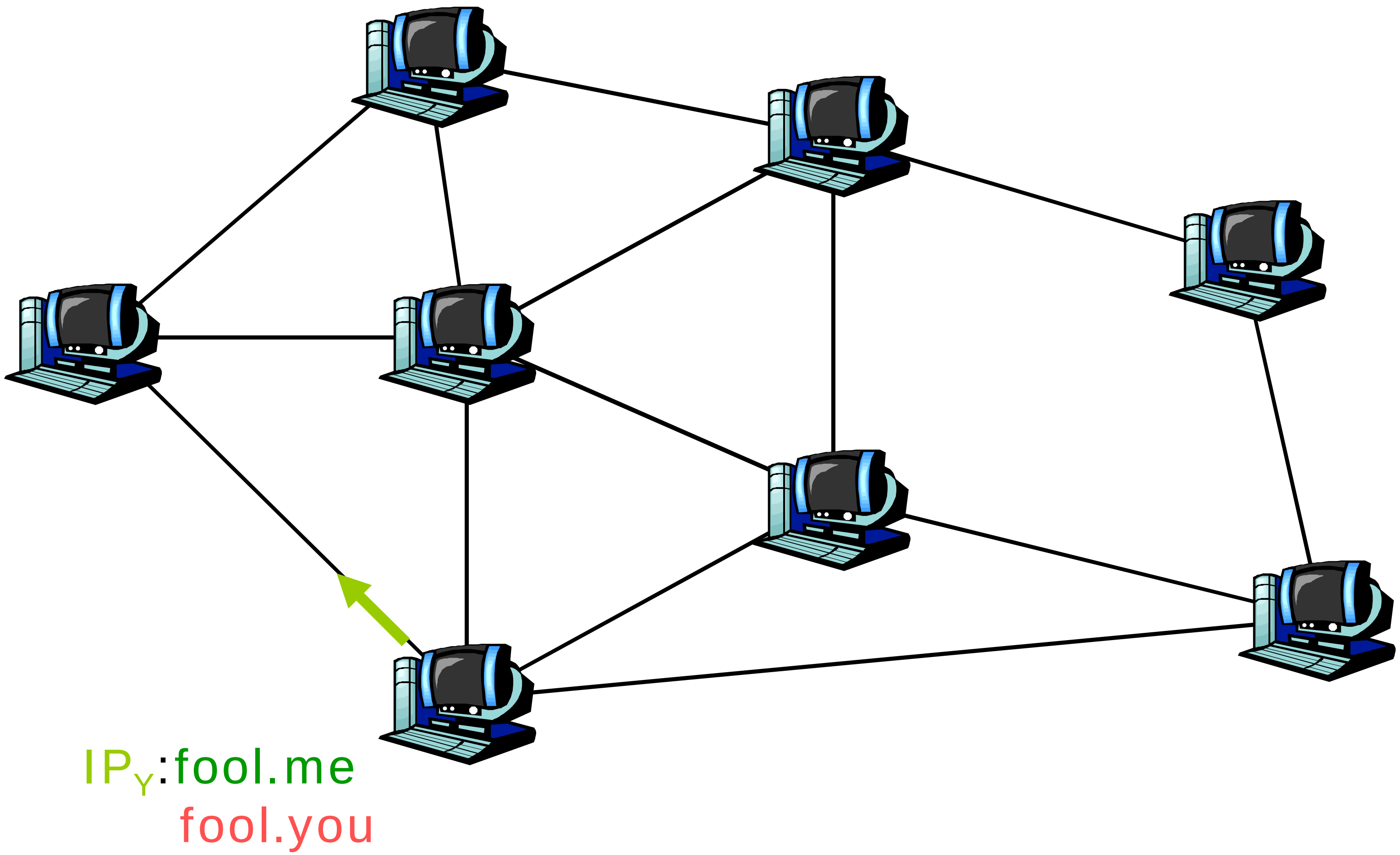
Gnutella



Gnutella



Gnutella



Gnutella: strengths and weaknesses

§ pros:

þ flexibility in query processing

þ complete decentralization

þ simplicity

þ fault tolerance/self-organization

§ cons:

ý severe scalability problems

ý susceptible to attacks

§ Pure P2P system

Gnutella: initial problems and fixes

- 2000: avg size of reachable network only 400-800 hosts. Why so small?
  - modem users: not enough bandwidth to provide search routing capabilities: routing black holes
- Fix: create peer hierarchy based on capabilities
  - previously: all peers identical, most modem black holes
  - preferential connection:
    - favors routing to well-connected peers
    - favors replies to clients that themselves serve large number of files: prevent freeloading

Kazaa (Fasttrack network)

- Hybrid of centralized Napster and decentralized Gnutella
  - hybrid P2P system
- Super-peers act as local search hubs
  - Each super-peer is similar to a Napster server for a small portion of the network
  - Super-peers are automatically chosen by the system based on their capacities (storage, bandwidth, etc.) and availability (connection time)
- Users upload their list of files to a super-peer
- Super-peers periodically exchange file lists
- You send queries to a super-peer for files of interest

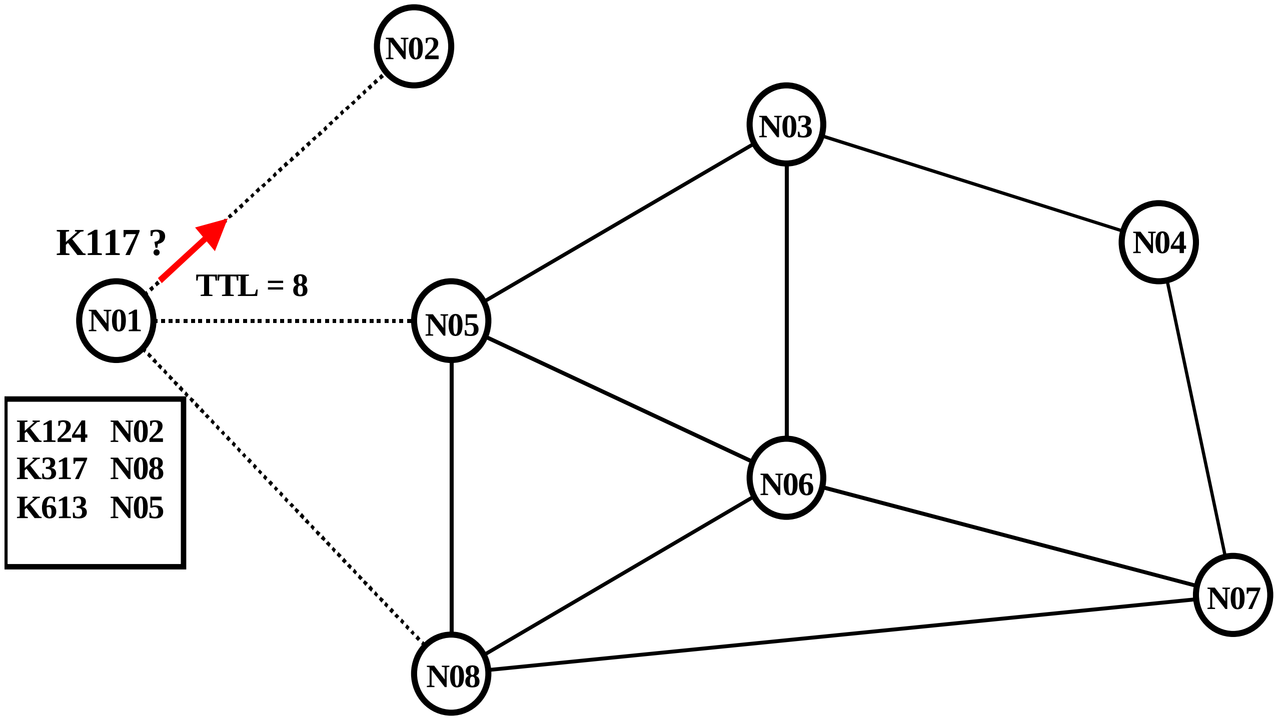
Anonymity

- Napster, Gnutella, Kazaa don’t provide anonymity
  - Users know who they are downloading from
  - Others know who sent a query
- Freenet
  - Designed to provide anonymity among other features

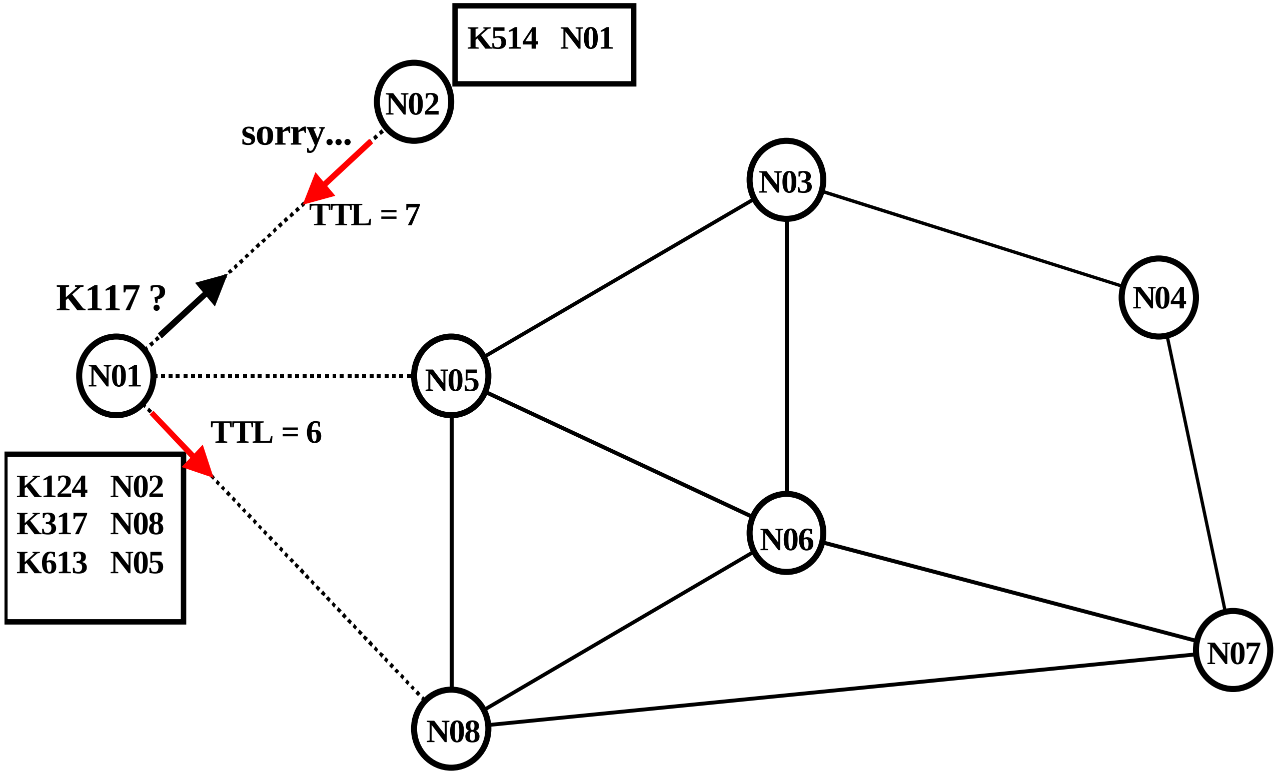
Freenet

- Keys are mapped to IDs
- Each node stores a cache of keys, and a routing table for some keys
  - routing table defines the overlay network
- Queries are routed to the node with the most similar key
- Data flows in reverse path of query
  - Impossible to know if a user is initiating or forwarding a query
  - Impossible to know if a user is consuming or forwarding data
  - Keys replicated in (some) of the nodes along the path

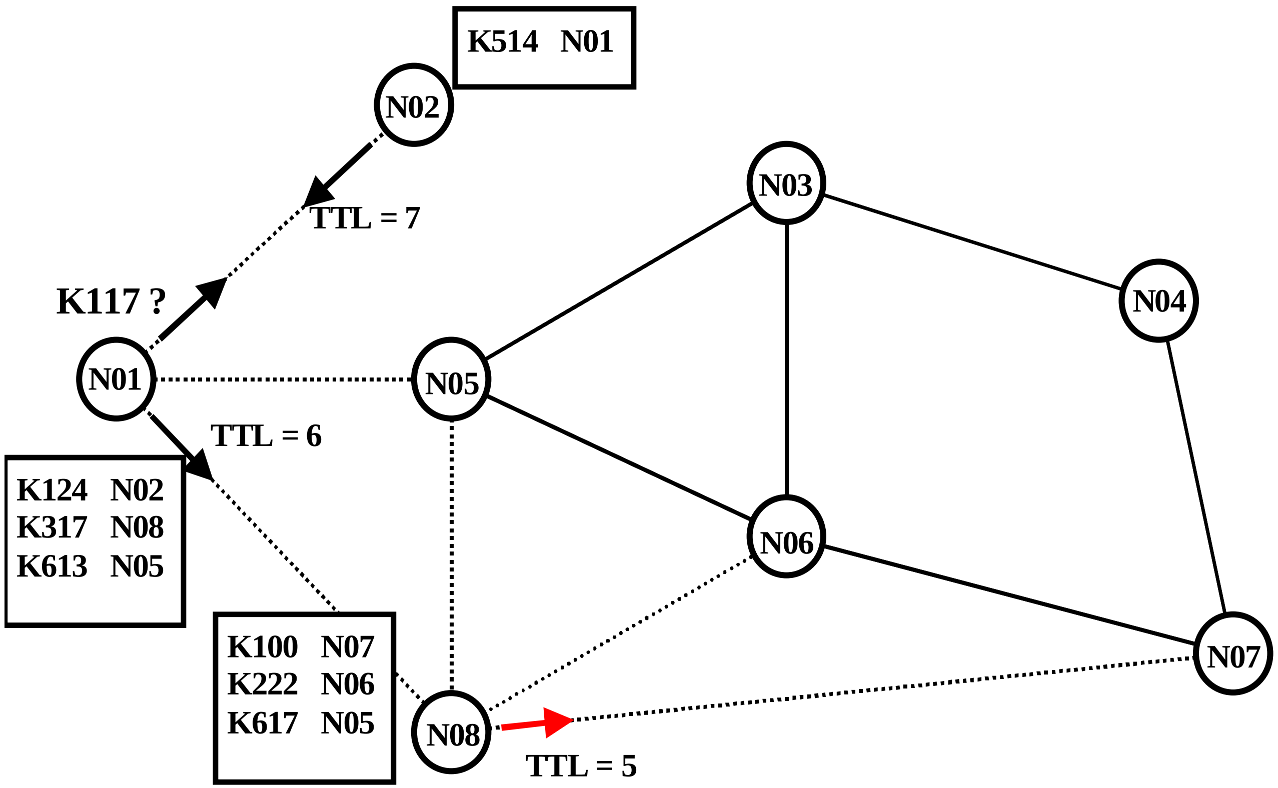
Freenet routing



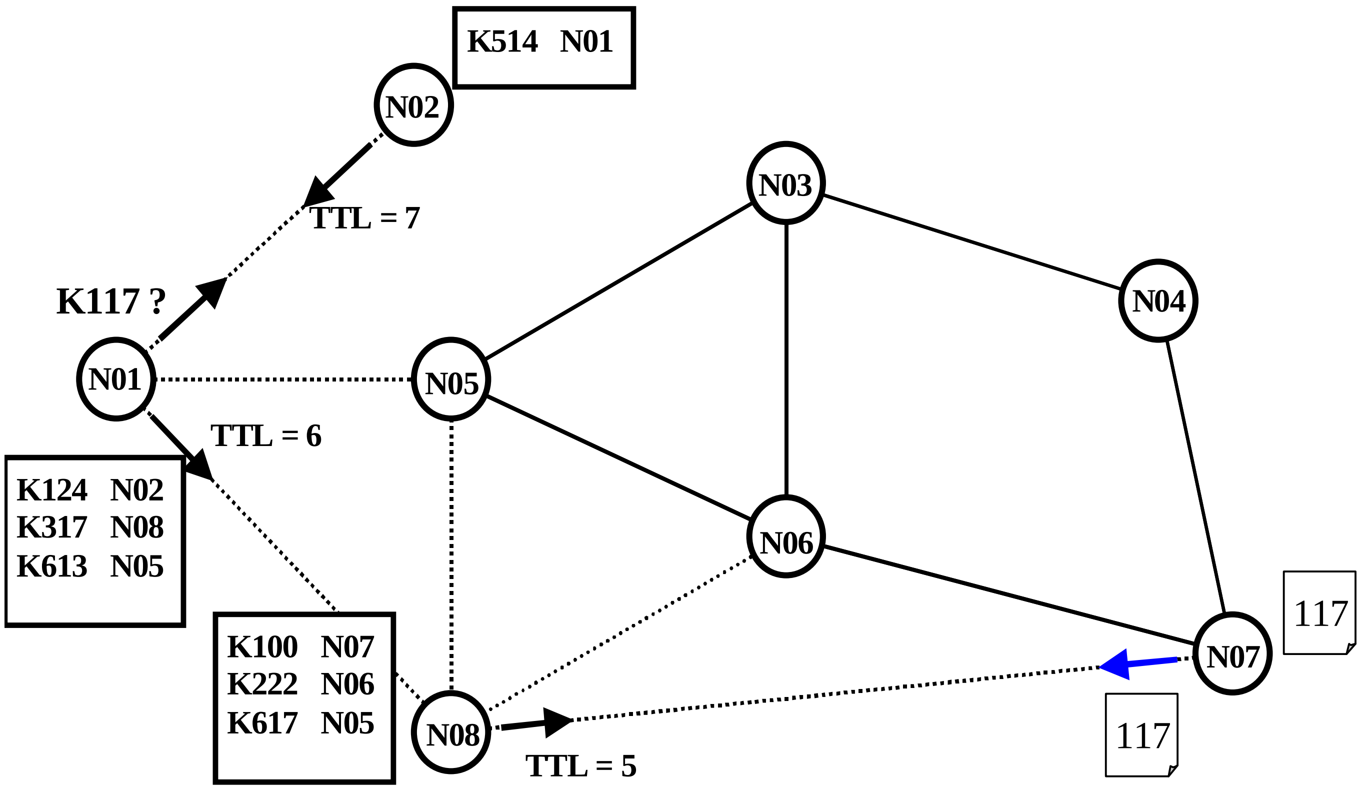
Freenet routing



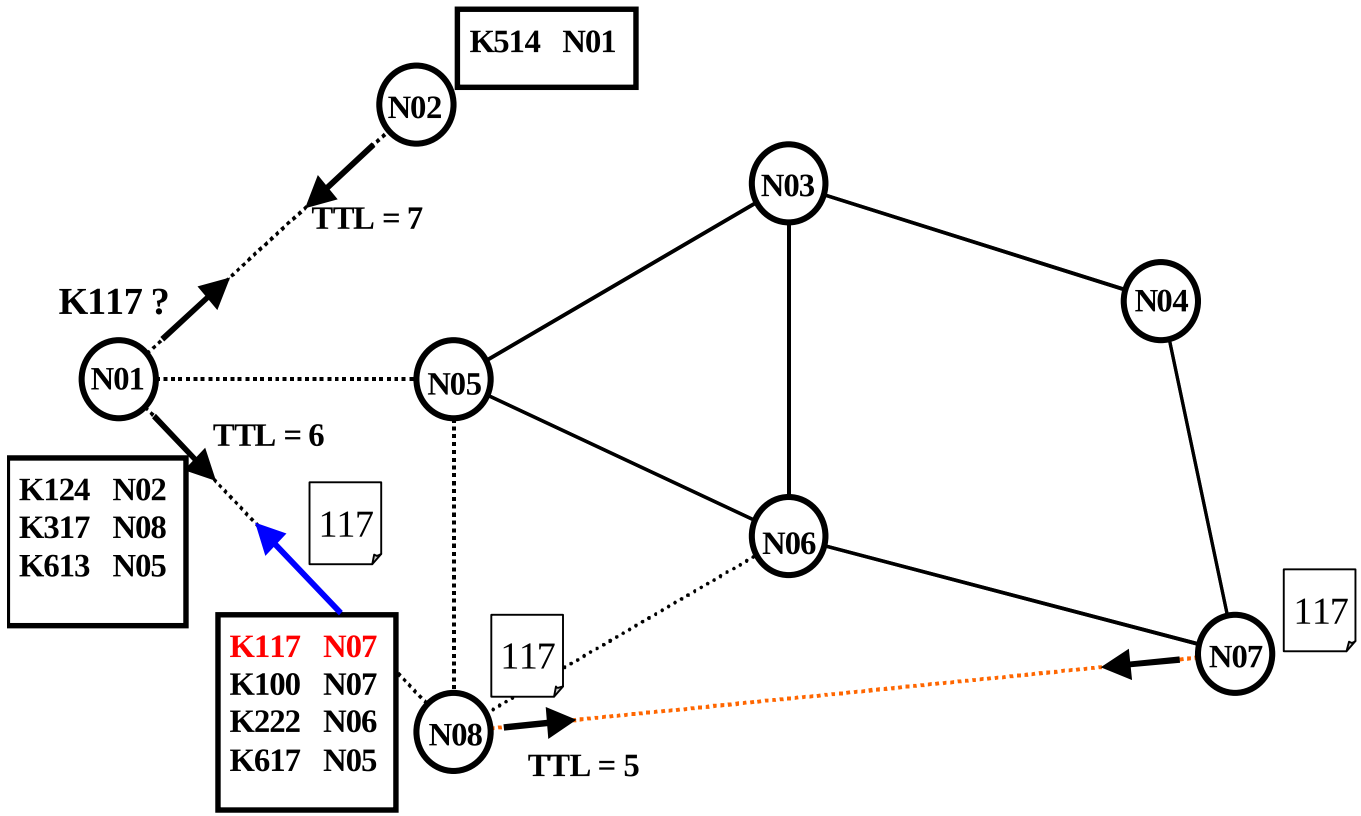
Freenet routing



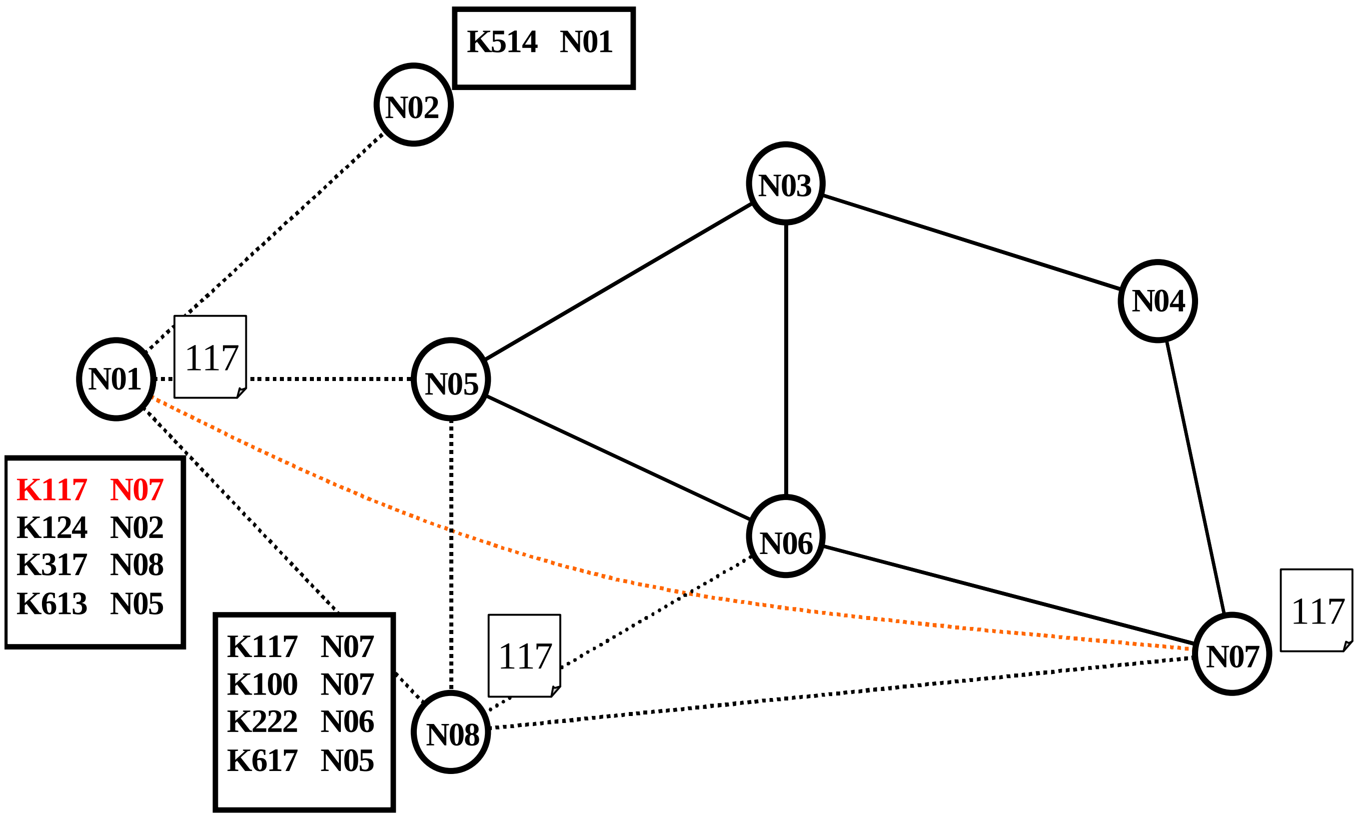
Freenet routing



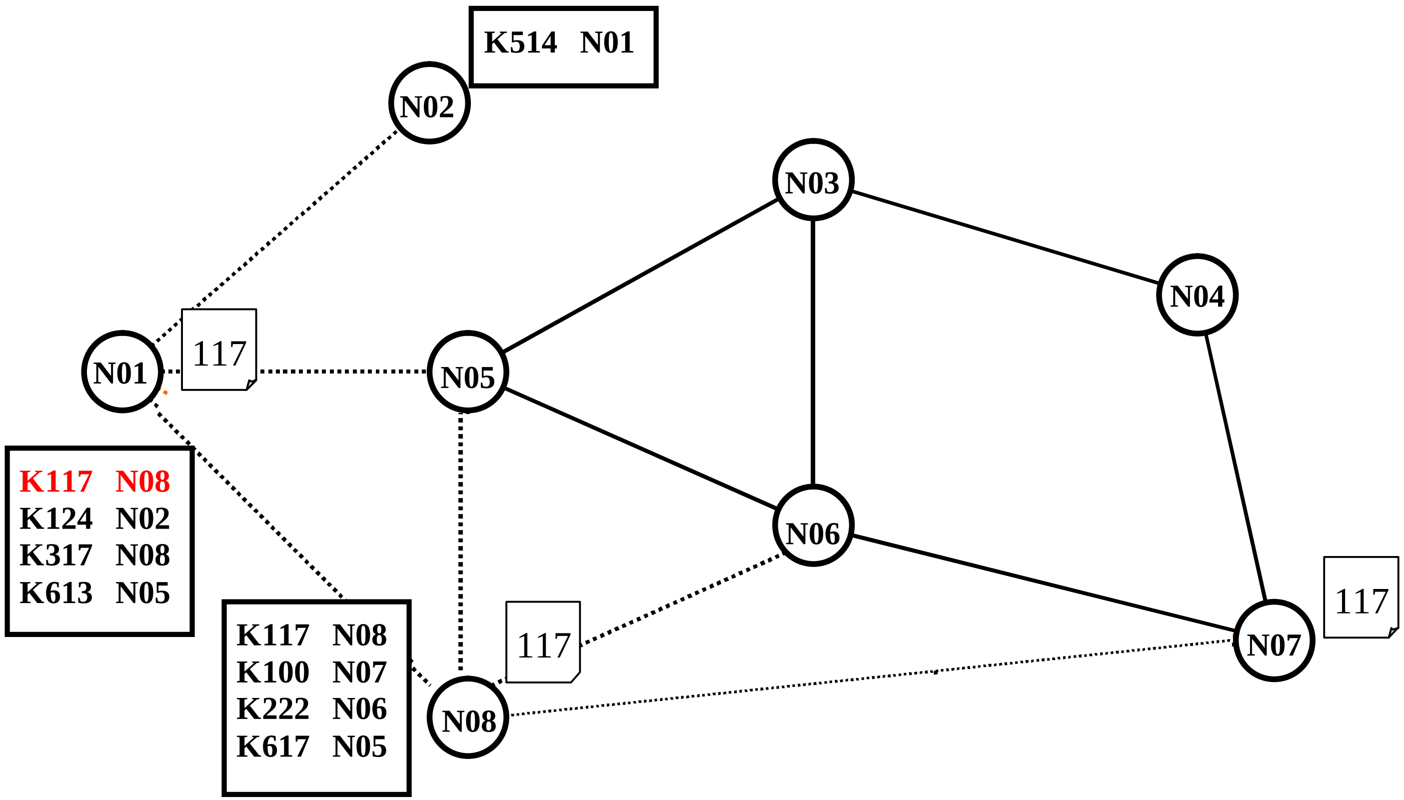
Freenet routing



Freenet routing



Freenet routing



Inserts are performed similarly – they are unsuccessful queries

Freenet strengths and weaknesses

§ pros:

- þ complete decentralization
- þ fault tolerance/self-organization
- þ anonymity
- þ scalability (to some degree)

§ cons:

- ý questionable efficiency & performance
- ý rare keys disappear from the system
- ý improvement of performance requires high overhead (maintenance of additional information for routing)

Unstructured vs Structured P2P

- The systems we described do not offer any guarantees about their performance (or even correctness)
- Structured P2P
  - Scalable guarantees on numbers of hops to answer a query
  - Maintain all other P2P properties (load balance, self-organization, dynamic nature)

- Approach: Distributed Hash Tables (DHT)

Distributed Hash Tables (DHT)

- Distributed version of a hash table data structure
- Stores (key, value) pairs
  - The key is like a filename
  - The value can be file contents, or pointer to location
- Goal: Efficiently insert/lookup/delete (key, value) pairs

- Each peer stores a subset of (key, value) pairs in the system
- Core operation: Find node responsible for a key
  - Map key to node
  - Efficiently route insert/lookup/delete request to this node
- Allow for frequent node arrivals/departures

DHT Desirable Properties

- Keys should mapped evenly to all nodes in the network (load balance)
- Each node should maintain information about only a few other nodes (scalability, low update cost)
- Messages should be routed to a node efficiently (small number of hops)
- Node arrival/departures should only affect a few nodes

DHT Routing Protocols

- DHT is a generic interface

- There are several implementations of this interface
  - Chord [MIT]
  - Pastry [Microsoft Research UK, Rice University]
  - Tapestry [UC Berkeley]
  - Content Addressable Network (CAN) [UC Berkeley]
  - Viceroy [Israel, UC Berkeley]

  - SkipNet [Microsoft Research US, Univ. of Washington]
  - Kademlia [New York University]
  - P-Grid [EPFL Switzerland]
  - Freenet [Ian Clarke]

Basic Approach

In all approaches:

- keys are associated with globally unique IDs
  - integers of size m (for large m)

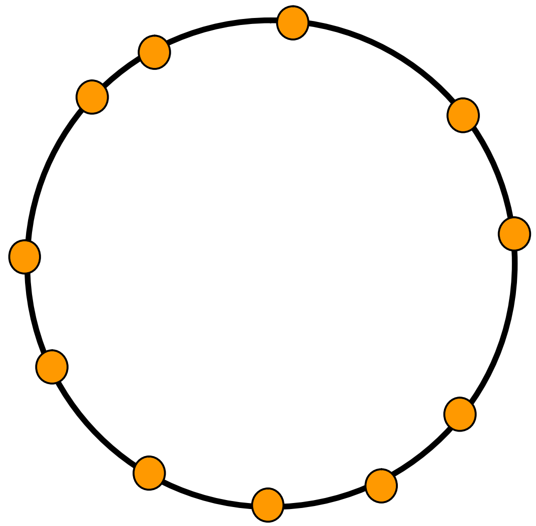
- key ID space (search space) is uniformly populated - mapping of keys to IDs using (consistent) hashing

- a node is responsible for indexing all the keys in a certain subspace (zone) of the ID space

- nodes have only partial knowledge of other node's responsibilities

Consistent Hashing

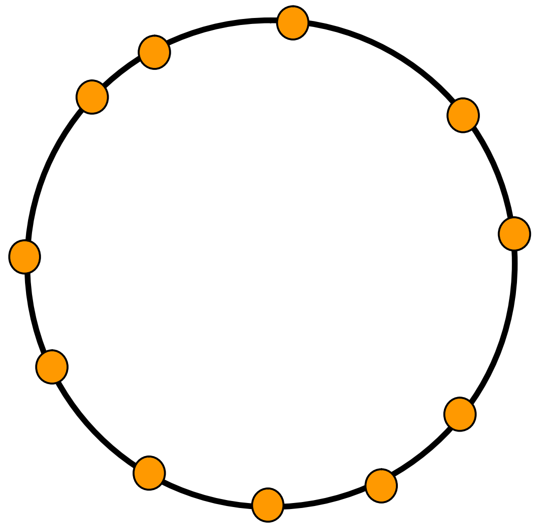
§ Example: map the keys to the ring



The ring is just a possibility.
Any metric space will do

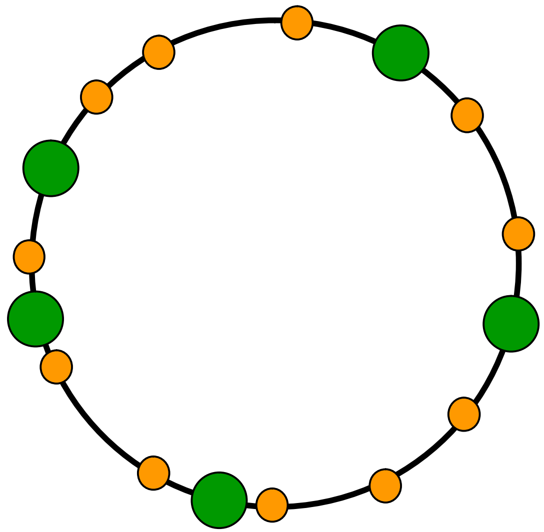
Consistent Hashing

- How do we distribute the keys uniformly to the nodes?



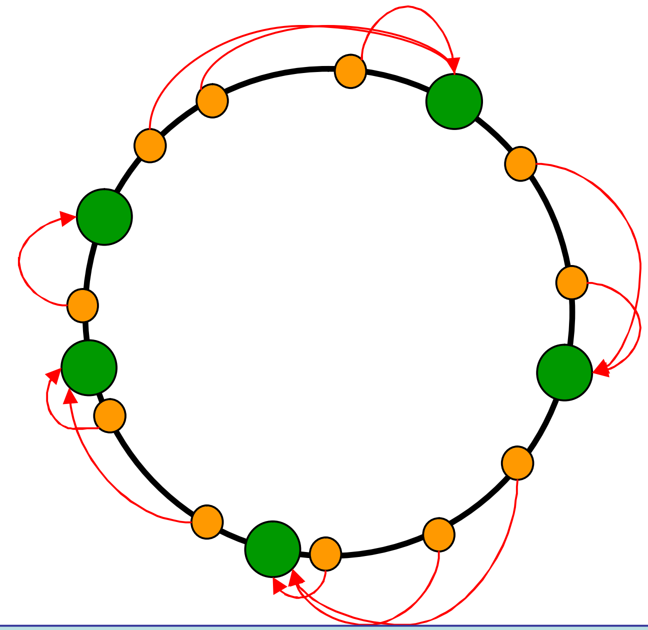
Consistent Hashing

§ The main idea: map both keys and nodes (node IPs) to the same (metric) ID space



Consistent Hashing

- The main idea: map both keys and nodes (node IPs) to the same (metric) ID space



- Each key is assigned to the node with ID clockwise closest to the key ID
    - uniformly distributed
    - at most logarithmic number of keys assigned to each node

Problem: Starting from a node, how do we locate the node responsible for a key, while maintaining as little information about other nodes as possible

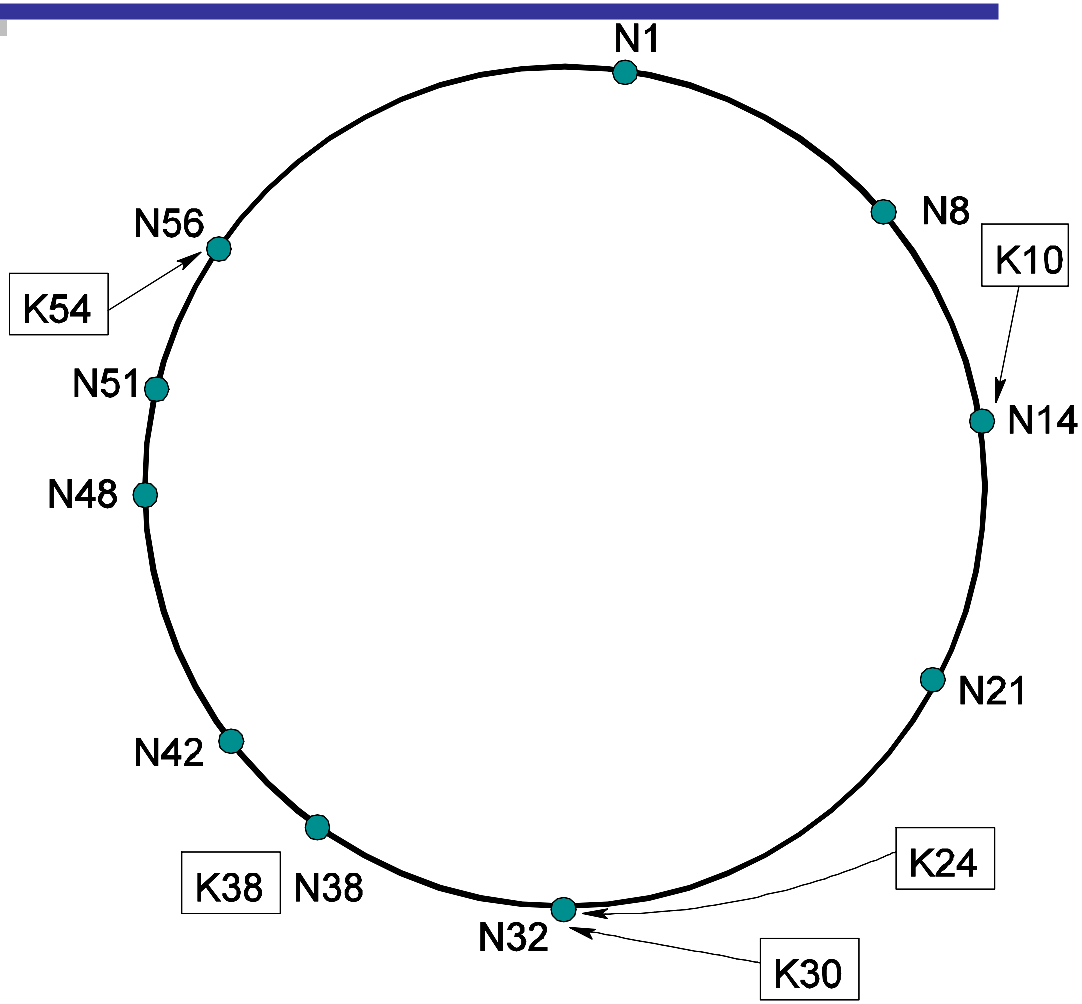
Basic Approach Differences

- Different P2P systems differ in:
  - the choice of the ID space
  - the structure of their network of nodes (i.e. how each node chooses its neighbors)

- The goals are
  - search in logarithmic time
  - insertions and deletions in logarithimic time
  - maintain good load balance

Chord

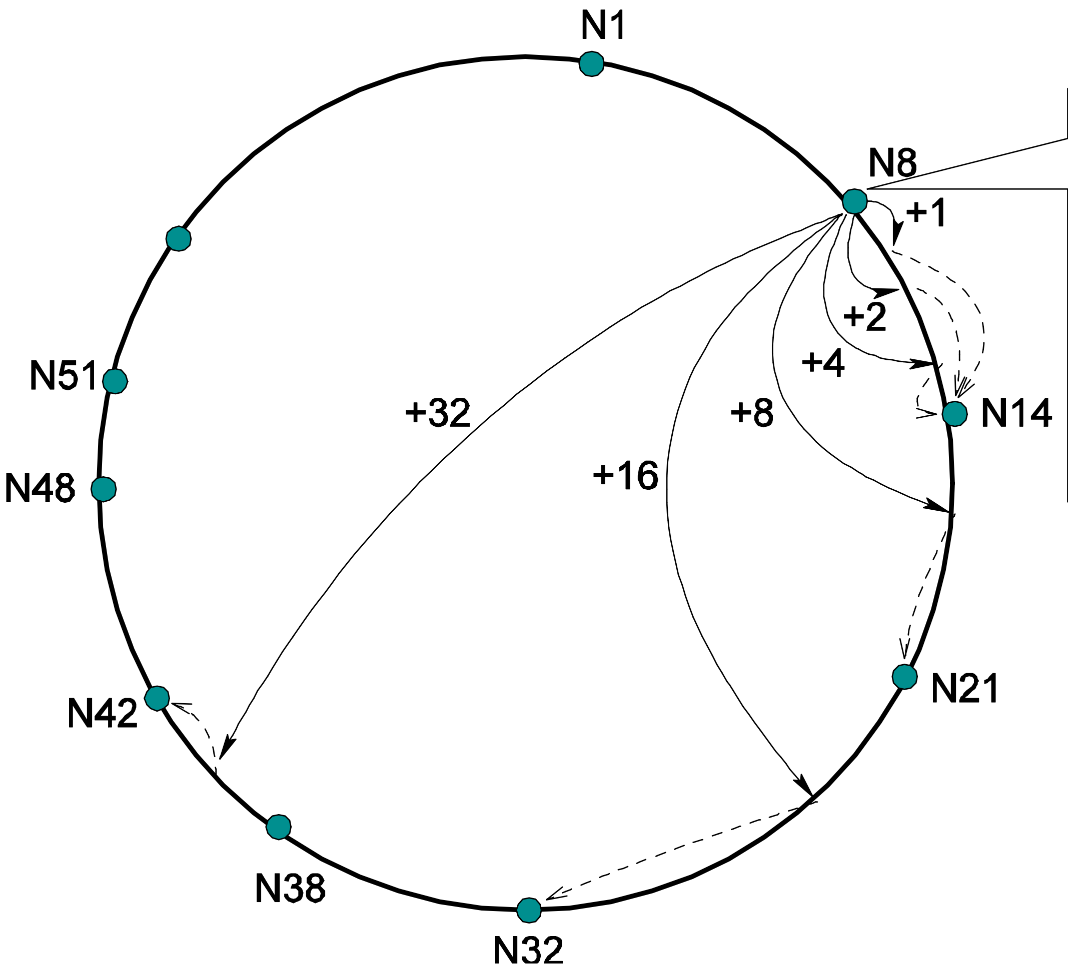
- Nodes organized in an identifier circle based on node identifiers
- Keys assigned to their successor node in the identifier circle
- Hash function ensures even distribution of nodes and keys on the circle



* All Chord figures from "Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications", Ion Stoica et al., IEEE/ACM Transactions on Networking, Feb. 2003.

Chord Finger Table

- O(logN) table size
- i^{th} finger points to first node that succeeds n by at least 2^{i-1}
- maintain also pointers to predecessors (for correctness)

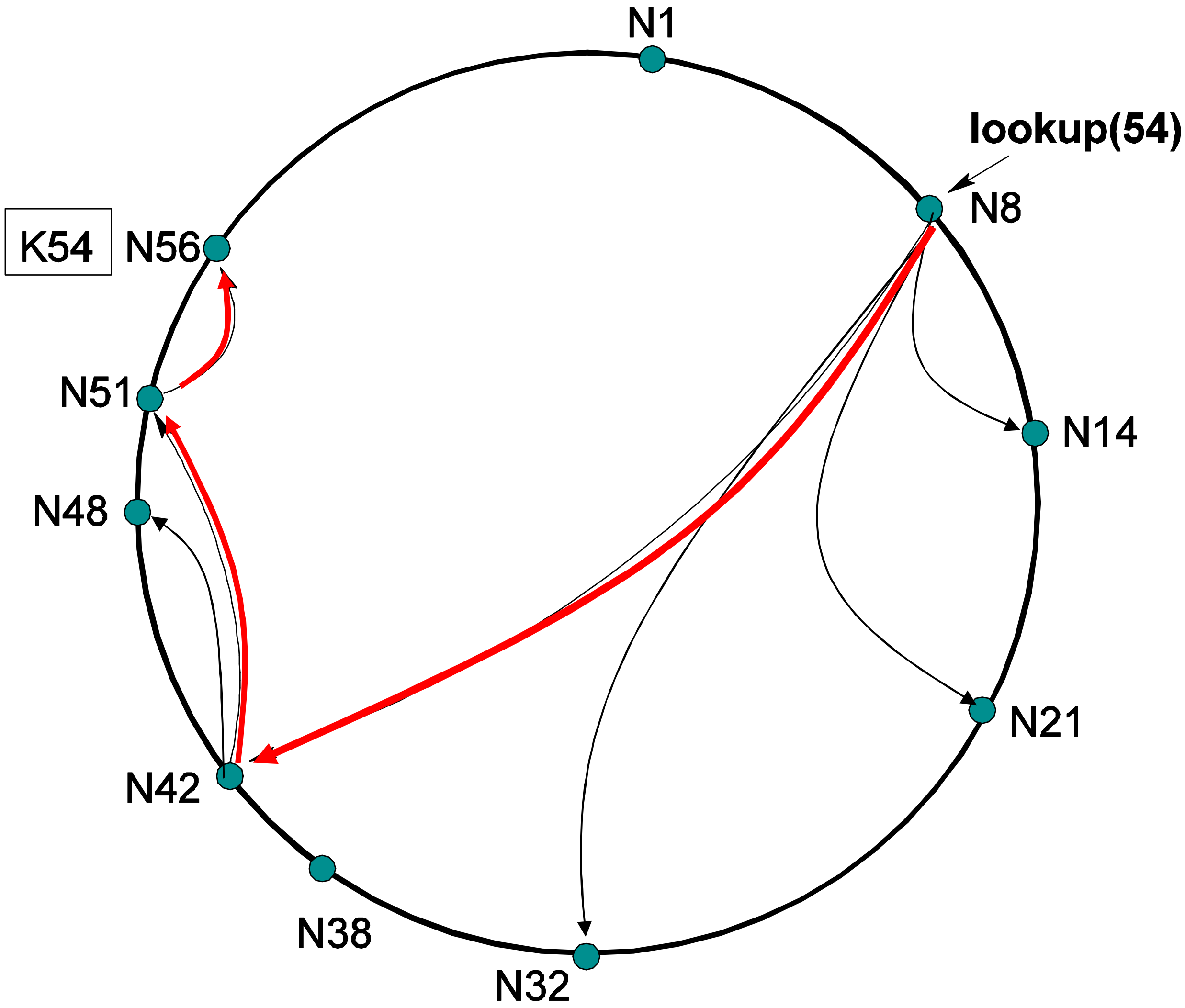


Finger table

N8 + 1	N14
N8 + 2	N14
N8 + 4	N14
N8 + 8	N21
N8 +16	N32
N8 +32	N42

Chord Key Location

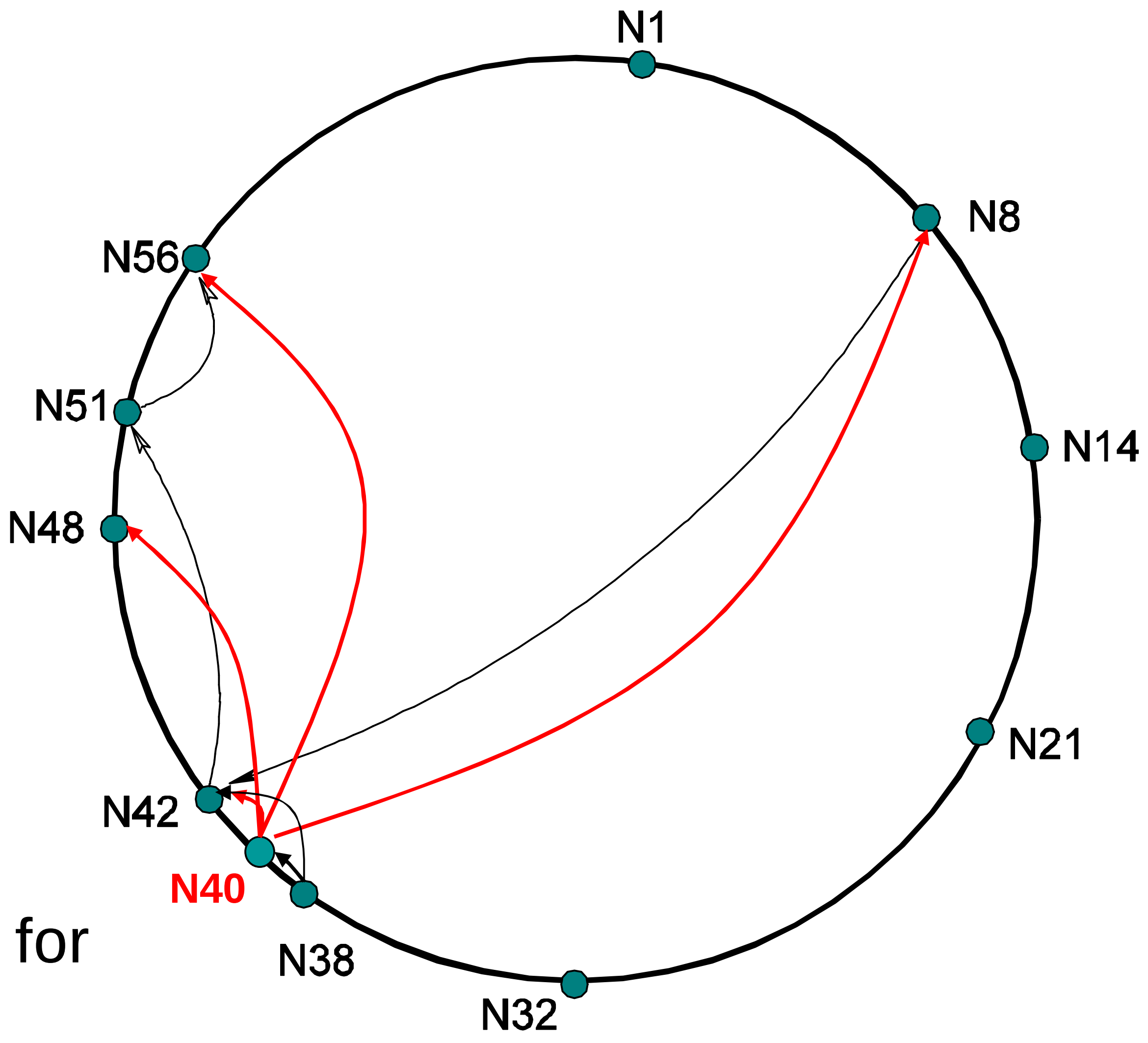
- Lookup in finger table the furthest node that precedes key
- Query homes in on target in O(logN) hops



Chord node insertion

Insert node N40:

- Locate node
- Add fingers
- Update successor pointers and other node's fingers (max in-degree $O(\log^2 n)$ whp)



Time $O(\log^2 n)$

Stabilization protocol for refreshing links

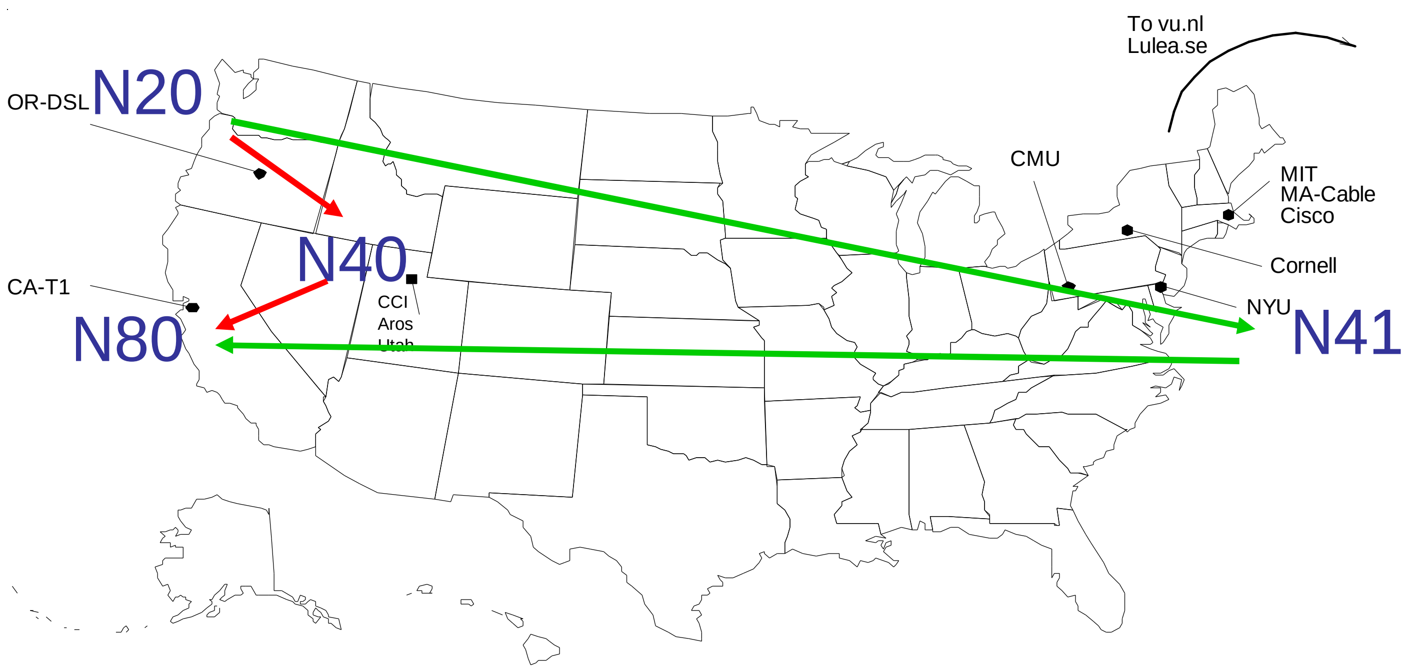
Chord Properties

- In a system with N nodes and K keys, with high probability...
  - each node receives at most K/N keys
  - each node maintains info. about O(logN) other nodes
  - lookups resolved with O(logN) hops
  - Insertions $O(\log^2 N)$

- In practice never stabilizes
- No consistency among replicas
- Hops have poor network locality

Network locality

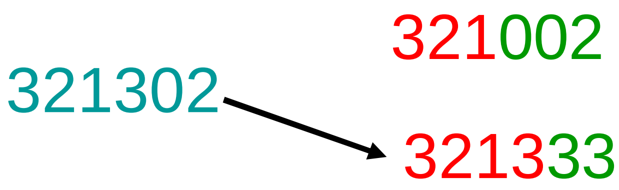
- Nodes close on ring can be far in the network.



* Figure from http://project-iris.net/talks/dht-toronto-03.ppt

Plaxton’s Mesh

- map the nodes and keys to b-ary numbers of m digits
- assign each key to the node with which it shares the largest prefix
  - e.g. b = 4 and m = 6



Plaxton's Mesh – Routing Table

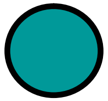
- for b = 4, m = 6, nodeID = 110223; routing table:

	d = 0	d = 1	d = 2	d = 3
p = 0	032130	1	210231	303213
p = 1	103002	1	123011	133233
p = 2	0	111210	112301	113331
p = 3	110031	110122	2	110310
p = 4	110200	110212	2	110232
p = 5	110220	110221	110222	3

Plaxton algorithm: routing

Move closer to the target one digit at the time

locate
322210

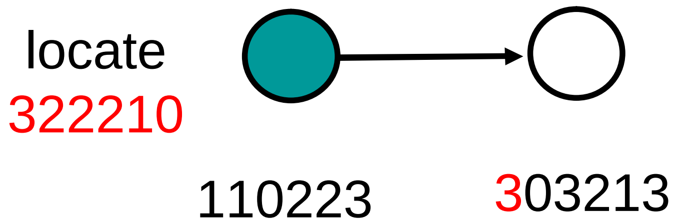


110223

	d = 0	d = 1	d = 2	d = 3
p = 0	032130	1	210231	303213
p = 1	103002	1	123011	133233
p = 2	0	111210	112301	113331
p = 3	110031	110122	2	110310
p = 4	110200	110212	2	110232
p = 5	110220	110221	110222	3

Plaxton algorithm: routing

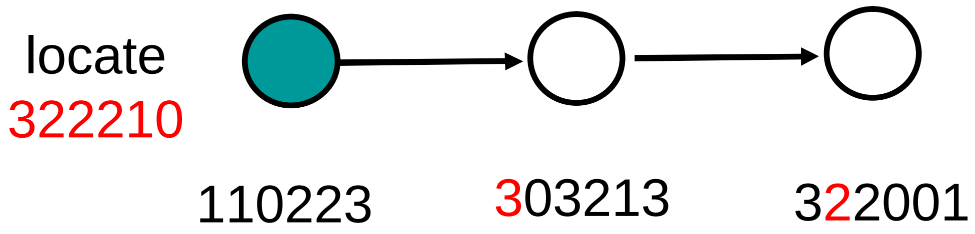
Move closer to the target one digit at the time



	d = 0	d = 1	d = 2	d = 3
p = 0	032130	1	210231	303213
p = 1	103002	1	123011	133233
p = 2	0	111210	112301	113331
p = 3	110031	110122	2	110310
p = 4	110200	110212	2	110232
p = 5	110220	110221	110222	3

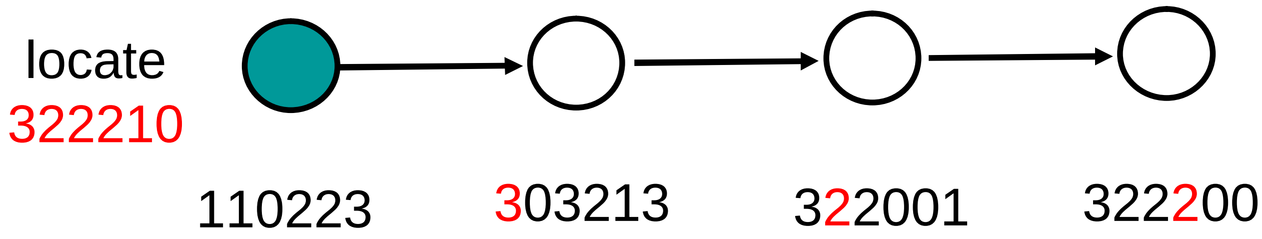
Plaxton algorithm: routing

Move closer to the target one digit at the time



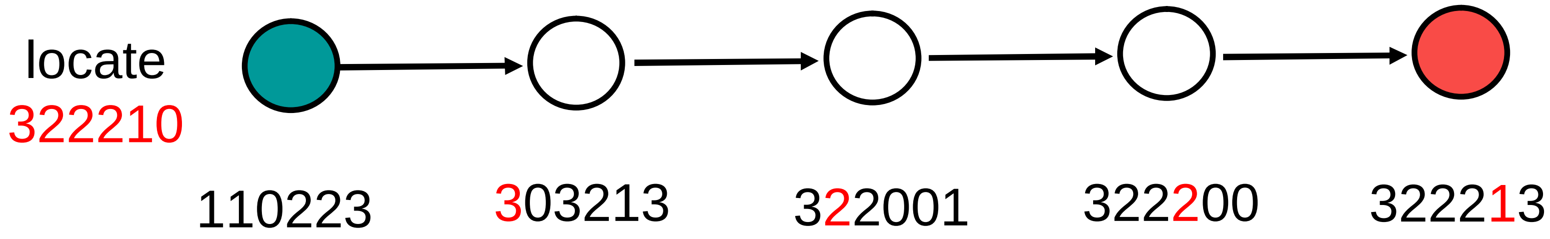
Plaxton algorithm: routing

Move closer to the target one digit at the time



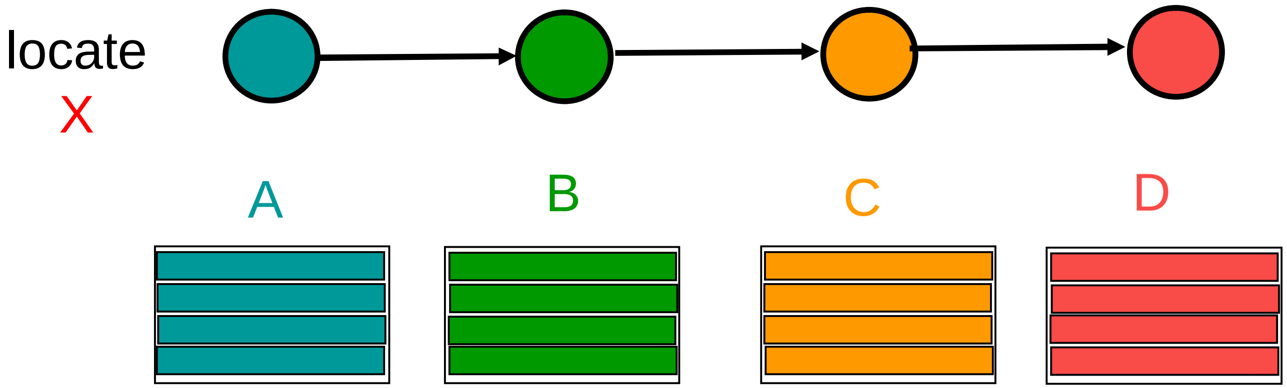
Plaxton algorithm: routing

Move closer to the target one digit at the time



Pastry: Node Joins

- Node X finds the closest (in network proximity) node and makes a query with its own ID



- Routing table of X
  - the i-th row of the routing table is the i-th row of the i-th node along the search path for X

Enforcing Network Locality - Pastry

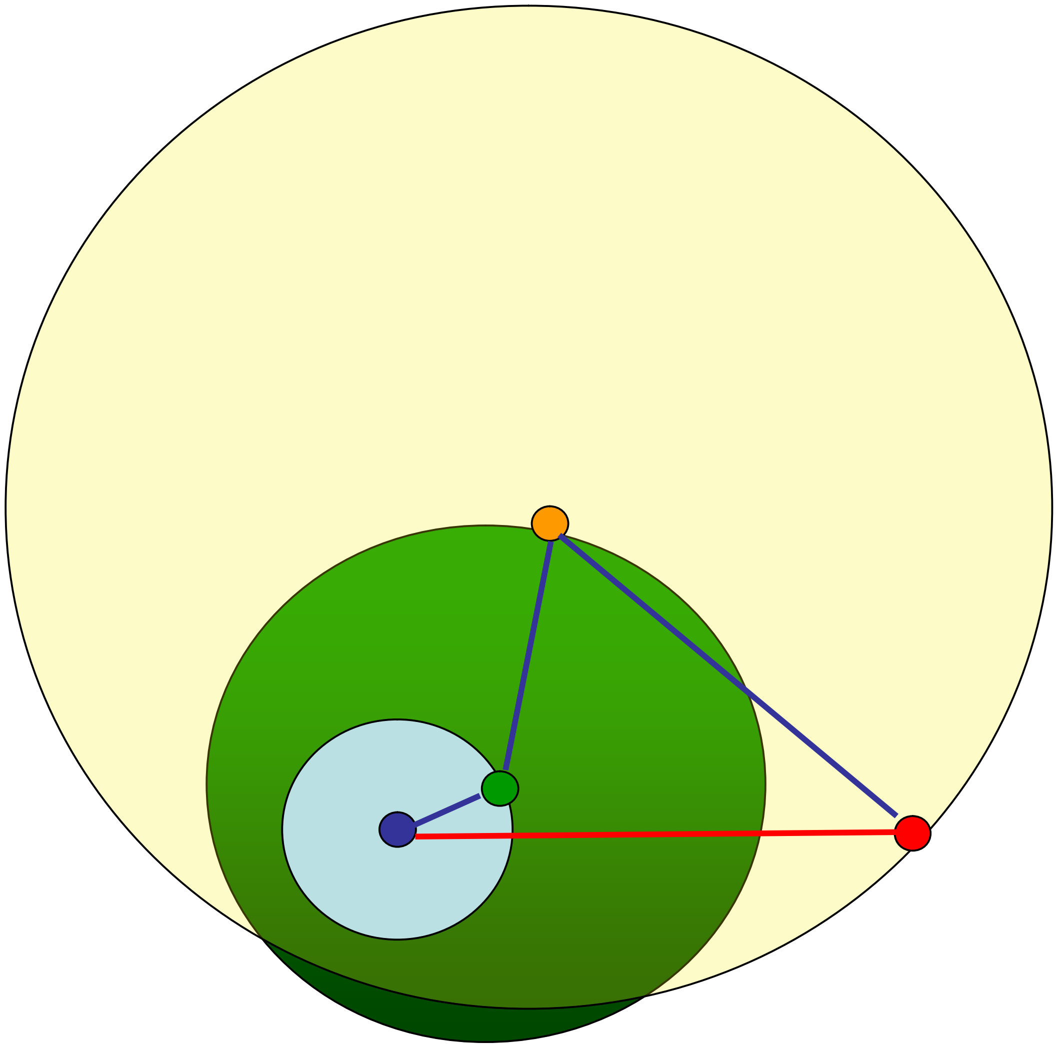
- For the (i,j) entry of the table select the node that is geographically closer to the current node.

110223	d = 0	d = 1	d = 2	d = 3
p = 0	032130	1	210231	303213
p = 1	103002	1	123011	133233
p = 2	0	111210	112301	113331
p = 3	110031	110122	2	110310
p = 4	110200	110212	2	110232
p = 5	110220	110221	110222	3

Enforcing Network Locality -Pastry

- Critical property
  - for larger row numbers the number of possible choices decreases exponentially
    - in row i+1 we have 1/b the choices we had in row i
  - for larger row numbers the expected distance to the nearest neighbor increases exponentially
  - the distance of the source to the target is approximately equal to the distance in the last step – as a result it is well approximated

Enforcing Network Locality - Pastry



Network Proximity

- The starting node A is the closest one to node X, so by triangular inequality the neighbors in first row of the starting node A will also be close to X

- For the remaining entries of the table the same argument applies: the distance of the intermediate node Y to its neighbors dominates the distance from X to the intermediate node Y

CAN

- Search space: d-dimensional coordinate space (on a d-torus)
- Each node owns a distinct zone in the space
- Each node keeps links to the nodes responsible for zones adjacent to its zone (in the search space) – ~2d on avg
- Each key hashes to a point in the space

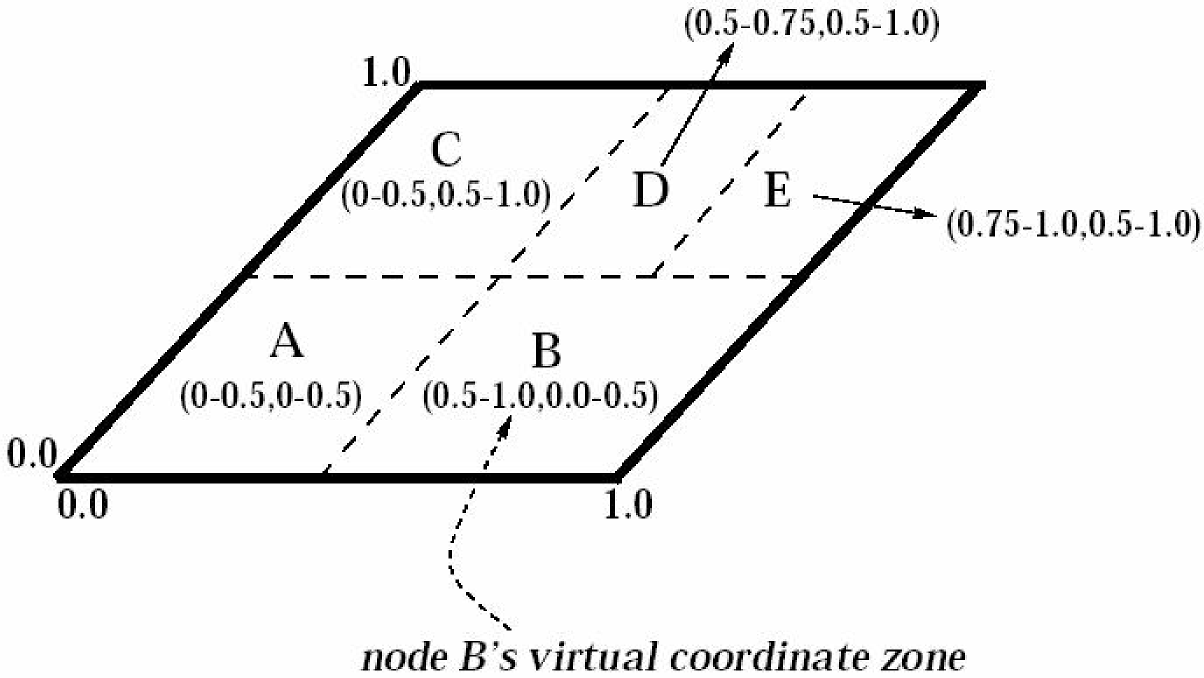


Figure 1: ***Example 2-d space with 5 nodes***

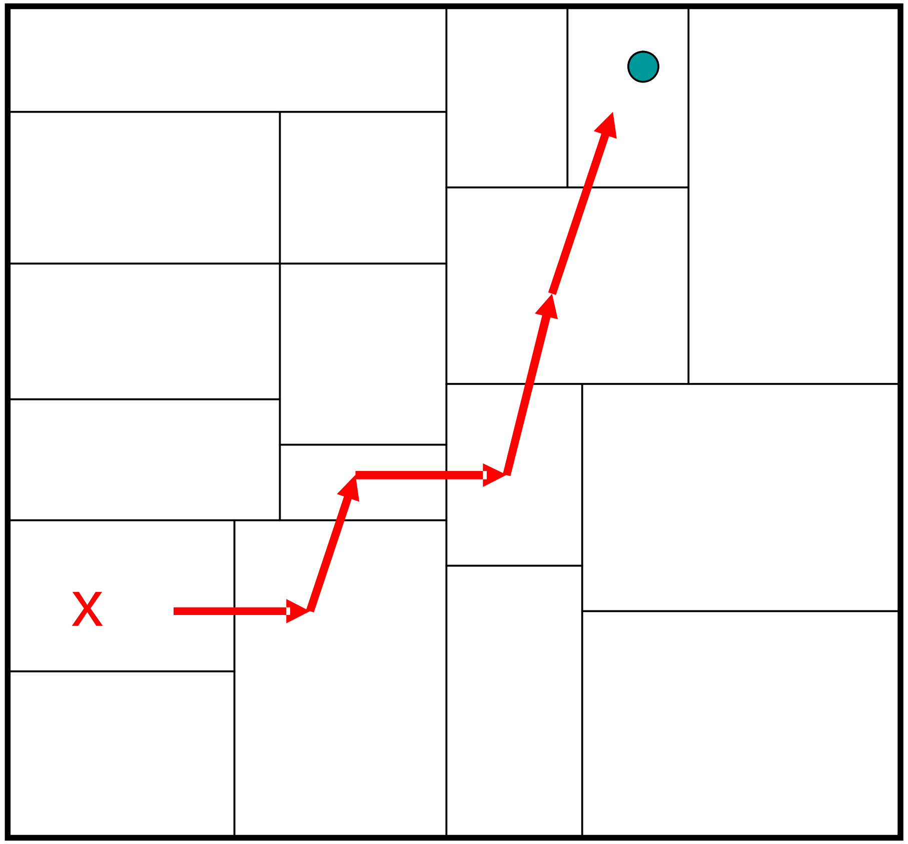
* Figure from "A Scalable Content-Addressable Network", S. Ratnasamy et al., In Proceedings of ACM SIGCOMM 2001.

CAN Lookup

Node x wants to lookup key K

K→(a,b)

Move along neighbors to the zone of the key each time moving closer to the key



expected time $O(dn^{1/d})$

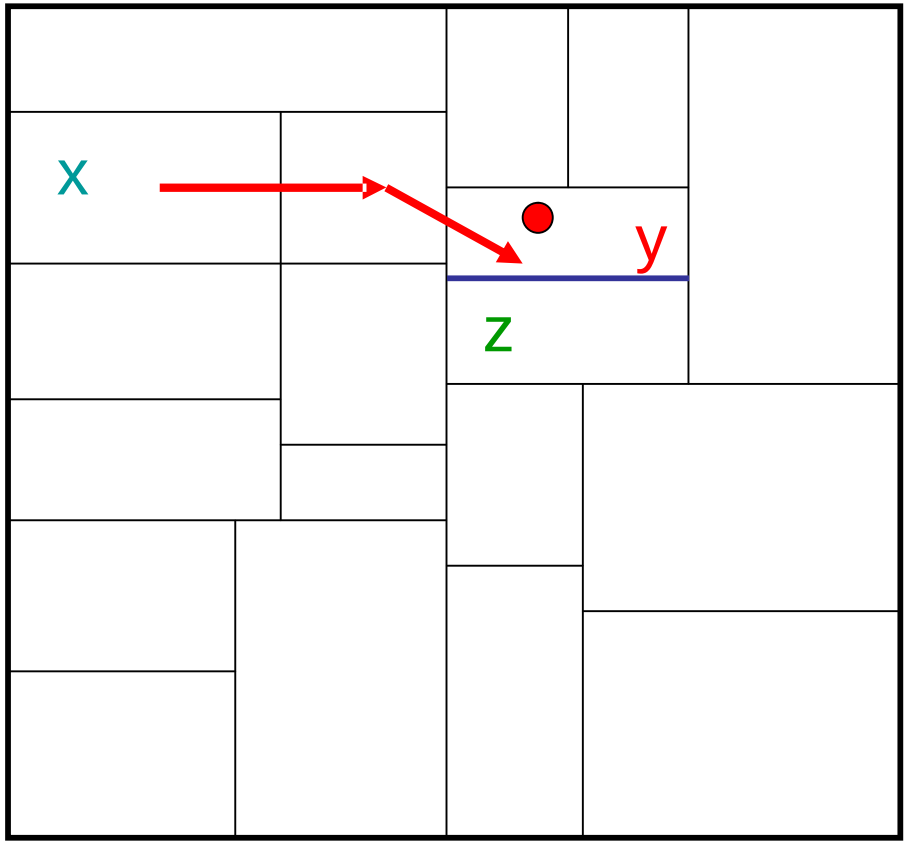
can we do it in O(logn)?

CAN node insertion

Node y needs to be inserted
It has knowledge of node x

IP of y → (c,d)
zone belongs to z

Split z's zone



Kademlia

- The routing idea is similar to Plaxton's mesh: improve closeness one bit at the time.

Kademlia – Hashing and distance

- Nodes and Keys are mapped to m-bit binary strings
- Distance between two identifiers: the XOR string, as a binary number

x = 0 1 0 1 1 0
y = 0 1 1 0 1 1

x⊕y = 0 0 1 1 0 1

d(x,y) = 13

XOR: 1 if two bits are different
0 if two bits are the same

- If x and y agree in the first i digits and disagree in the (i+1) then $2^i \leq d(x,y) \leq 2^{i+1}-1$

x = 0 1 0 1 1 0
y = 0 1 1 1 1 0

x⊕y = 0 0 1 0 0 0

d(x,y) = 8

x = 0 1 0 1 1 0
y = 0 1 1 0 0 1

x⊕y = 0 0 1 1 1 1

d(x,y) = 15

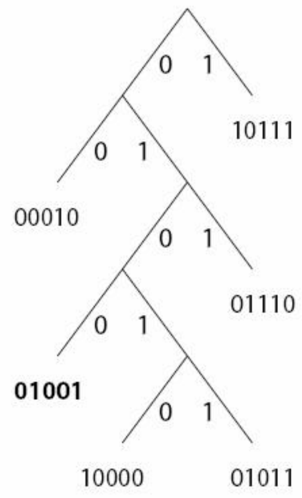
symmetric
unidirectional

Kademlia – Routing table

- Each node with ID x, stores m k-buckets
  - a k-bucket stores k nodes that are at distance $[2^i, 2^{i+1})$
    - empty bucket if no nodes are known

Routing table for node 01001

k = 1



Kademlia – Updating buckets

- Whenever a node receives any message, it updates the appropriate k-bucket
- If the bucket is full the least-recently node is removed if it is not live

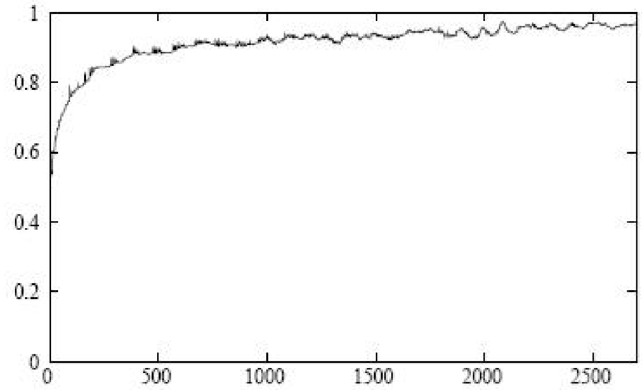


Figure 1: Probability of remaining online another hour as a function of uptime. The x axis represents minutes. The y axis shows the the fraction of nodes that stayed online at least x minutes that also stayed online at least $x + 60$ minutes.

Kademlia – Node lookup

- The lookup process is iterative: everything is controlled by the initiator node
  1. query in parallel the α nodes closest to the query ID
  2. nodes return the k nodes closest to the query ID
  3. go back to step 1, and select the α nodes from the new set of nodes
  4. Terminate when you have the k closest nodes
- Key lookups are done in a similar fashion, but they terminate when the key is found
  - the requesting node cashes the key locally.

Kademlia – Other operations

- Refresh
  - periodically, all k-buckets are refreshed by making a query for a value within the bucket

- Node Joins
  - contact a participating node and insert it in the appropriate bucket
  - perform a query for your own ID
  - refresh all buckets

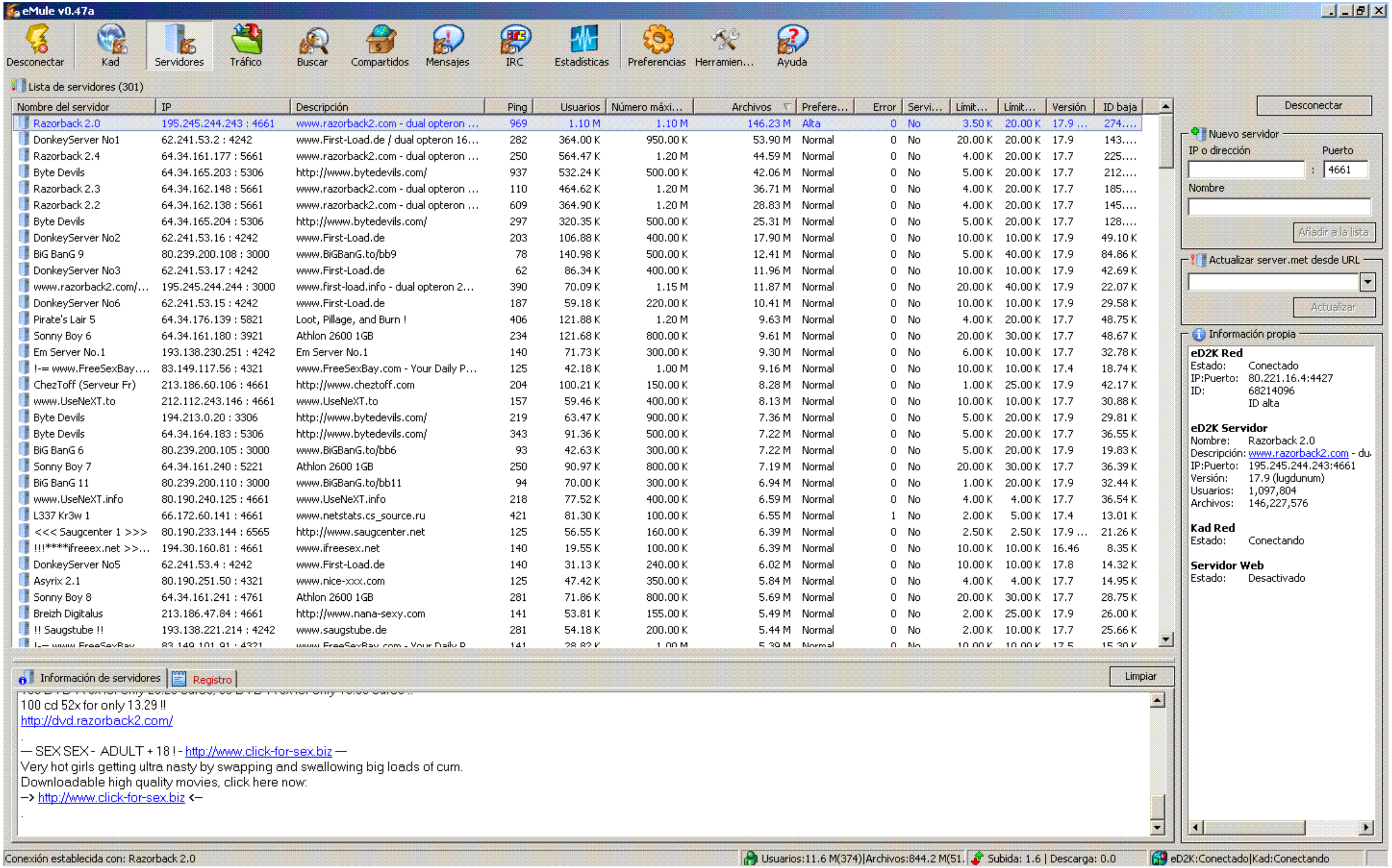
Kademlia – Lookups

- Invariant: If there exists some node with ID within a specific range then the k-bucket is not empty
  - if the invariant is true, then the time is logarithmic
    - we move one bit closer each time
- Due to refreshes the invariant holds with high probability

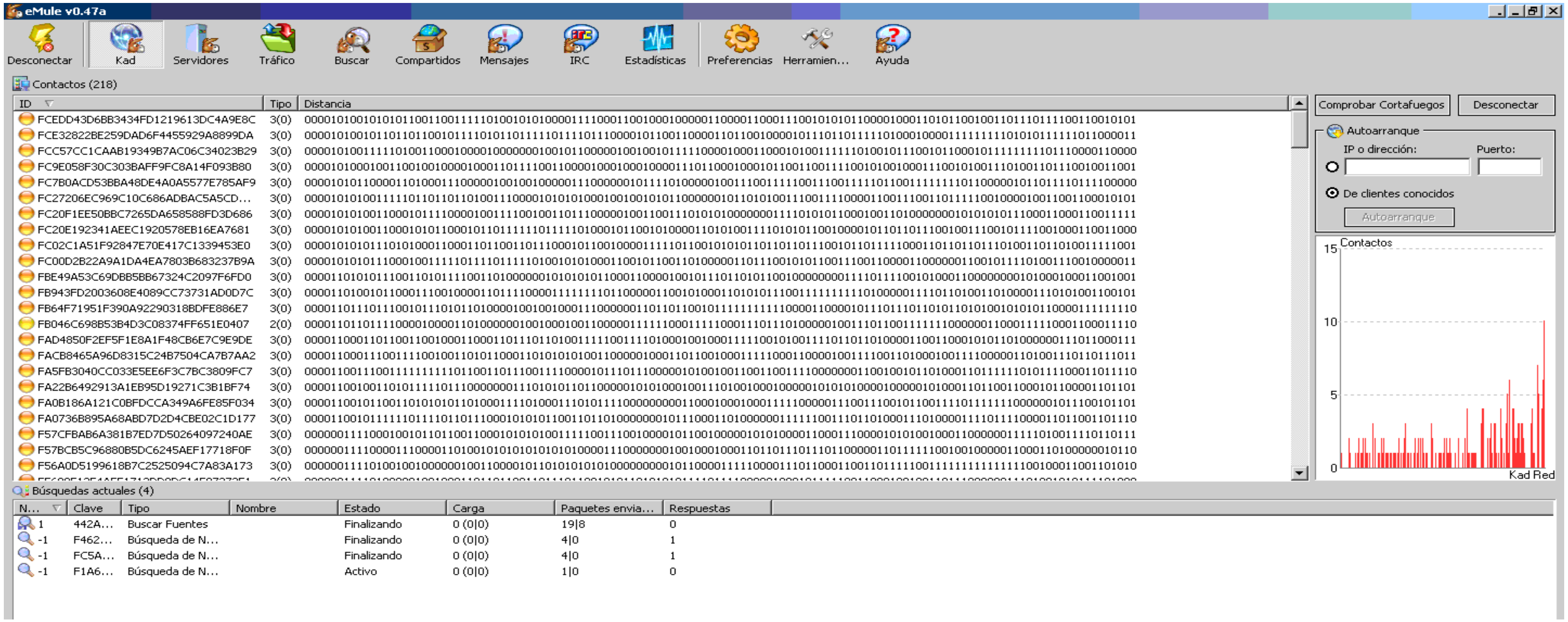
Kademlia - Properties

- Easy table maintenance. Tables are updated when lookups are performed
  - due to XOR symmetry a node receives lookups from the nodes that are in its own table
- Fast lookup by making parallel searches
  - at the expense of increased traffic.

e-mule snapshots

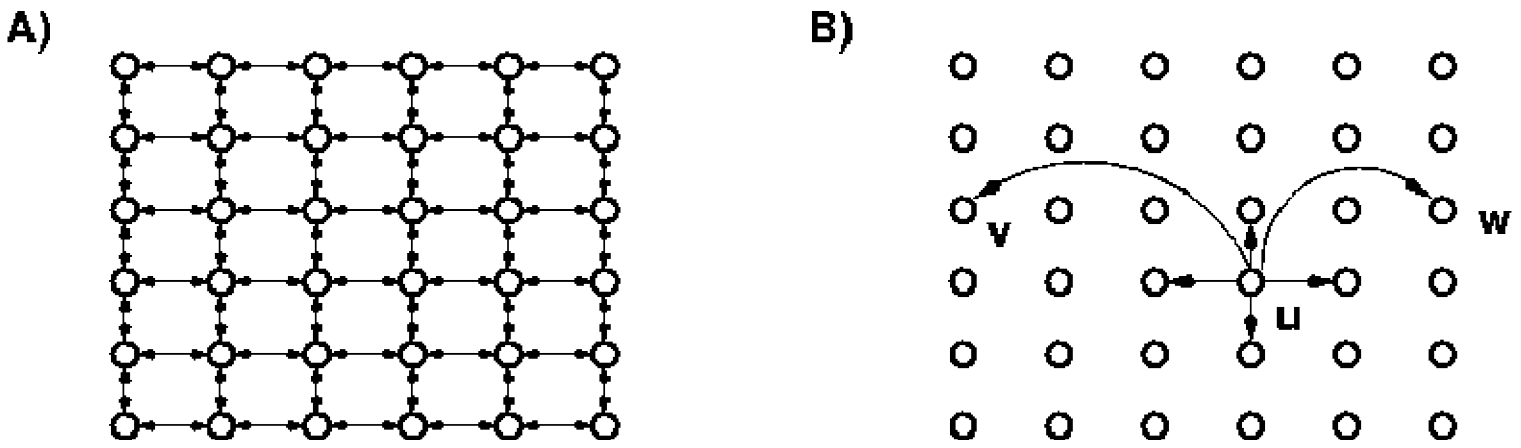


e-mule snapshots



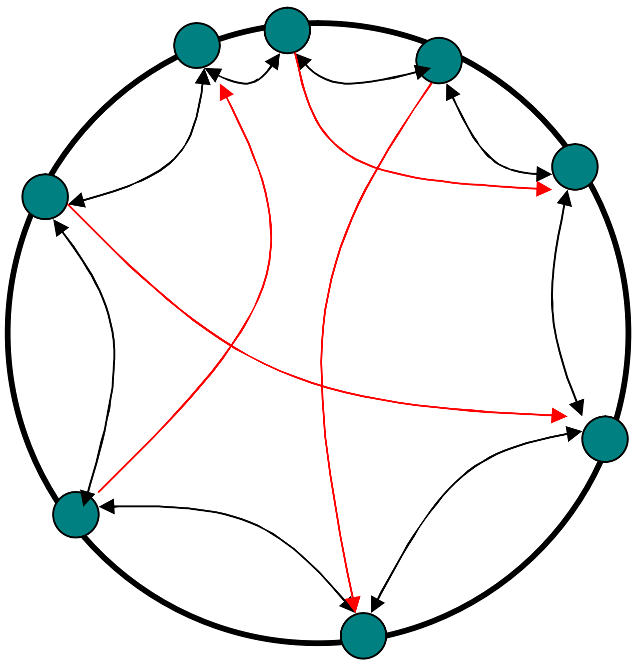
Kleinberg's small world

- Consider a 2-dimensional grid
- For each node u add edge (u,v) to a vertex v selected with pb proportional to $[d(u,v)]^{-r}$
- Simple Greedy routing
  - If $r=2$, expected lookup time is $O(\log^2 n)$
  - If $r \neq 2$, expected lookup time is $\Omega(n^{\varepsilon})$, ε depends on r
- The theorem generalizes in d-dimensions for $r=d$



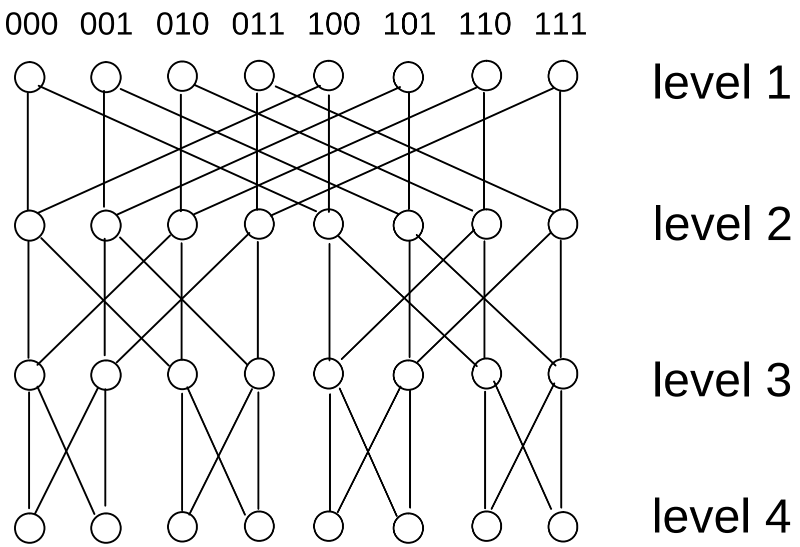
Symphony

- Map the nodes and keys to the ring
- Link every node with its successor and predecessor
- Add k random links with probability proportional to $1/(d \log n)$, where d is the distance on the ring
- Lookup time $O(\log^2 n)$
- If $k = \log n$ lookup time $O(\log n)$
- Easy to insert and remove nodes (perform periodical refreshes for the links)



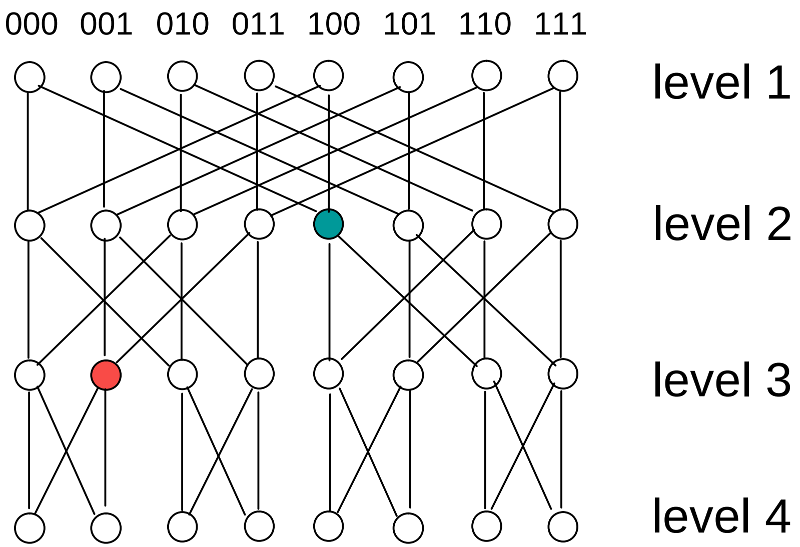
Viceroy

- Emulating the butterfly network



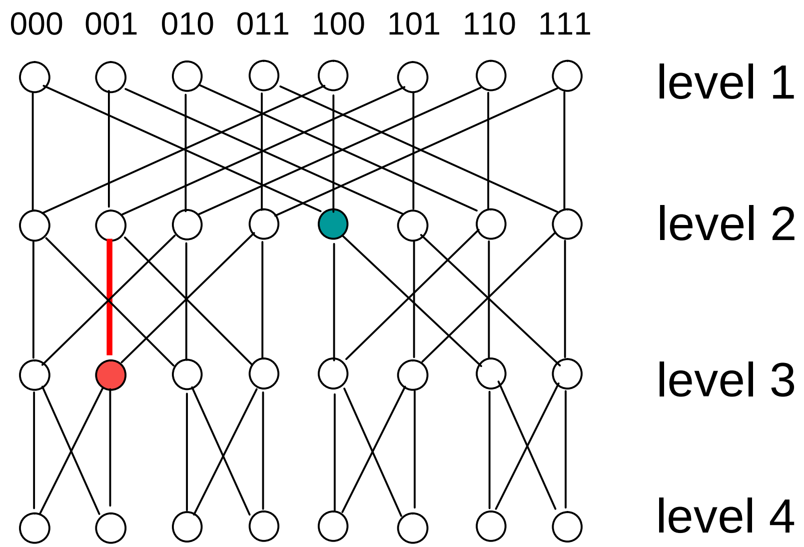
Viceroy

§ Emulating the butterfly network



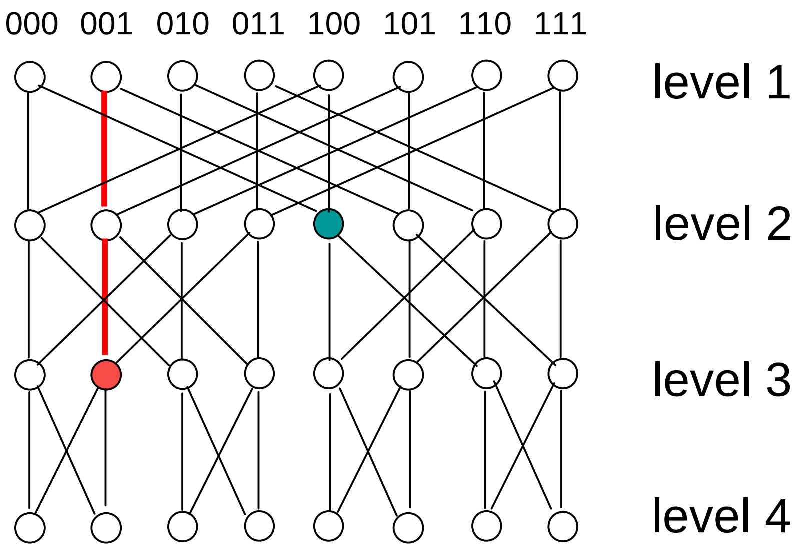
Viceroy

- Emulating the butterfly network



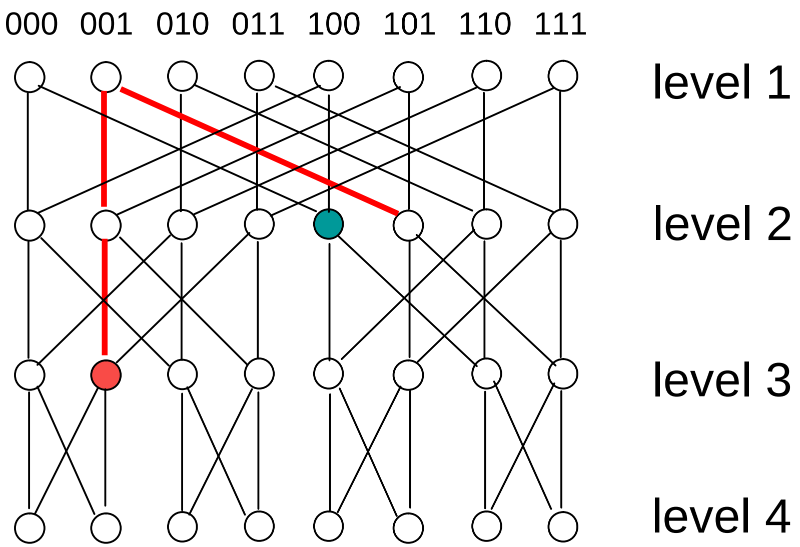
Viceroy

- Emulating the **butterfly** network



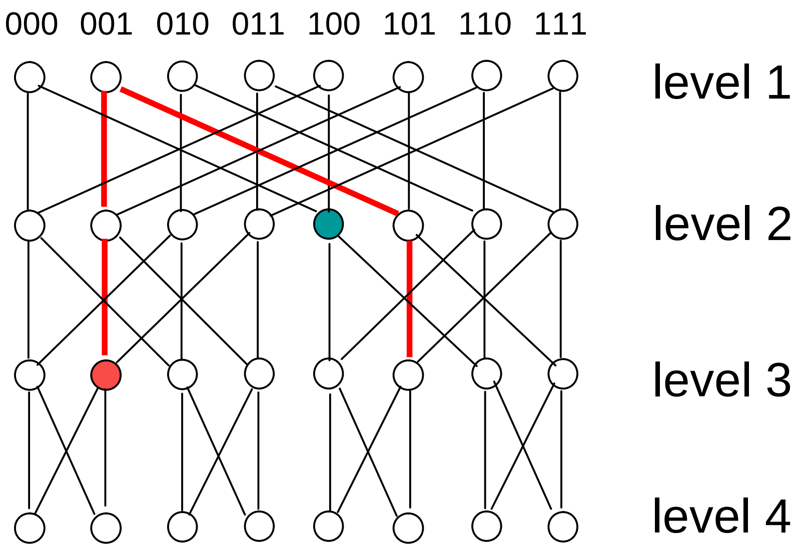
Viceroy

- Emulating the butterfly network



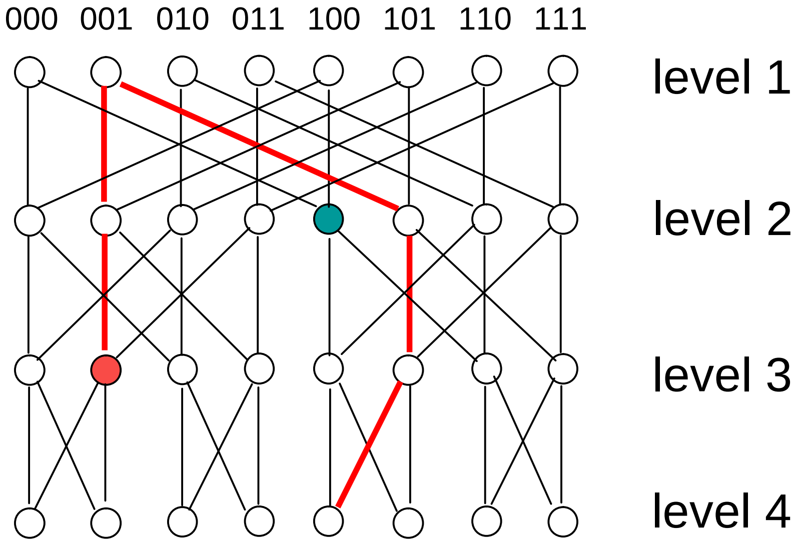
Viceroy

- Emulating the **butterfly** network



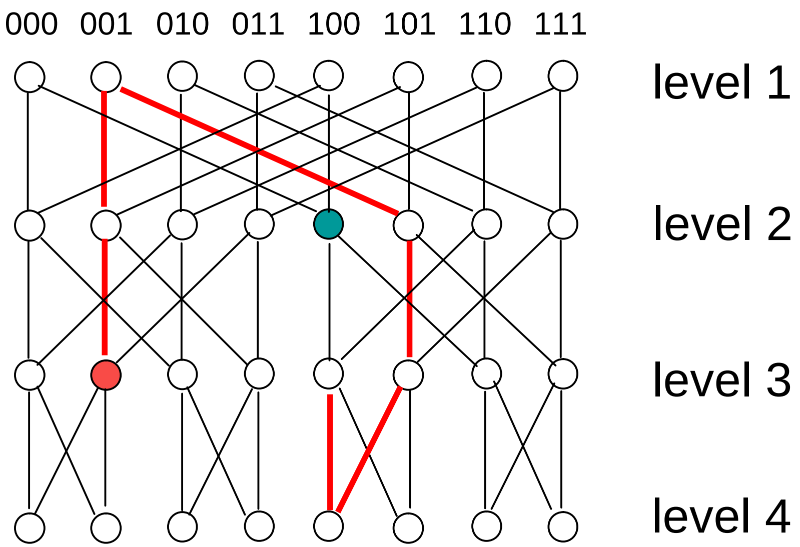
Viceroy

- Emulating the butterfly network



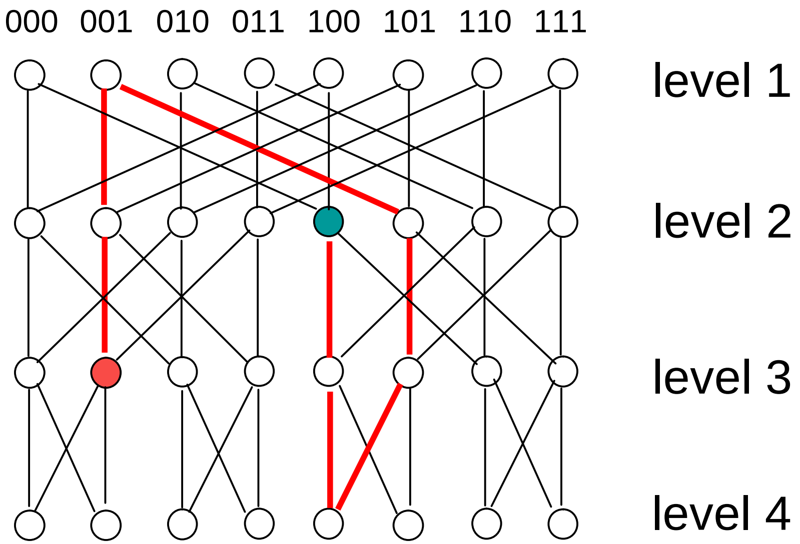
Viceroy

§ Emulating the butterfly network



Viceroy

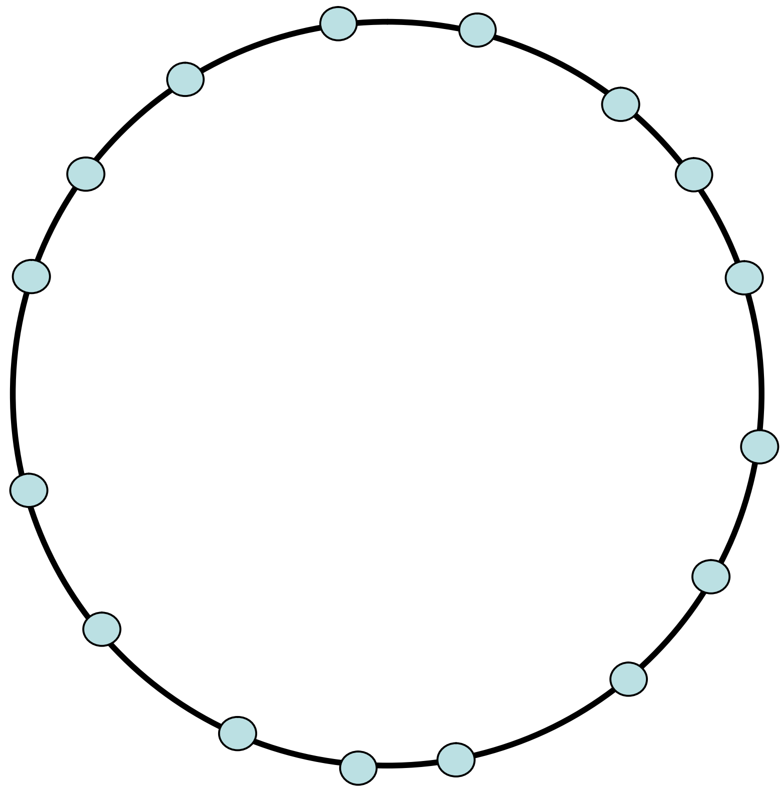
- Emulating the butterfly network



- Logarithmic path lengths between any two nodes in the network
- Constant degree per node

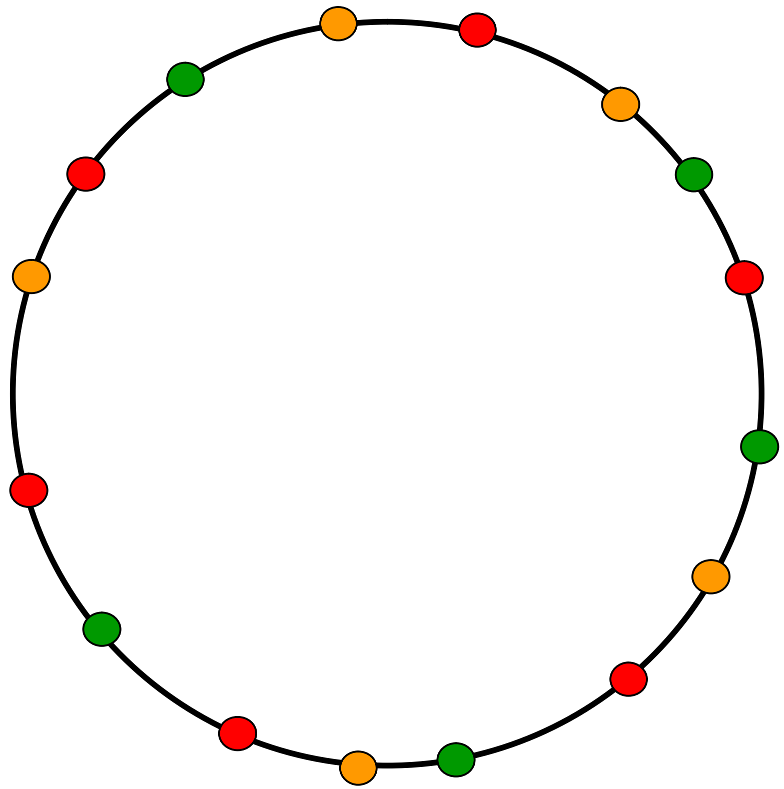
Viceroy network

- Arrange nodes and keys on a ring, like in Chord.



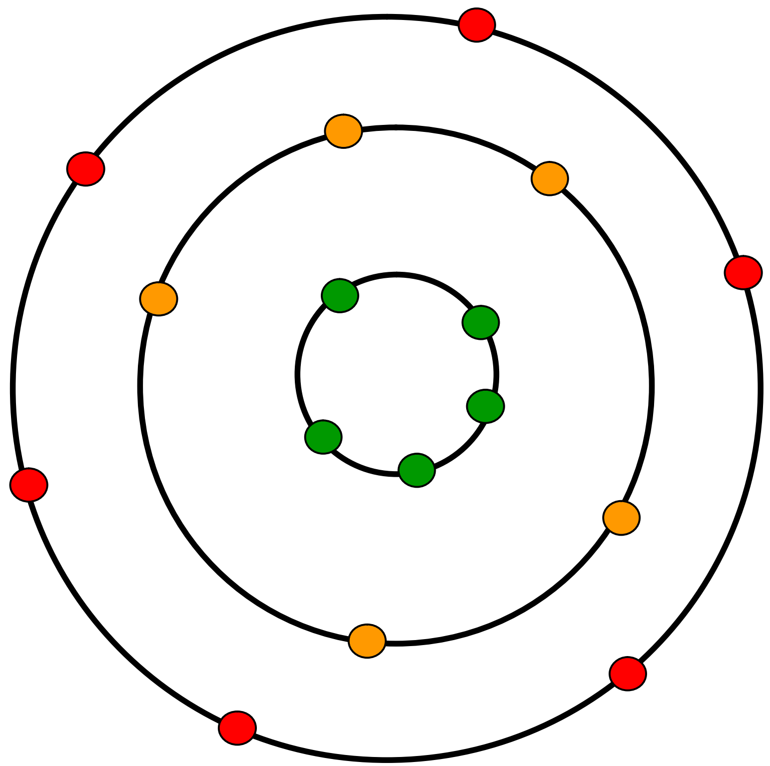
Viceroy network

- Assign to each node a level value, chosen uniformly from the set {1,...,logn}
  - estimate n by taking the inverse of the distance of the node with its successor
  - easy to update



Viceroy network

§ Create a ring of nodes within the same level



Butterfly links

- Each node x at level i has two downward links to level i+1
  - a left link to the first node of level i+1 after position x on the ring
  - a right link to the first node of level i+1 after position $x + (½)^i$

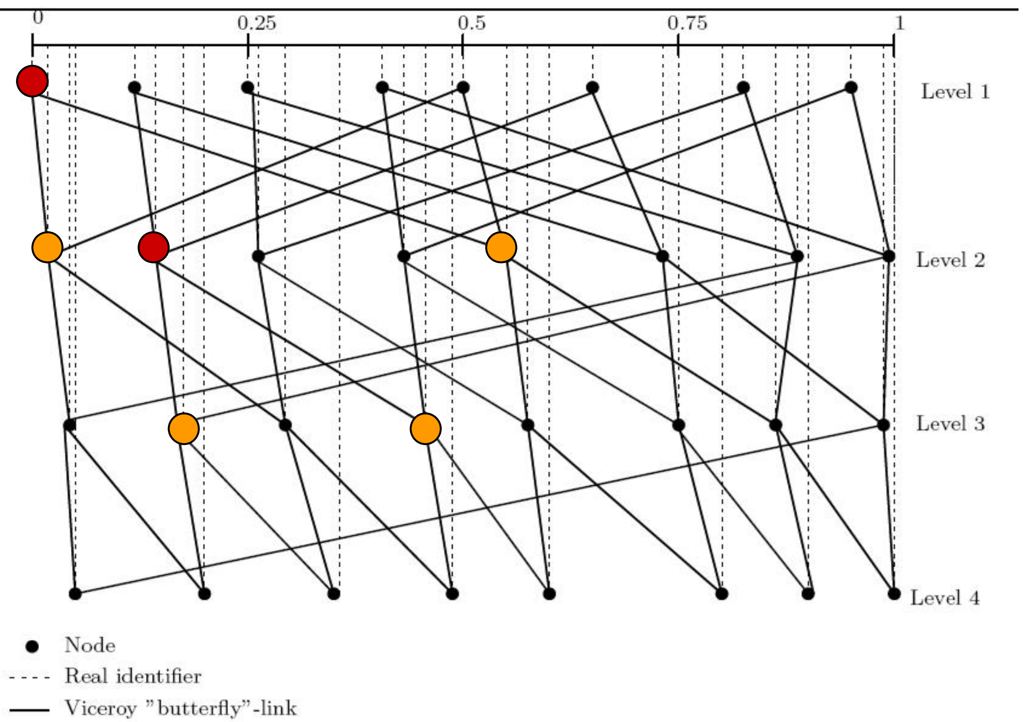
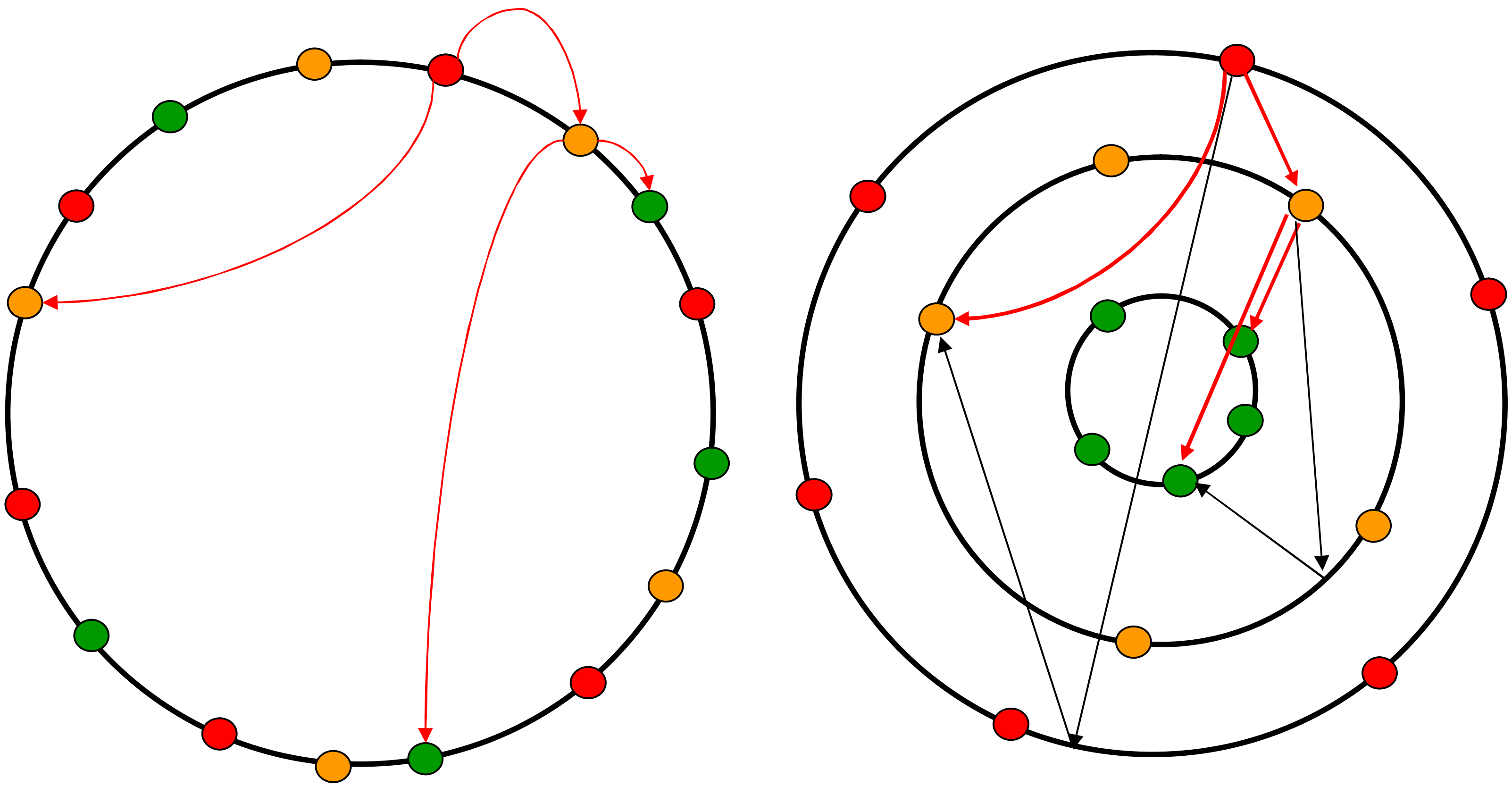


Figure 1: An ideal Viceroy network. Up and ring links are omitted for simplicity.

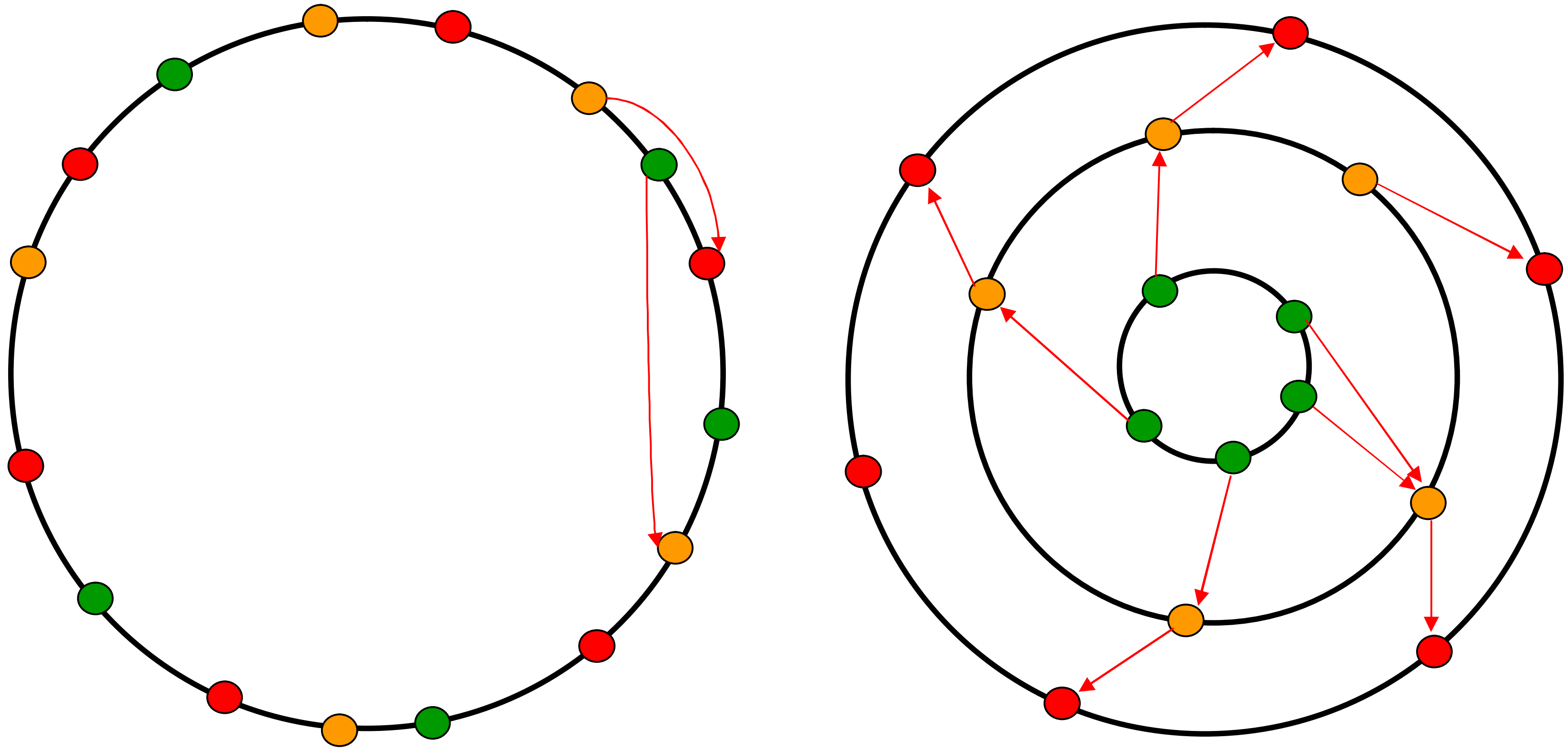
Downward links



Upward links

- Each node x at level i has an upward link to the next node on the ring at level i-1

Upward links



Lookup

- Lookup is performed in a similar fashion like the butterfly
  - expected time O(logn)

- Viceroy was the first network with constant number of links and logarithmic lookup time

P2P Review

- Two key functions of P2P systems
  - Sharing content
  - Finding content

- Sharing content
  - Direct transfer between peers
    - All systems do this
  - Structured vs. unstructured placement of data
  - Automatic replication of data

- Finding content
  - Centralized (Napster)
  - Decentralized (Gnutella)
  - Probabilistic guarantees (DHTs)

Distributed Hash Tables

- Consistent Hashing: map both keys and nodes to the same space
  - guarantees good load balancing properties

- Routing (overlay) networks
  - degree d per node (usually O(logn))
  - number of hops O(logn/logd) (or O(logn))
  - can it be made to be fault tolerant?

Acknowledgements

- Thanks to Vinod Muthusamy, George Giakkoupis, Jim Kurose, Brian, Levine, Don Towsley

References

- G. Giakkoupis, Routing algorithms for Distributed Hash Tables, Technical Report, Univeristy of Toronto, 2003
- Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System," in Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability, LNCS 2009
- S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker. A Scalable Content-Addressable Network. ACM SIGCOMM, 2001
- I. Stoica, R. Morris, D. Karger, F. Kaashoek, H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. ACM SIGCOMM, 2001.
- A. Rowstron, P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001).
- P. Maymounkov and D. Mazieres. Kademlia: A peer-to -peer information system based on the XOR metric. In Proceedings of IPTPS02,
- B. Y. Zhao, J. D. Kubiatowicz, A. D. Joseph, Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing. UC Berkeley Computer Science Division, Report No. UCB/CSD 01/1141, April 2001.