P2P Systems

Keith W. Ross
Polytechnic University
http://cis.poly.edu/~ross
ross@poly.edu

Dan Rubenstein
Columbia University
http://www.cs.columbia.edu/~danr
danr@ee.columbia.edu

Thanks to: B. Bhattacharjee, A. Rowston, Don Towsley

Defintion of P2P

1) Significant autonomy from central servers

2) Exploits resources at the edges of the Internet
   - storage and content
   - CPU cycles
   - human presence

3) Resources at edge have intermittent connectivity, being added & removed

It's a broad definition:

- P2P file sharing
  - Napster, Gnutella, KaZaA, eDonkey, etc
- P2P communication
  - Instant messaging
  - Voice-over-IP: Skype
- P2P computation
  - seti@home
- DHTs & their apps
  - Chord, CAN, Pastry, Tapestry
- P2P apps built over emerging overlays
  - PlanetLab

Wireless ad-hoc networking not covered here

Tutorial Outline (1)

- 1. Overview: overlay networks, P2P applications, copyright issues, worldwide computer vision
- 2. Unstructured P2P file sharing: Napster, Gnutella, KaZaA, search theory, flashfloods
- 3. Structured DHT systems: Chord, CAN, Pastry, Tapestry, etc.

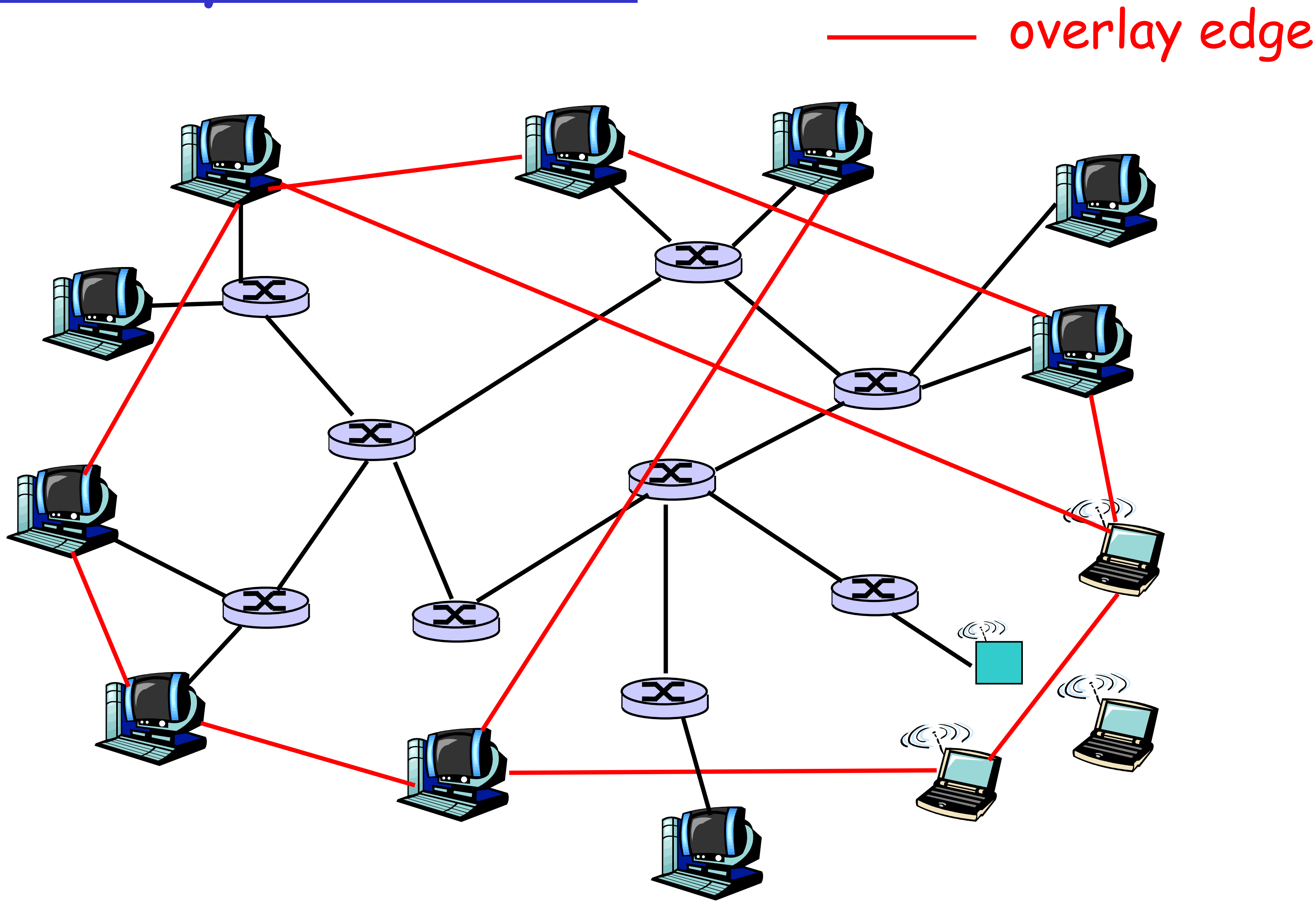
Tutorial Outline (cont.)

- 4. Applications of DHTs: persistent file storage, mobility management, etc.
- 5. Security issues: vulnerabilities, solutions, anonymity
- 6. Graphical structure: random graphs, fault tolerance
- 7. Experimental observations: measurement studies
- 8. Wrap up

1. Overview of P2P

- overlay networks
- P2P applications
- worldwide computer vision

Overlay networks



Overlay graph

Virtual edge

- TCP connection
- or simply a pointer to an IP address

Overlay maintenance

- Periodically ping to make sure neighbor is still alive
- Or verify liveness while messaging
- If neighbor goes down, may want to establish new edge
- New node needs to bootstrap

More about overlays

Unstructured overlays

- e.g., new node randomly chooses three existing nodes as neighbors

Structured overlays

- e.g., edges arranged in restrictive structure

Proximity

- Not necessarily taken into account

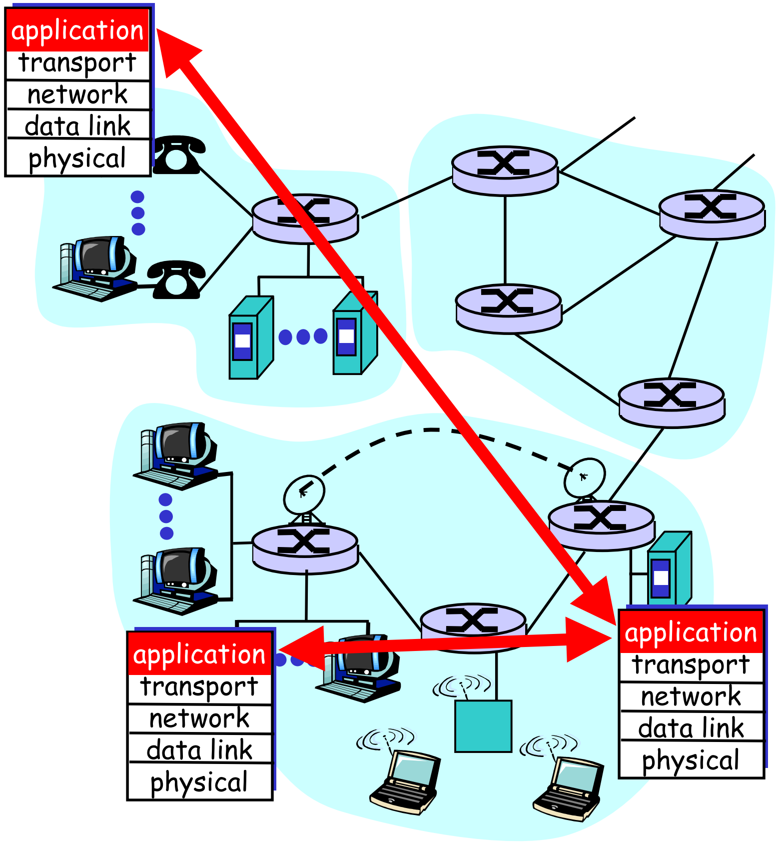
Overlays: all in the application layer

Tremendous design flexibility

- Topology, maintenance
- Message types
- Protocol
- Messaging over TCP or UDP

Underlying physical net is transparent to developer

- But some overlays exploit proximity



Examples of overlays

- DNS
- BGP routers and their peering relationships
- Content distribution networks (CDNs)
- Application-level multicast
  - economical way around barriers to IP multicast

- And P2P apps !

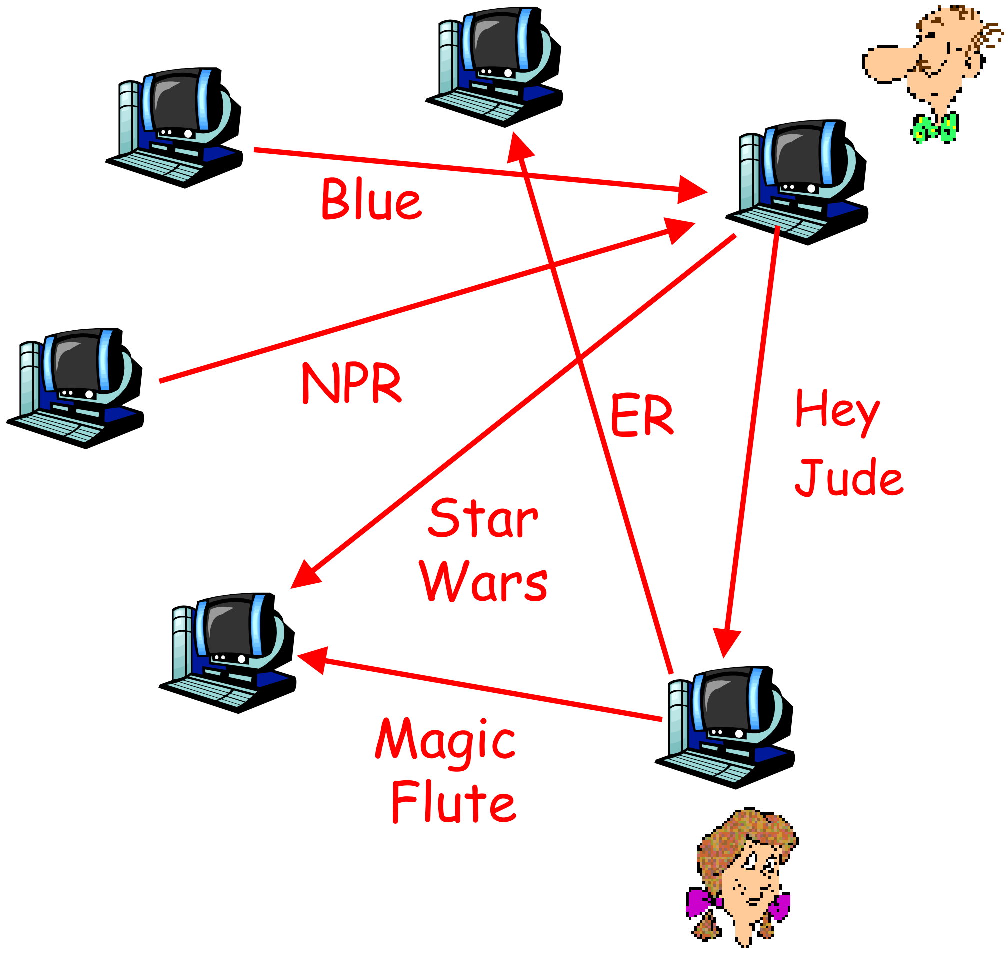
1. Overview of P2P

- overlay networks
- current P2P applications
  - P2P file sharing & copyright issues
  - Instant messaging / voice over IP
  - P2P distributed computing
- worldwide computer vision

P2P file sharing

- Alice runs P2P client application on her notebook computer
- Intermittently connects to Internet; gets new IP address for each connection
- Registers her content in P2P system
- Asks for "Hey Jude"
- Application displays other peers that have copy of Hey Jude.
- Alice chooses one of the peers, Bob.
- File is copied from Bob's PC to Alice's notebook: P2P
- While Alice downloads, other users uploading from Alice.

Millions of content servers



Killer deployments

- Napster
  - disruptive; proof of concept
- Gnutella
  - open source
- KaZaA/FastTrack
  - Today more KaZaA traffic then Web traffic!
- eDonkey / Overnet
  - Becoming popular in Europe
  - Appears to use a DHT

Is success due to massive number of servers, or simply because content is free?

P2P file sharing software

- Allows Alice to open up a directory in her file system
  - Anyone can retrieve a file from directory
  - Like a Web server
- Allows Alice to copy files from other users' open directories:
  - Like a Web client
- Allows users to search nodes for content based on keyword matches:
  - Like Google

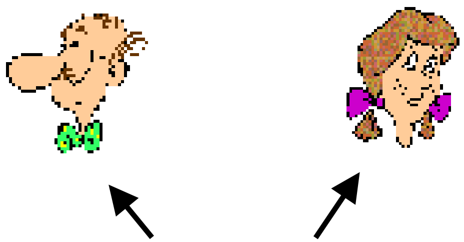
Copyright issues (1)

Direct infringement:

- end users who download or upload copyrighted works

Indirect infringement:

- Hold an individual accountable for actions of others
- Contributory
- Vicarious



direct infringers

Copyright issues (2)

Contributory infringer:

- knew of underlying direct infringement, and
- caused, induced, or materially contributed to direct infringement

Vicarious infringer:

- able to control the direct infringers (e.g., terminate user accounts), and
- derived direct financial benefit from direct infringement (money, more users)

(knowledge not necessary)

Copyright issues (3)

Betamax VCR defense

- Manufacturer not liable for contributory infringement
- "capable of substantial non-infringing use"
- But in Napster case, court found defense does not apply to all vicarious liability

Guidelines for P2P developers

- total control so that there's no direct infringement

or

- no control over users - no remote kill switch, automatic updates, actively promote non-infringing uses of product
- Disaggregate functions: indexing, search, transfer
- No customer support

Instant Messaging

- Alice runs IM client on her PC
- Intermittently connects to Internet; gets new IP address for each connection
- Registers herself with "system"
- Learns from "system" that Bob in her buddy list is active
- Alice initiates direct TCP connection with Bob: P2P
- Alice and Bob chat.

- Can also be voice, video and text.

We'll see that Skype is a VoIP P2P system

P2P Distributed Computing

seti@home

- Search for ET intelligence
- Central site collects radio telescope data
- Data is divided into work chunks of 300 Kbytes
- User obtains client, which runs in backgrd

- Peer sets up TCP connection to central computer, downloads chunk
- Peer does FFT on chunk, uploads results, gets new chunk

Not peer to peer, but exploits resources at network edge

1. Overview of P2P

- overlay networks
- P2P applications
- worldwide computer vision

Worldwide Computer Vision

Alice's home computer:

- Working for biotech, matching gene sequences
- DSL connection downloading telescope data
- Contains encrypted fragments of thousands of non-Alice files
- Occasionally a fragment is read; it's part of a movie someone is watching in Paris
- Her laptop is off, but it's backing up others' files

- Alice's computer is moonlighting
- Payments come from biotech company, movie system and backup service

Your PC is only a component in the "big" computer

Worldwide Computer (2)

Anderson & Kubiatowicz:

Internet-scale OS

- Thin software layer running on each host & central coordinating system running on ISOS server complex
- allocating resources, coordinating currency transfer
- Supports data processing & online services

Challenges

- heterogeneous hosts
- security
- payments

Central server complex

- needed to ensure privacy of sensitive data
- ISOS server complex maintains databases of resource descriptions, usage policies, and task descriptions

2. Unstructured P2P File Sharing

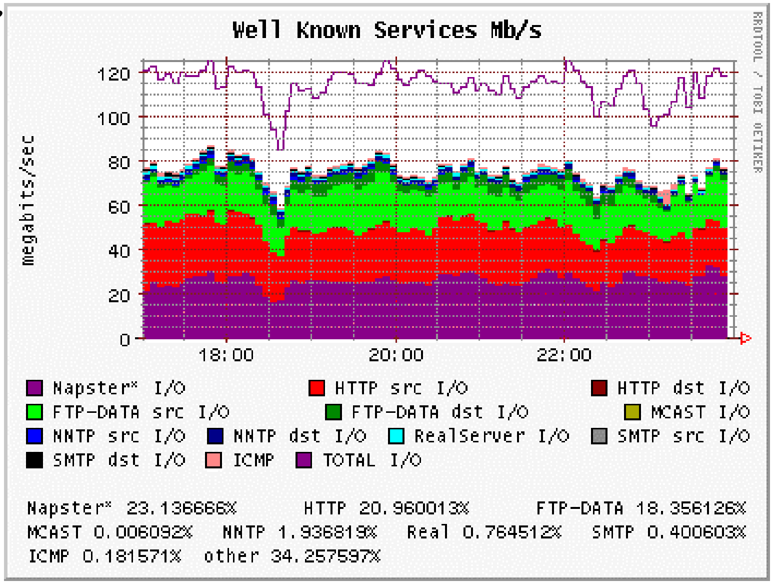
- Napster
- Gnutella
- KaZaA
- BitTorrent
- search theory
- dealing with flash crowds

Napster

- Paradigm shift
- not the first (c.f. probably Eternity, from Ross Anderson in Cambridge)
- but instructive for what it gets right, and
- also wrong…
- also had a political message…and economic and legal…

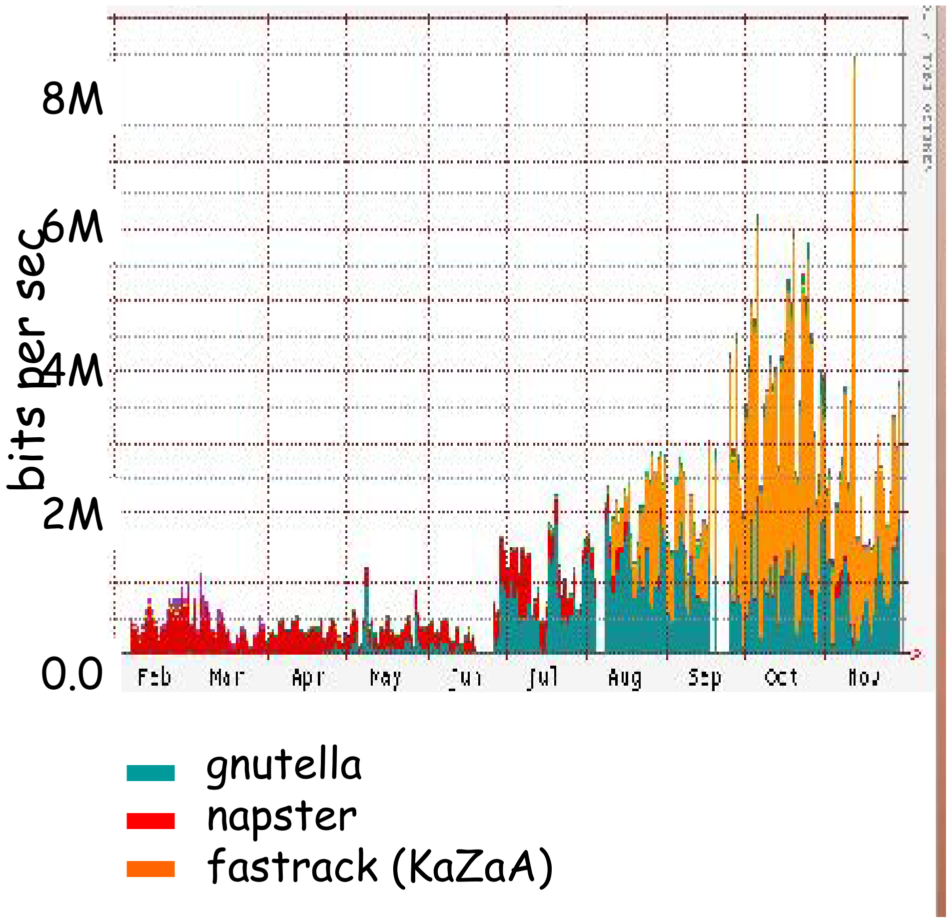
Napster

- 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
  - 2/01: US Circuit Court of Appeals: Napster knew users violating copyright laws
  - 7/01: # simultaneous online users: Napster 160K, Gnutella: 40K, Morpheus (KaZaA): 300K



Napster

- judge orders Napster to pull plug in July '01
- other file sharing apps take over!



Napster: how did it work

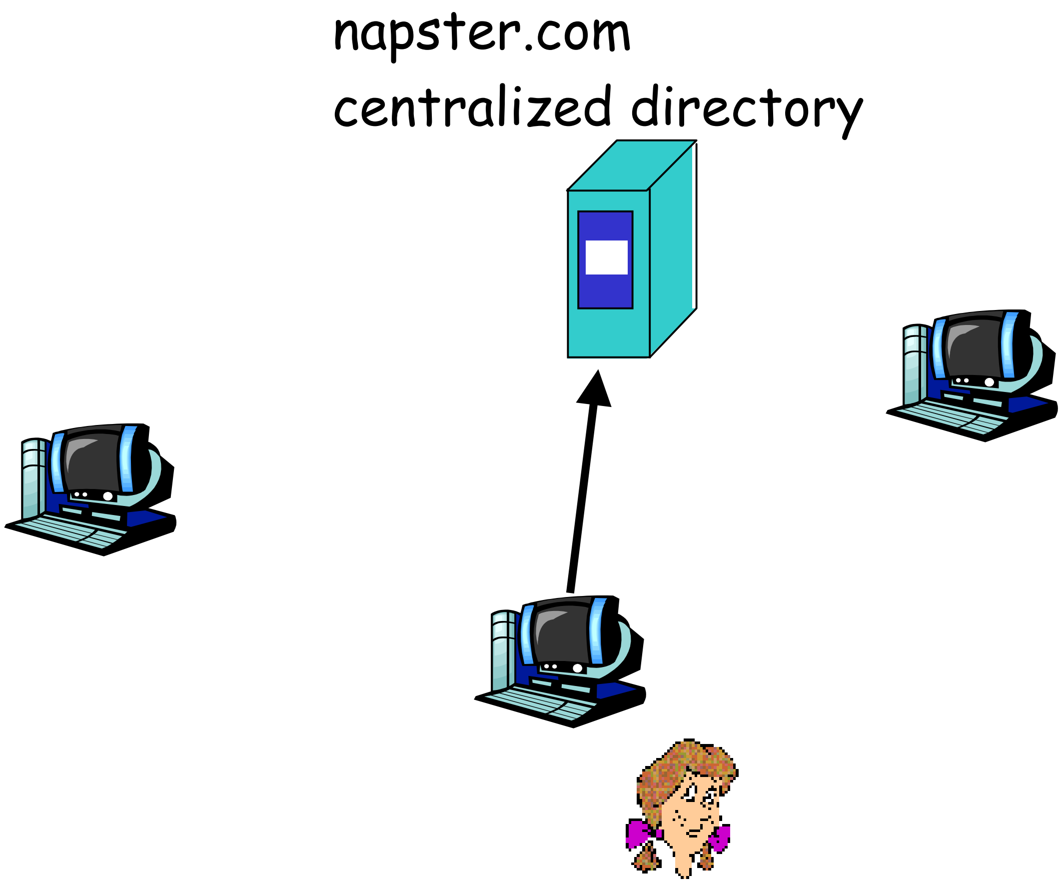
- Application-level, client-server protocol over point-to-point TCP
- Centralized directory server

Steps:

- connect to Napster server
- upload your list of files to server.
- give server keywords to search the full list with.
- select "best" of correct answers. (pings)

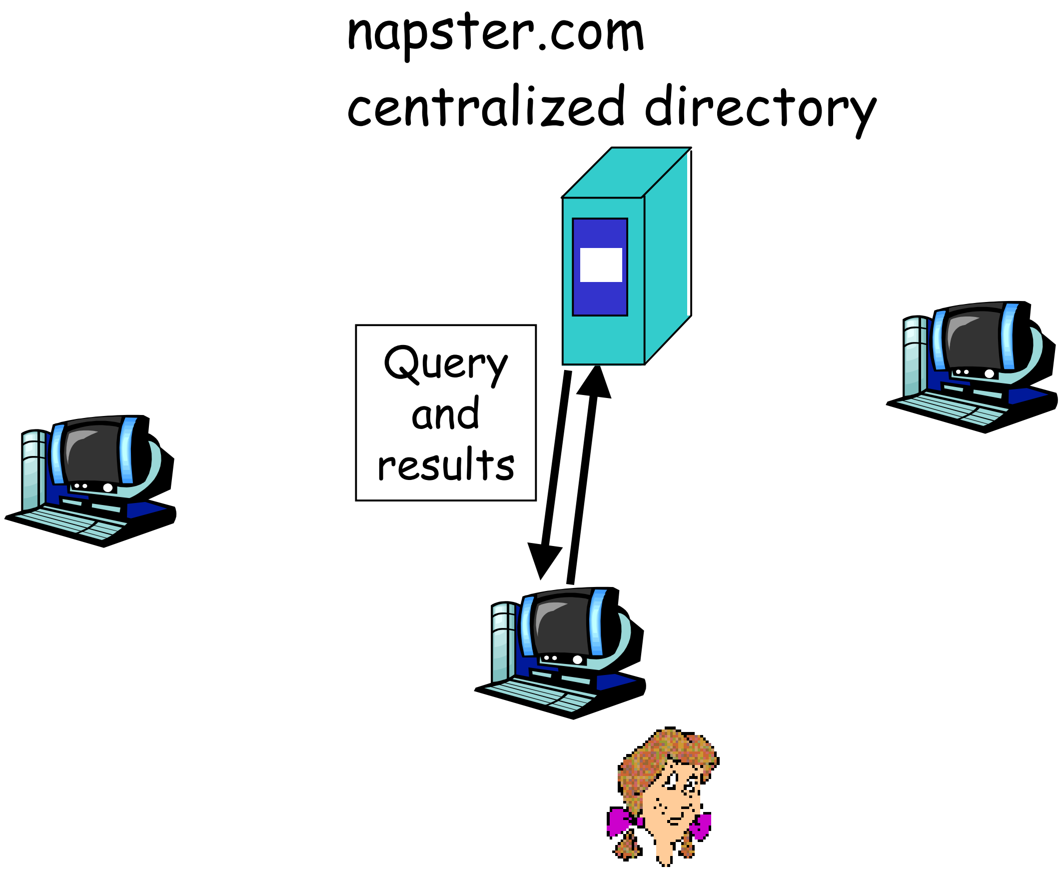
Napster

1. File list and IP address is uploaded



Napster

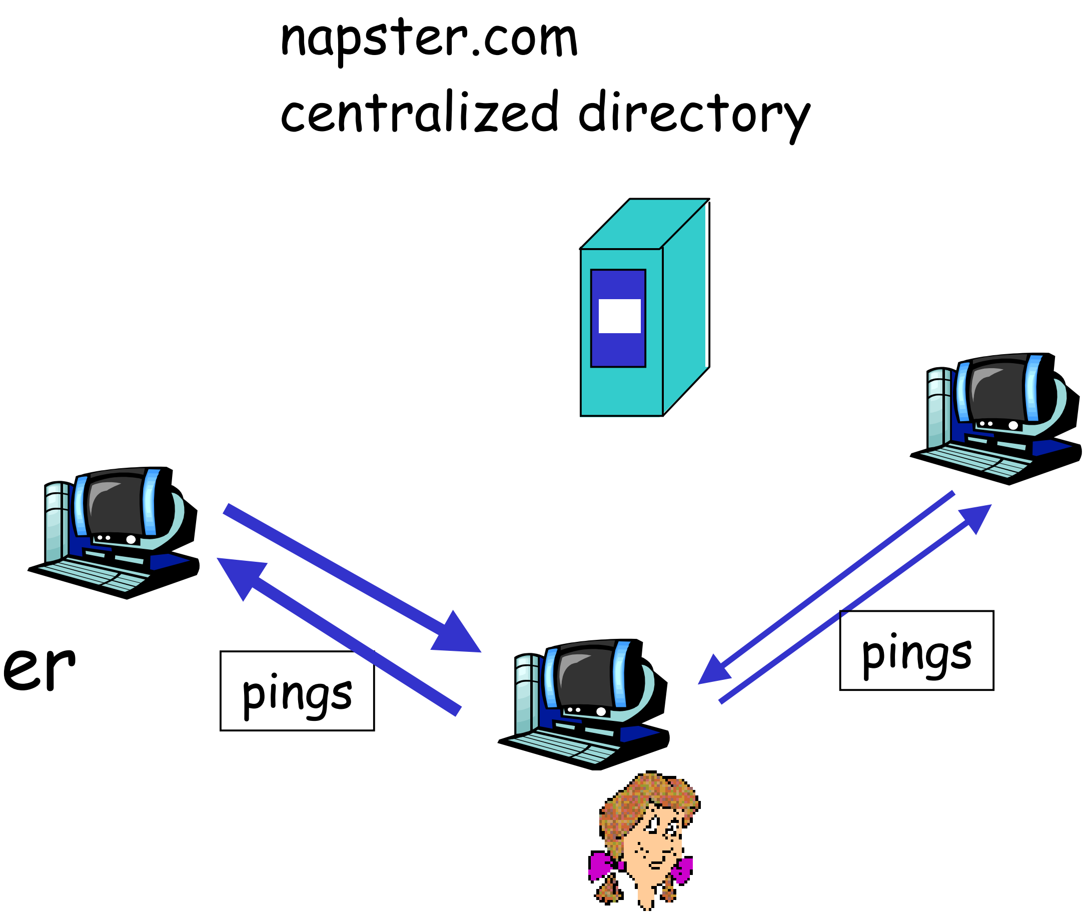
2. User requests search at server.



Napster

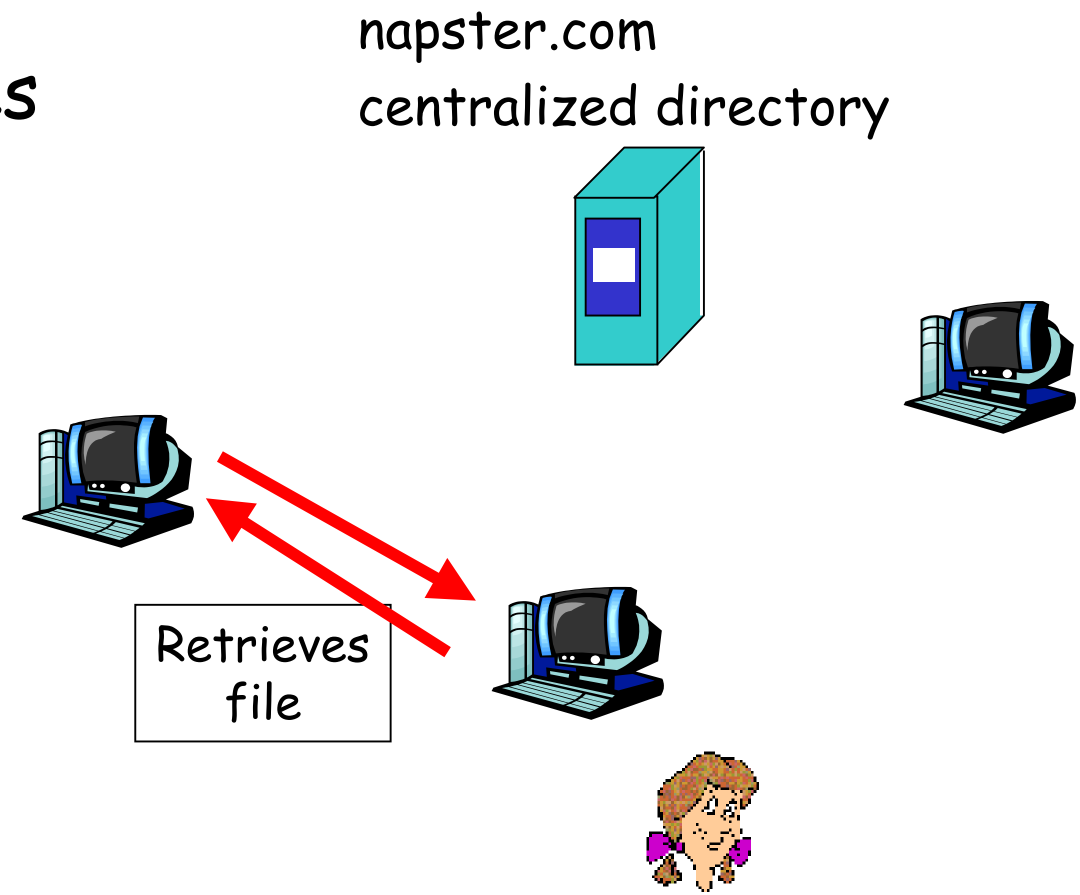
3. User pings hosts that apparently have data.

   Looks for ***best*** transfer rate.



Napster

4. User chooses server

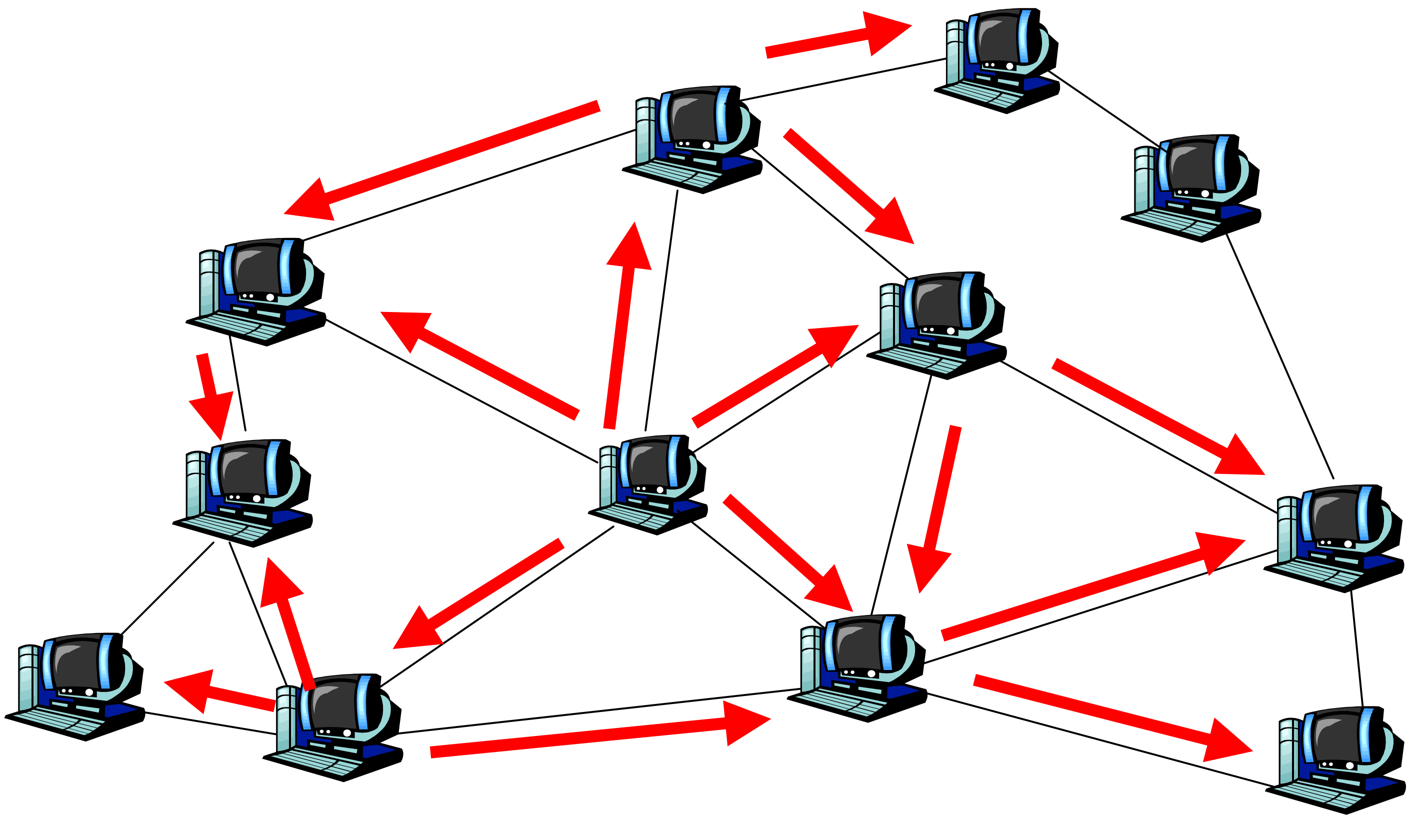


Napster's centralized server farm had difficult time keeping up with traffic

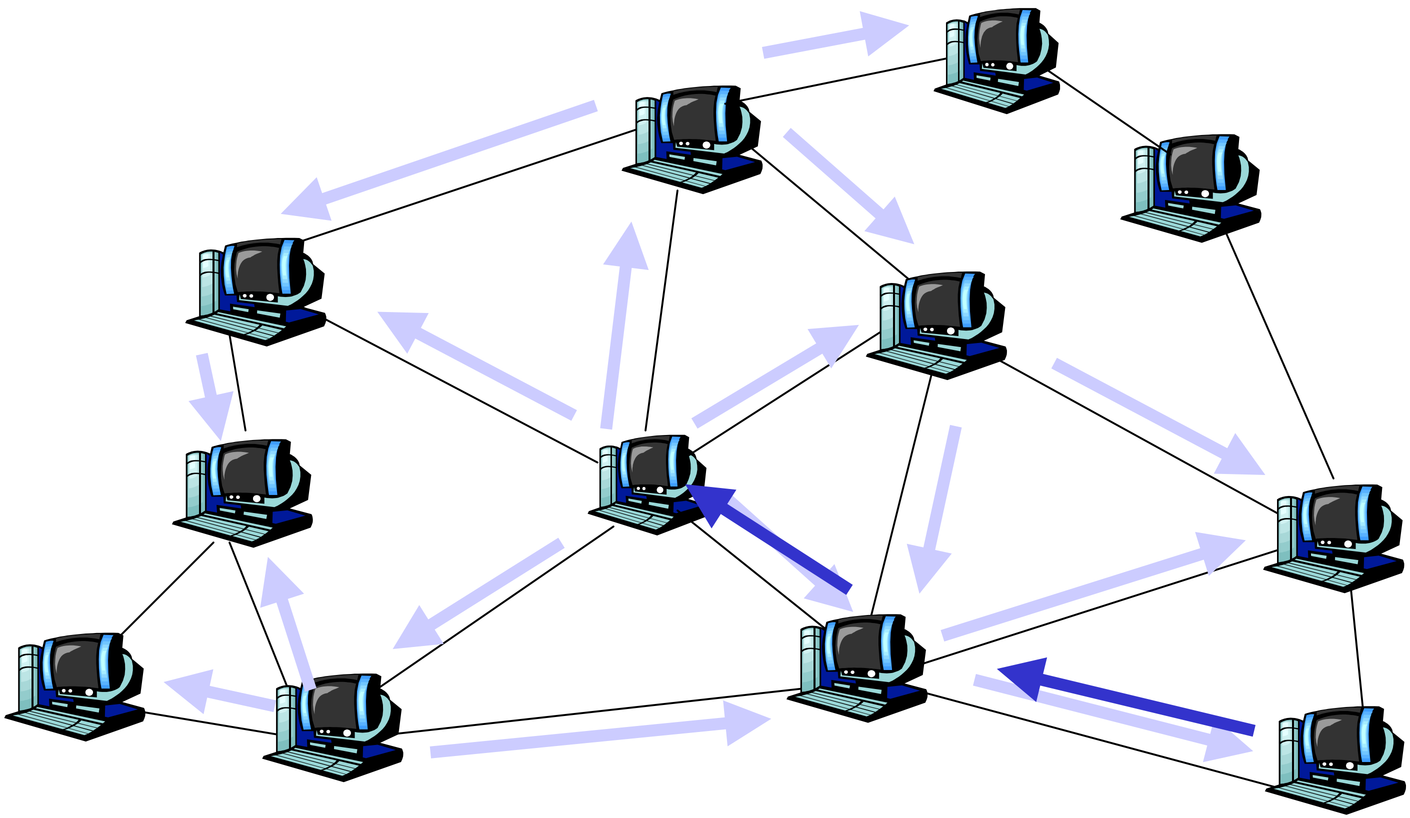
2. Unstructured P2P File Sharing

- Napster
- Gnutella
- KaZaA
- BitTorrent
- search theory
- dealing with flash crowds

Distributed Search/Flooding



Distributed Search/Flooding



Gnutella

- focus: decentralized method of searching for files
  - central directory server no longer the bottleneck
  - more difficult to "pull plug"
- each application instance serves to:
  - store selected files
  - route queries from and to its neighboring peers
  - respond to queries if file stored locally
  - serve files

Gnutella

- Gnutella history:
  - 3/14/00: release by AOL, almost immediately withdrawn
  - became open source
  - many iterations to fix poor initial design (poor design turned many people off)
- issues:
  - how much traffic does one query generate?
  - how many hosts can it support at once?
  - what is the latency associated with querying?
  - is there a bottleneck?

Gnutella: limited scope query

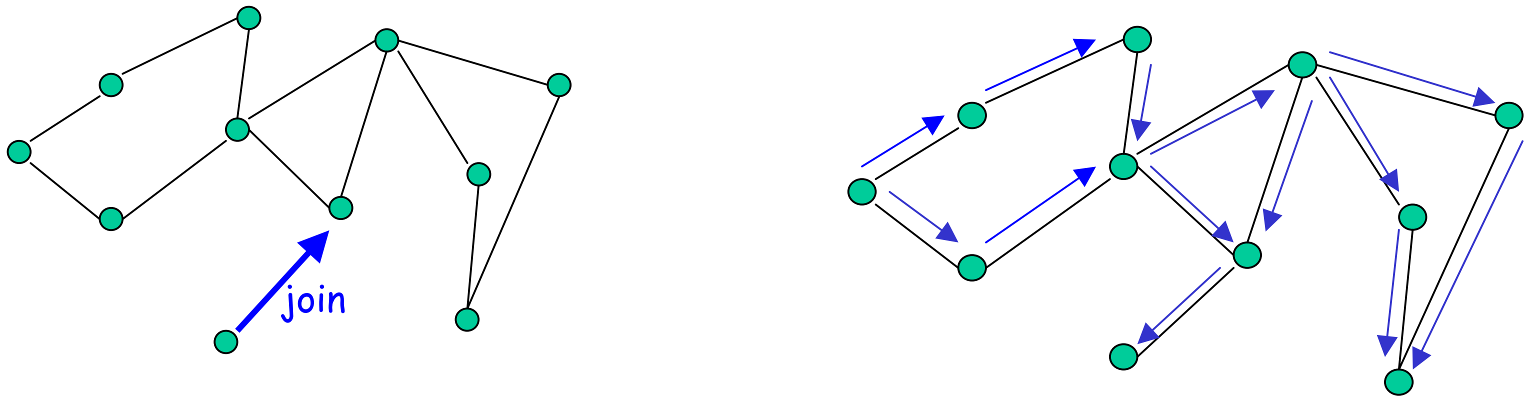
Searching by flooding:

- 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.
- reverse path forwarding for responses (not files)

Note: Play gnutella animation at:
http://www.limewire.com/index.jsp/p2p

Gnutella overlay management

- New node uses bootstrap node to get IP addresses of existing Gnutella nodes
- New node establishes neighboring relations by sending join messages



Gnutella in practice

- Gnutella traffic << KaZaA traffic
- 16-year-old daughter said "it stinks"
  - Couldn't find anything
  - Downloads wouldn't complete
- Fixes: do things KaZaA is doing: hierarchy, queue management, parallel download,...

Gnutella Discussion:

- researchers like it because it's open source
  - but is it truly representative?
- architectural lessons learned?
- More details in Kurose and Ross, 3rd edition

2. Unstructured P2P File Sharing

- Napster
- Gnutella
- KaZaA
- BitTorrent
- search theory
- dealing with flash crowds

KaZaA: The service

- more than 3 million up peers sharing over 3,000 terabytes of content
- more popular than Napster ever was
- more than 50% of Internet traffic ?
- MP3s & entire albums, videos, games
- optional parallel downloading of files
- automatically switches to new download server when current server becomes unavailable
- provides estimated download times

KaZaA: The service (2)

- User can configure max number of simultaneous uploads and max number of simultaneous downloads
- queue management at server and client
  - Frequent uploaders can get priority in server queue
- Keyword search
  - User can configure "up to x" responses to keywords
- Responses to keyword queries come in waves; stops when x responses are found
- From user's perspective, service resembles Google, but provides links to MP3s and videos rather than Web pages

KaZaA: Technology

Software

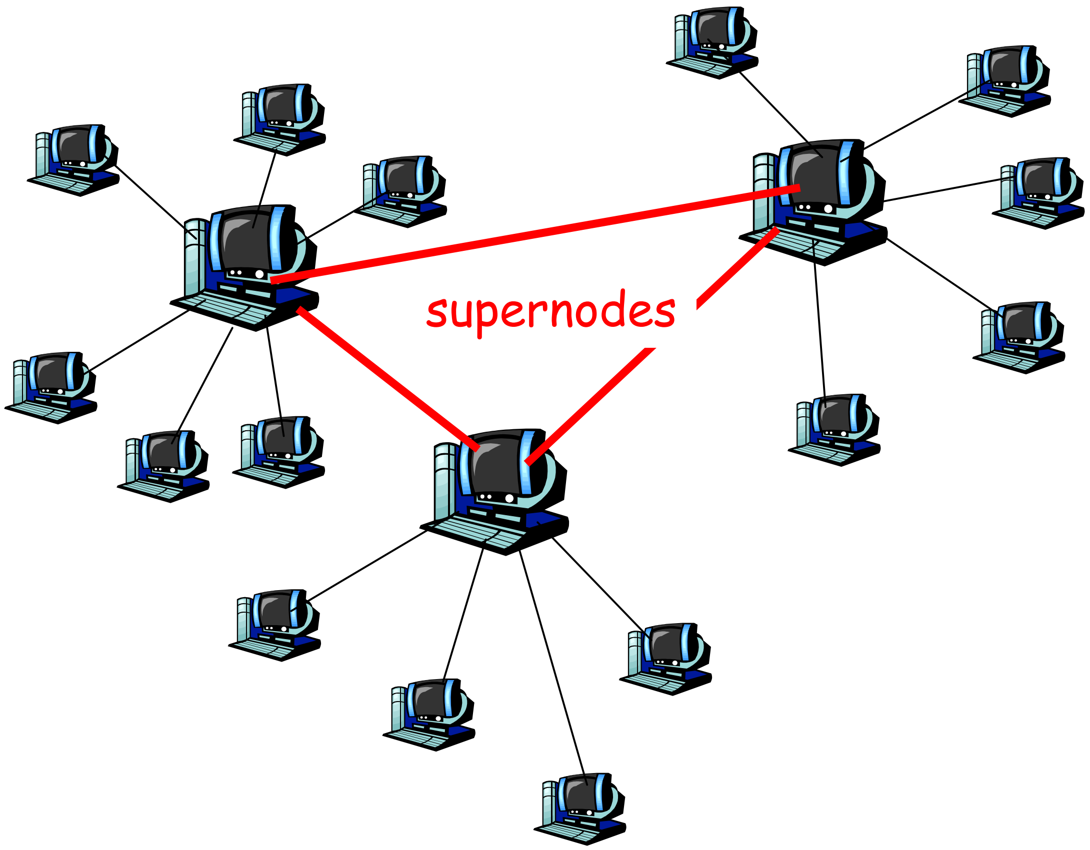
- Proprietary
- control data encrypted
- Everything in HTTP request and response messages

Architecture

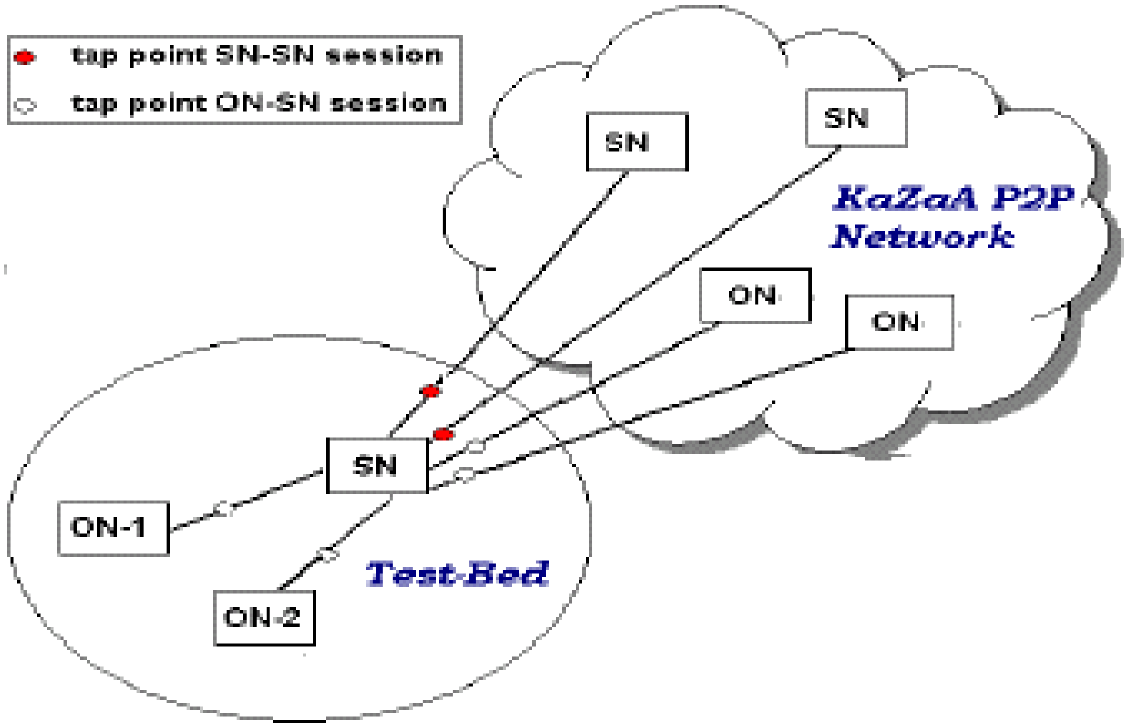
- hierarchical
- cross between Napster and Gnutella

KaZaA: Architecture

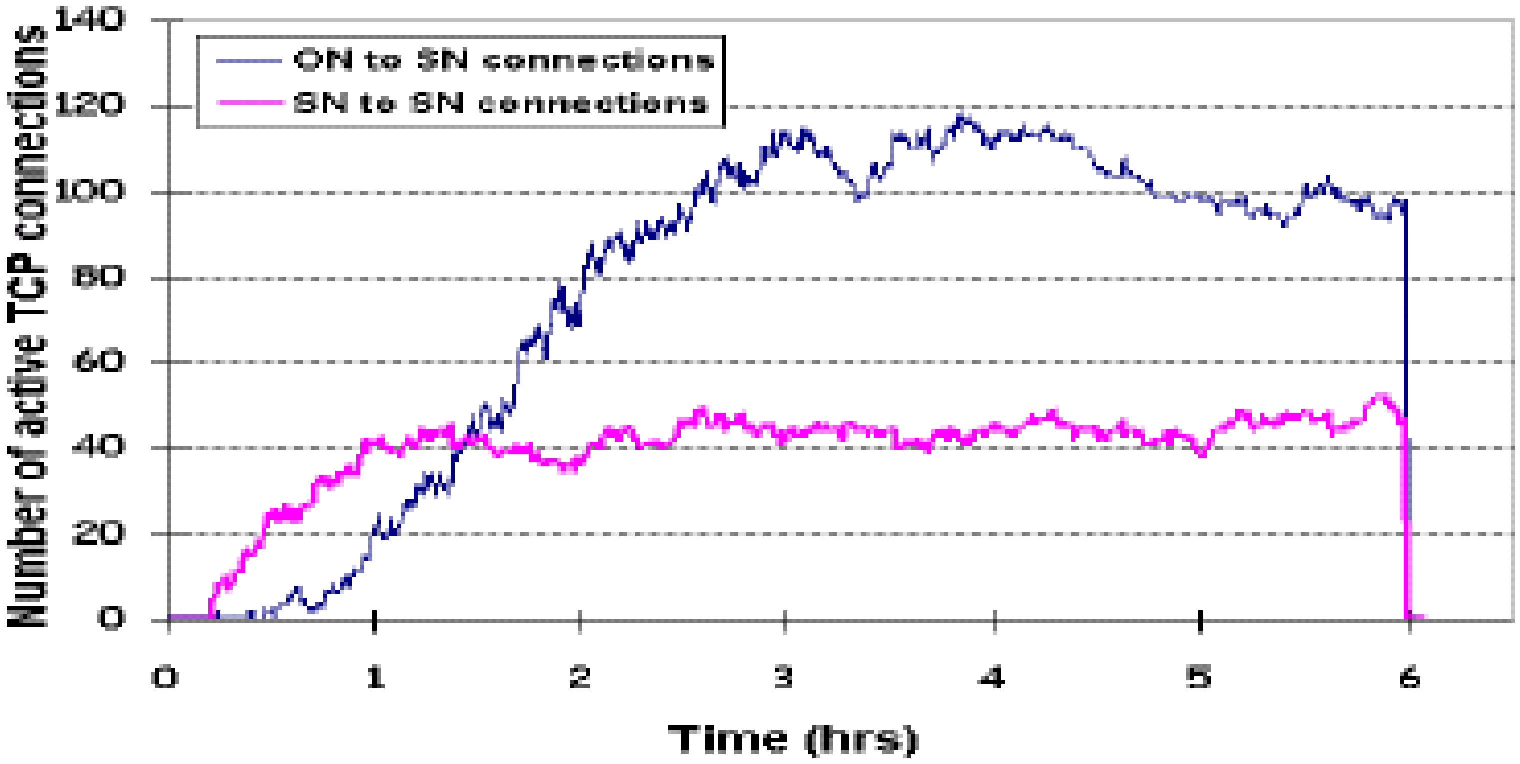
- Each peer is either a supernode or is assigned to a supernode
  - 56 min avg connect
  - Each SN has about 100-150 children
  - Roughly 30,000 SNs
- Each supernode has TCP connections with 30-50 supernodes
  - 0.1% connectivity
  - 23 min avg connect



Measurement study



Evolution of connections at SN



KaZaA: Architecture (2)

- Nodes that have more connection bandwidth and are more available are designated as supernodes
- Each supernode acts as a mini-Napster hub, tracking the content and IP addresses of its descendants
- Does a KaZaA SN track only the content of its children, or does it also track the content under its neighboring SNs?
  - Testing indicates only children.

KaZaA metadata

- When ON connects to SN, it uploads its metadata.
- For each file:
  - File name
  - File size
  - Content Hash
  - File descriptors: used for keyword matches during query
- Content Hash:
  - When peer A selects file at peer B, peer A sends ContentHash in HTTP request
  - If download for a specific file fails (partially completes), ContentHash is used to search for new copy of file.

KaZaA: Overlay maintenance

- List of potential supernodes included within software download
- New peer goes through list until it finds operational supernode
  - Connects, obtains more up-to-date list, with 200 entries
  - Nodes in list are "close" to ON.
  - Node then pings 5 nodes on list and connects with the one
- If supernode goes down, node obtains updated list and chooses new supernode

KaZaA Queries

- Node first sends query to supernode
  - Supernode responds with matches
  - If x matches found, done.
- Otherwise, supernode forwards query to subset of supernodes
  - If total of x matches found, done.
- Otherwise, query further forwarded
  - Probably by original supernode rather than recursively

Kazaa-lite

- Hacked version of the KaZaA client
- No spyware; no pop-up windows
- Everyone is rated as a priority user
- Supernode hopping
  - After receiving replies from SN, ON often connects to new SN and re-sends query
  - SN does not cache hopped-out ON's metadata

Parallel Downloading; Recovery

- If file is found in multiple nodes, user can select parallel downloading
  - Identical copies identified by ContentHash
- HTTP byte-range header used to request different portions of the file from different nodes
- Automatic recovery when server peer stops sending file
  - ContentHash

KaZaA Corporate Structure

- Software developed by Estonians
- FastTrack originally incorporated in Amsterdam
- FastTrack also deploys KaZaA service
- FastTrack licenses software to Music City (Morpheus) and Grokster
- Later, FastTrack terminates license, leaves only KaZaA with killer service
- Summer 2001, Sharman networks, founded in Vanuatu (small island in Pacific), acquires FastTrack
  - Board of directors, investors: secret
- Employees spread around, hard to locate

Lessons learned from KaZaA

KaZaA provides powerful file search and transfer service without server infrastructure

- Exploit heterogeneity
- Provide automatic recovery for interrupted downloads
- Powerful, intuitive user interface

Copyright infringement

- International cat-and-mouse game
- With distributed, serverless architecture, can the plug be pulled?
- Prosecute users?
- Launch DoS attack on supernodes?
- Pollute?

Measurement studies by Gribble et al

- 2002 U. Wash campus study
- P2P: 43%; Web: 14%
- Kazaa objects fetched at most once per client
- Popularity distribution deviates substantially from Zipf distribution
  - Flat for 100 most popular objects
- Popularity of objects is short.

KaZaA users are patient

- Small objects (<10MB): 30% take more than hour to download
- Large objects (>100MB): 50% more than 1 day
- Kazaa is a batch-mode system, downloads done in background

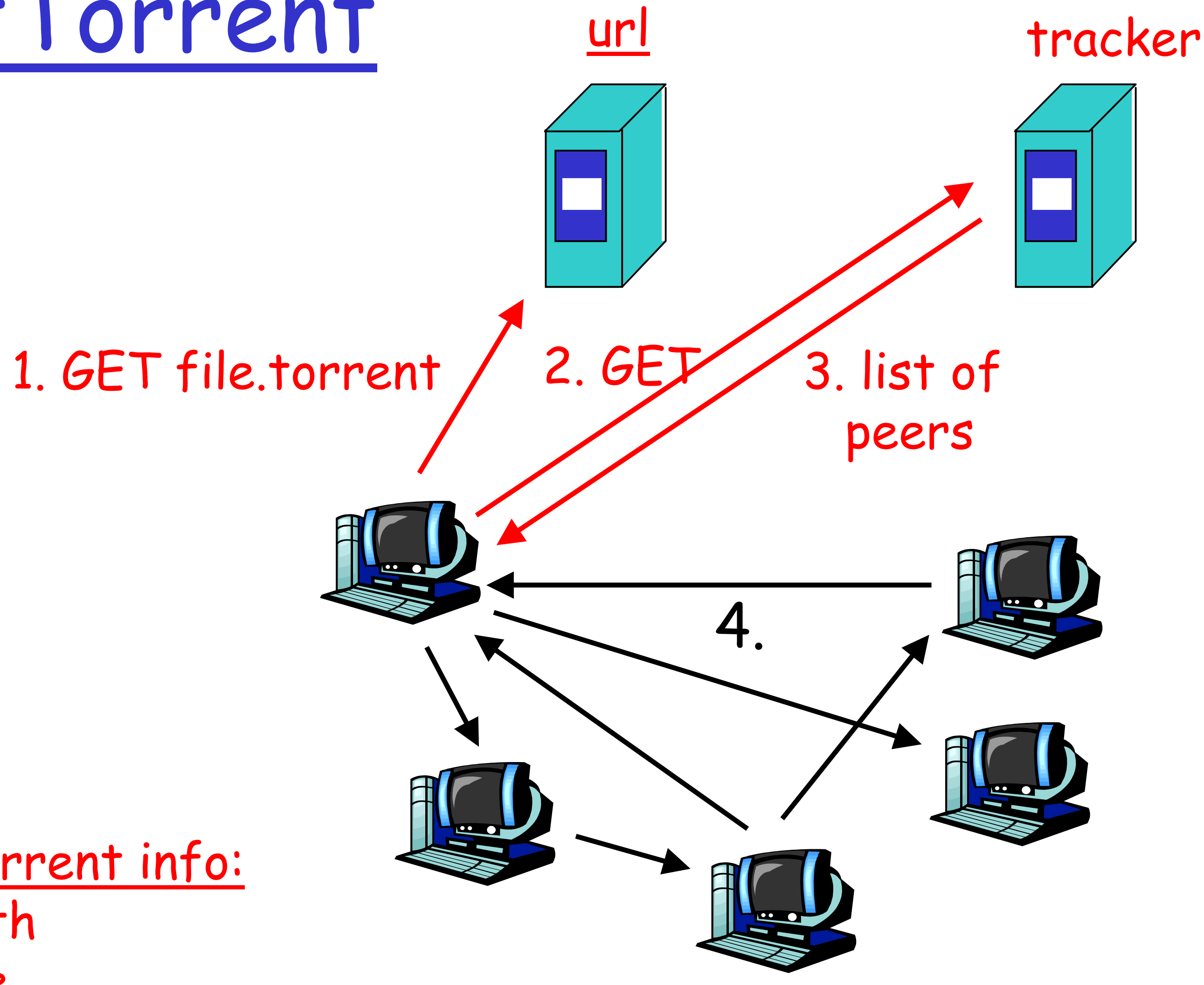
Pollution in P2P

- Record labels hire "polluting companies" to put bogus versions of popular songs in file sharing systems
- Polluting company maintains hundreds of nodes with high bandwidth connections
- User A downloads polluted file
- User B may download polluted file before A removes it
- How extensive is pollution today?
- Anti-pollution mechanisms?

2. Unstructured P2P File Sharing

- Napster
- Gnutella
- KaZaA
- BitTorrent
- search theory
- dealing with flash crowds

BitTorrent



file.torrent info:
- length
- name
- hash
- url of tracker

BitTorrent: Pieces

- File is broken into pieces
  - Typically piece is 256 KBytes
  - Upload pieces while downloading pieces
- Piece selection
  - Select rarest piece
  - Except at beginning, select random pieces
- Tit-for-tat
  - Bit-torrent uploads to at most four peers
  - Among the uploaders, upload to the four that are downloading to you at the highest rates
  - A little randomness too, for probing

NATs

- nemesis for P2P
- Peer behind NAT can't be a TCP server
- Partial solution: reverse call
  - Suppose A wants to download from B, B behind NAT
  - Suppose A and B have each maintain TCP connection to server C (not behind NAT)
  - A can then ask B, through C, to set up a TCP connection from B to A.
  - A can then send query over this TCP connection, and B can return the file
- What if both A and B are behind NATs?

2. Unstructured P2P File Sharing

- Napster
- Gnutella
- KaZaA
- search theory
- dealing with flash crowds

Modeling Unstructured P2P Networks

- In comparison to DHT-based searches, unstructured searches are
  - simple to build
  - simple to understand algorithmically
- Little concrete is known about their performance
- Q: what is the expected overhead of a search?
- Q: how does caching pointers help?

Replication

- Scenario
  - Nodes cache copies (or pointers to) content
    - object info can be "pushed" from nodes that have copies
    - more copies leads to shorter searches
  - Caches have limited size: can't hold everything
  - Objects have different popularities: different content requested at different rates
- Q: How should the cache be shared among the different content?
  - Favor items under heavy demand too much then lightly demanded items will drive up search costs
  - Favor a more "flat" caching (i.e., independent of popularity), then frequent searches for heavily-requested items will drive up costs
- Is there an optimal strategy?

Model

- Given
  - m objects, n nodes, each node can hold c objects, total system capacity = cn
  - q_i is the request rate for the i^{th} object, $q_1 \geq q_2 \geq \ldots \geq q_m$
  - p_i is the fraction of total system capacity used to store object i, $\sum p_i = 1$
- Then
  - Expected length of search for object i = K / p_i for some constant K
    - note: assumes search selects node w/ replacement, search stops as soon as object found
  - Network "bandwidth" used to search for all objects: $B = \sum q_i K / p_i$
- Goal: Find allocation for $\{p_i\}$ (as a function of $\{q_i\}$) to minimize B
- Goal 2: Find distributed method to implement this allocation of $\{p_i\}$

Some possible choices for $\{p_i\}$

- Consider some typical allocations used in practice
  - Uniform: $p_1 = p_2 = \ldots = p_m = 1/m$
    - easy to implement: whoever creates the object sends out cn/m copies
  - Proportional: $p_i = a\, q_i$ where $a = 1/\sum q_i$ is a normalization constant
    - also easy to implement: keep the received copy cached
- What is $B = \sum q_i K / p_i$ for these two policies?
  - Uniform: $B = \sum q_i K / (1/m) = Km/a$
  - Proportional: $B = \sum q_i K / (a\, q_i) = Km/a$
- B is the same for the Proportional and Uniform policies!

In between Proportional and Uniform

- Uniform: $p_i / p_{i+1} = 1$, Proportional: $p_i / p_{i+1} = q_i / q_{i+1} \geq 1$
- In between: $1 \leq p_i / p_{i+1} \leq q_i / q_{i+1}$
- Claim: any in-between allocation has lower B than B for Uniform / Proportional
- Proof: Omitted here
- Consider Square-Root allocation: $p_i = \mathrm{sqrt}(q_i) / \sum \mathrm{sqrt}(q_i)$
- Thm: Square-Root is optimal
- Proof (sketch):
  - Noting $p_m = 1 - (p_1 + \ldots + p_{m-1})$
  - write $B = F(p_1, \ldots, p_{m-1}) = \sum^{m-1} q_i/p_i + q_m/(1 - \sum^{m-1} p_i)$
  - Solving $dF/dp_i = 0$ gives $p_i = p_m \,\mathrm{sqrt}(q_i/q_m)$

Distributed Method for Square-Root Allocation

- Assumption: each copy in the cache disappears from the cache at some rate independent of the object cached (e.g., object lifetime is i.i.d.)
- Algorithm Sqrt-Cache: cache a copy of object i (once found) at each node visited while searching for object i
- Claim Algorithm implements Square-Root Allocation

Proof of Claim

- Sketch of Proof of Correctness:
  - Let $f_i(t)$ be fraction of locations holding object i @ time t
  - $p_i = \lim_{t \to \infty} f_i(t)$
  - At time t, using Sqrt-Cache, object i populates cache at avg rate $r_i = q_i / f_i(t)$
  - When $f_i(t) / f_j(t) < sqrt(q_i) / sqrt(q_j)$, then
    - $r_i(t) / r_j(t) = q_i f_j(t) / q_j f_i(t) > sqrt(q_i) / sqrt(q_j)$
    - hence, ratio $f_i(t) / f_j(t)$ will increase
  - When $f_i(t) / f_j(t) > sqrt(q_i) / sqrt(q_j)$, then
    - $r_i(t) / r_j(t) = q_i f_j(t) / q_j f_i(t) < sqrt(q_i) / sqrt(q_j)$
    - hence, ratio $f_i(t) / f_j(t)$ will decrease
  - Steady state is therefore when $f_i(t) / f_j(t) = sqrt(q_i) / sqrt(q_j)$,

2. Unstructured P2P File Sharing

- Napster
- Gnutella
- KaZaA
- search theory
- dealing with flash crowds

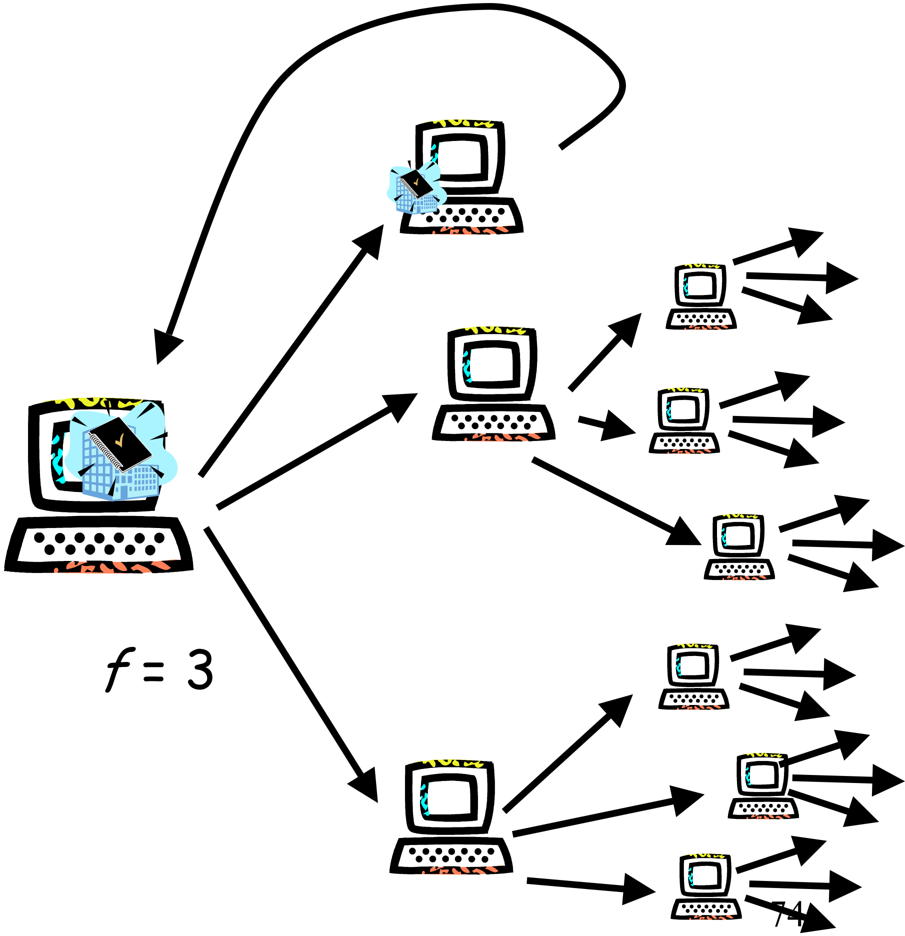
Flash Crowd

- Def: A sudden, unanticipated growth in demand of a particular object
- Assumption: content was previously "cold" and hence an insufficient number of copies is loaded into the cache
- How long will it take (on average) for a user to locate the content of interest?
- How many messages can a node expect to receive due to other nodes' searches?

Generic Search Protocol

Randomized TTL-scoped search

- Initiator sends queries to f randomly chosen neighbors
- Node receiving query
  - with object: forwards object directly (via IP) to the initiator
  - w/o object TTL not exceeded: forwards query to f neighbors, else does nothing
  - w/o object and TTL exceeded: do nothing
- If object not found, increase TTL and try again (to some maximum TTL)
- Note: dumb protocol, nodes do not supress repeat queries



Analysis of the Search Protocol

- Modeling assumptions:
  - Neighbor overlay is fully-connected
    - queries are "memoryless" – if a node is queried multiple times, it acts each time as if it's the first time (and a node may even query itself)
    - Accuracy of analysis verified via comparison to simulation on neighbor overlays that are sparsely connected
  - Protocol is round-based: query received by participant in round i is forwarded to f neighbors in round i+1
  - Time searchers start their searches: will evaluate 2 extremes
    - sequential: one user searches at a time
    - simultaneous: all users search simultaneously

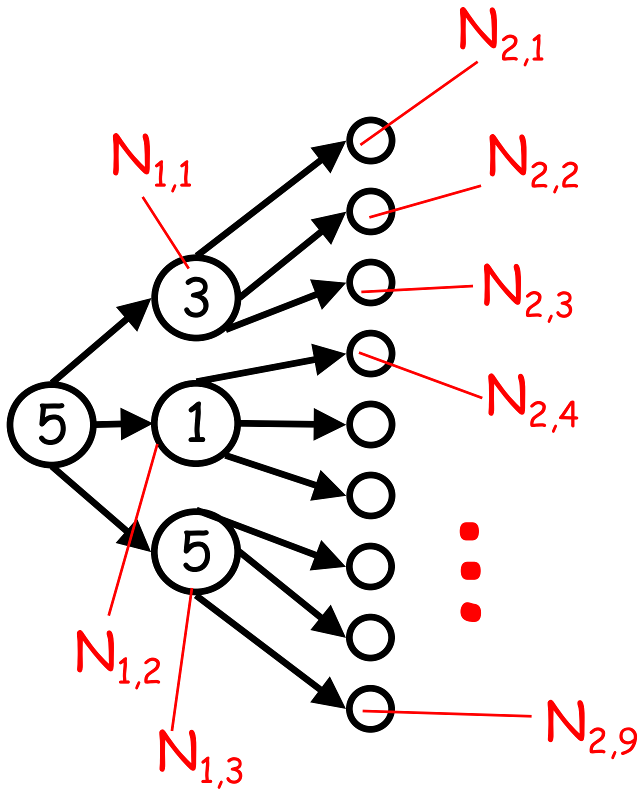
Search Model: Preliminaries

- Parameters
  - N = # nodes in the overlay (fully connected)
  - H = # nodes that have a copy of the desired object (varies w/ time)
- Performance Measures
  - R = # rounds needed to locate the object
  - T = # query transmissions
- p = P(Randomly chosen node does not have object) = 1-(H/N)
- Recall: f = # neighbors each node forwards query to

- $P(R > i) = p^{\wedge}(f+f^2+f^3+\ldots+f^i) = p^{\wedge}((f^i+1-f)/(f-1))$
- $E[R] = \sum_{i \geq 0} P(R > i)$

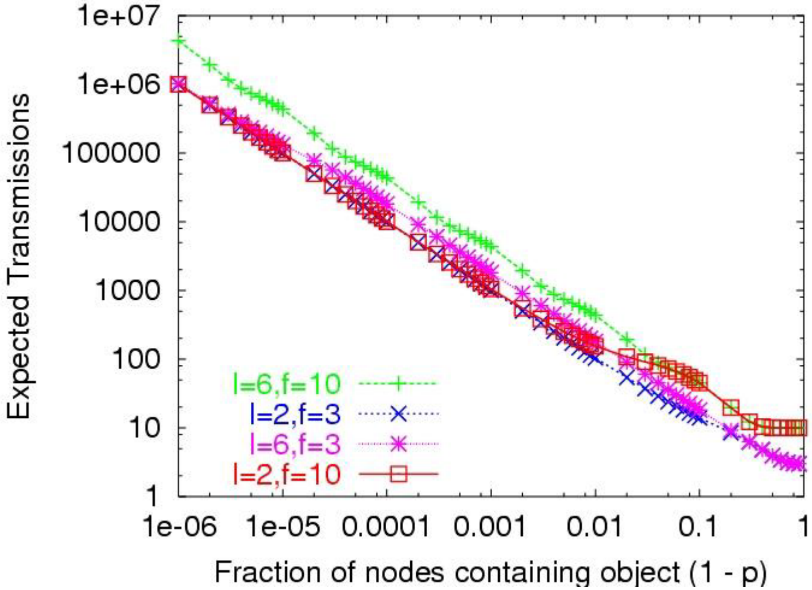
Search Model cont'd

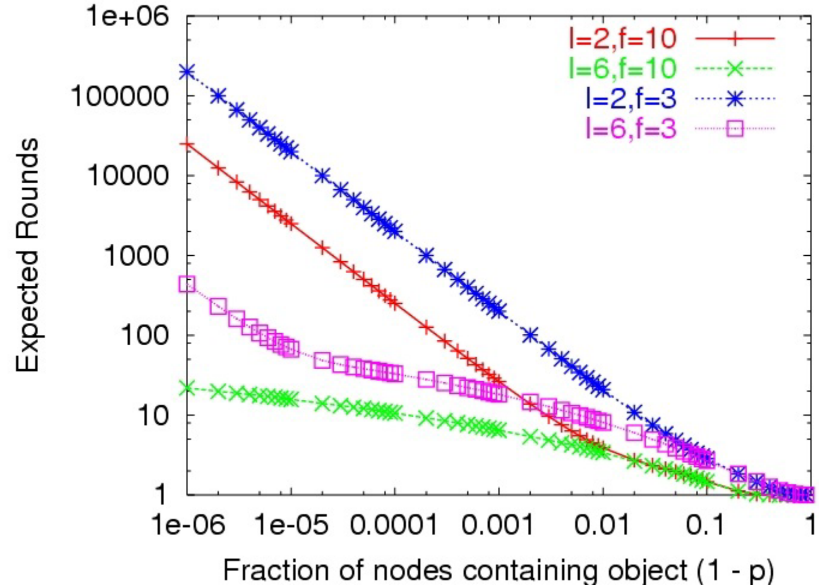
- To compute E[T]:
  - Create a schedule: Each node determines in advance who to query if a query is necessary
  - $N_{i,j}$ is the jth node at depth i in the schedule
  - $X_{i,j}$ = 1 if the query scheduled at $N_{i,j}$ is executed and is 0 otherwise
  - $X_{i,j}$= 1 if and only if both
    - $X_{i',j'}$=1 for the i-1 entries $N_{i',j'}$ along the path from $N_{1,1}$ to $N_{i,j}$
    - $N_{i,j}$ does not have a copy of the object



  - $P(X_{i,j}=1) = p^{i-1}$
  - $E[T] = \sum_{i,j} P(X_{i,j}=1) = \sum_{i} p\hat{}(0+f+f^2+\ldots+f^{i-1}) = \sum_{i} p\hat{}((f^i-1)/(f-1))$

Single-user search results



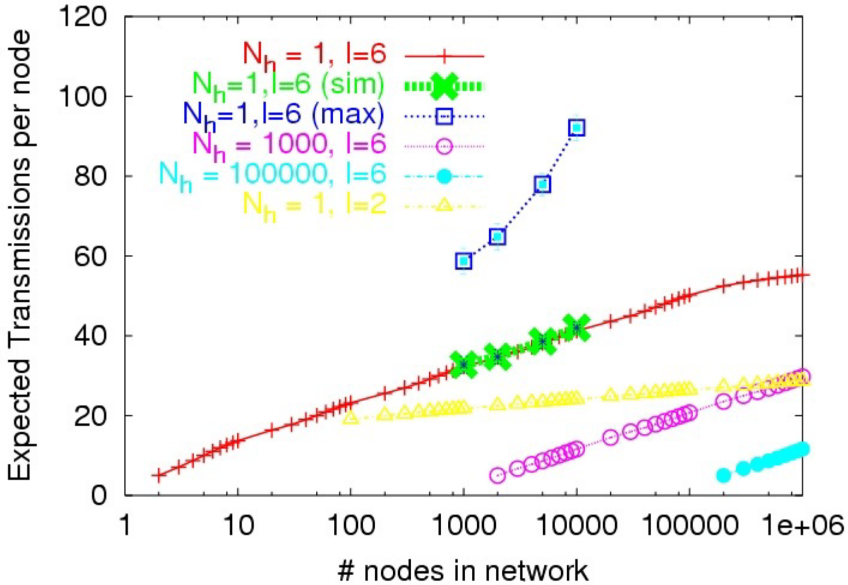


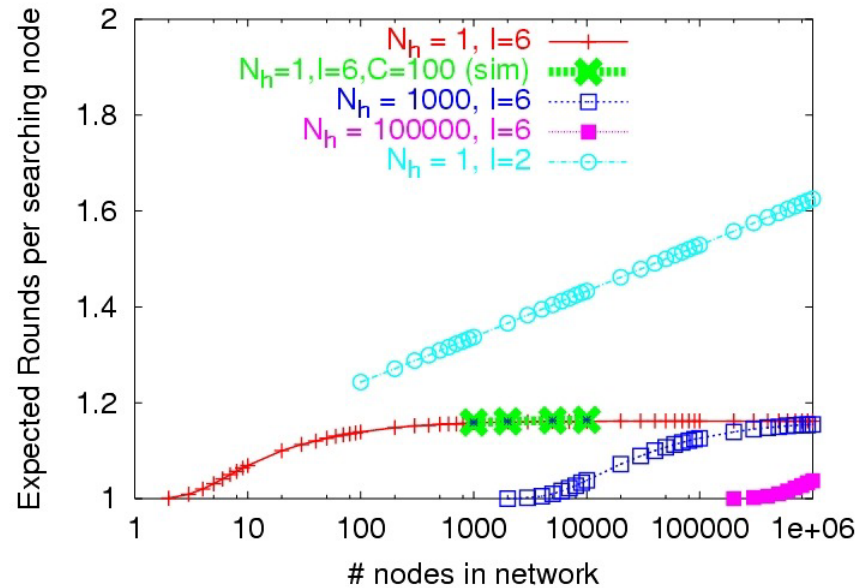
- Search cost inversely proportional to fraction of nodes w/ object
- Varying f and I (max TTL) has more of an affect on rounds than on transmissions

Analyzing undirected searches during a flash crowd

- Scenario: A large majority of users suddenly want the same object
- Numerous independent searches for the same object are initiated throughout the network
  - Nodes cannot suppress one user's search for an object with the other. Each search has different location where object should be delivered
- What is the cost of using an unstructured search protocol?

One-after-the-other Searches

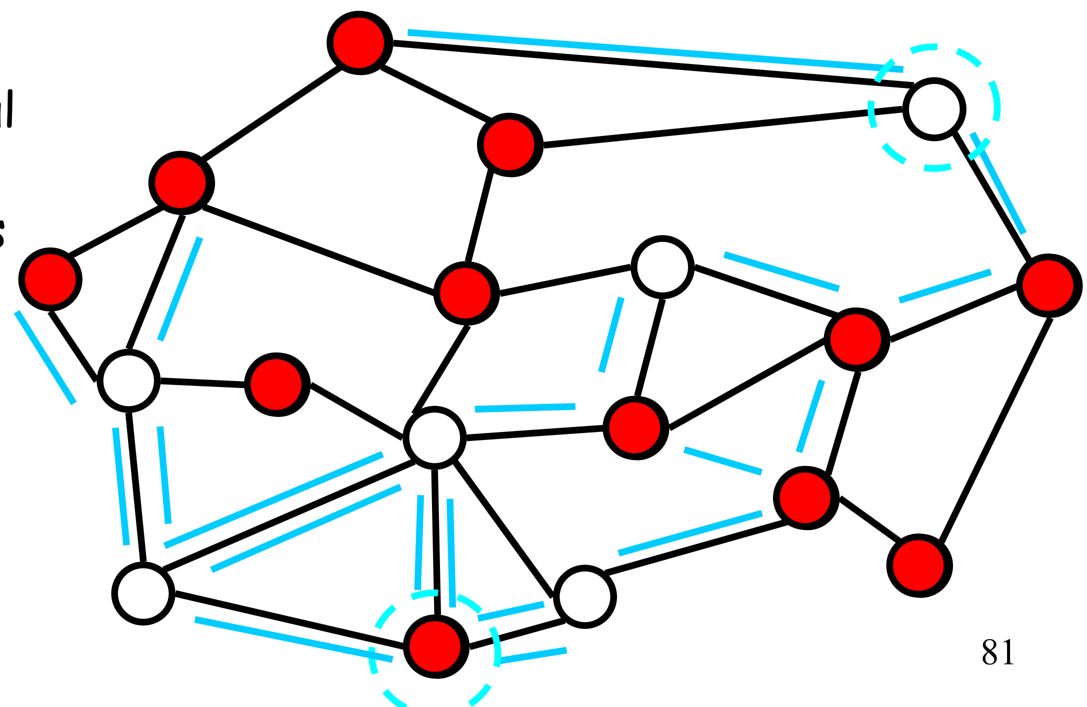




- N_h = # nodes that initially have object, I = max TTL
- Sequential searches,f = 10, terminates when all nodes have object
- Analytical Results (confirmed with simulation):
  - Expected transmissions sent and received per node is small (max is manageable)
  - Expected # of rounds small (unless max TTL kept small)
  - Simulation results use overlay graphs where # of neighbors bounded by 100: Note error using full connectivity is negligible

Flash Crowd Scalability: Intuitive Explanation

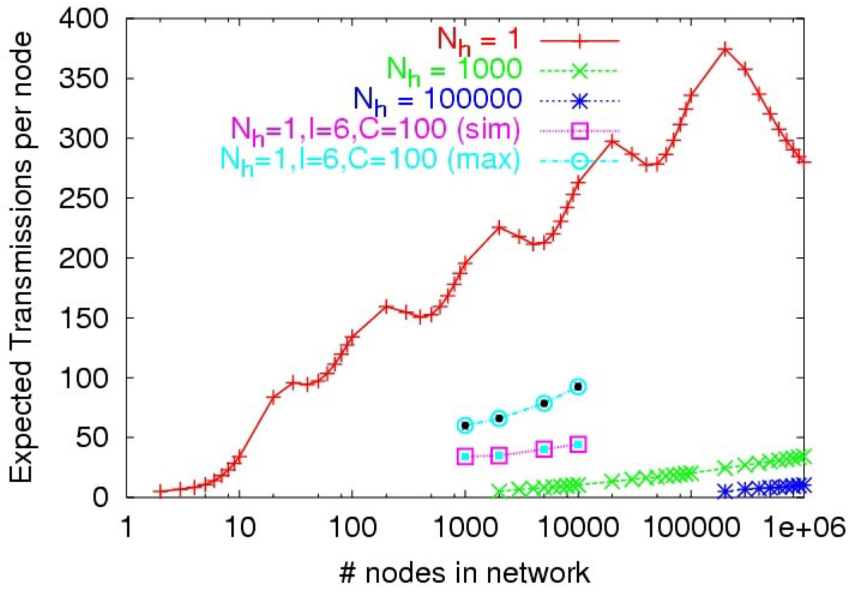
- Gnutella scales poorly when different users search for different objects: high transmission overhead
- Q: Why will expanding-ring TTL search achieve better scalability?
- A:
  - Popular objects propagate through overlay via successful searches
  - Subsequent searches often succeed with smaller TTL: require less overhead

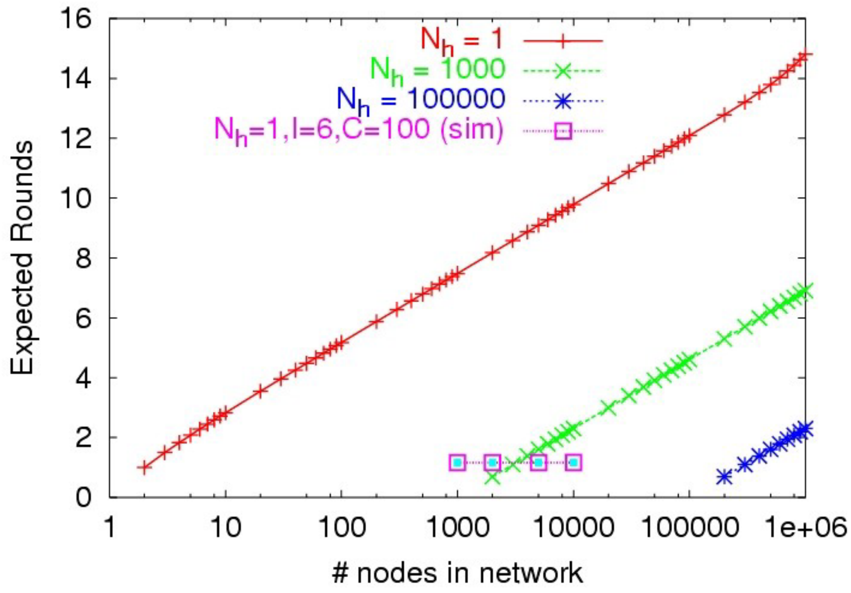


Simultaneous Searches

- Model: Start measuring at a point in time where N_h have copies and N_d nodes have been actively searching for a “long time”
- Compute upper bound on expected # transmissions and rounds
- Details omitted here…

Simultaneous Search Results





- Simulation results show upper bounds to be extremely conservative (using branching process model of search starts)
- Conclusion (conservative) :
  - less than 400 transmissions on average received and sent per node to handle delivery to millions of participants
  - less than 15 query rounds on average

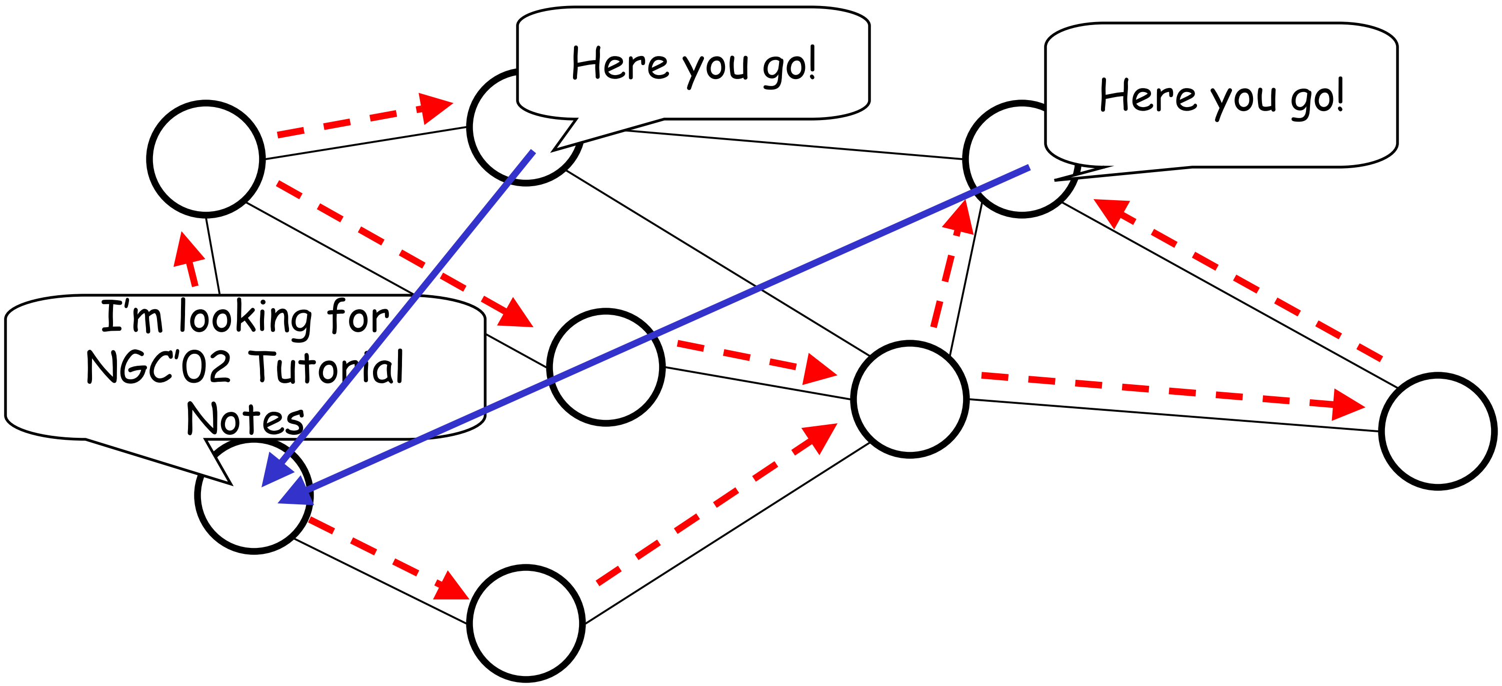
Simultaneous Search Intuition

- Let $h(t)$ be the number of nodes that have the object after the t^{th} round where d of N nodes are searching
- Each searching node contacts s nodes on average per round
- Approximation: $h(t) = h(t-1) + (d - h(t-1))\ s * \quad h(t-1)/N$, $h(0) > 0$
- Even when $h(t)/N$ is small, *some node* has high likelihood of finding object
- $h(t)$ grows quickly even when small when many users search simultaneously

3. Structured P2P: DHT Approaches

- DHT service and issues
- CARP
- Consistent Hashing
- Chord
- CAN
- Pastry/Tapestry
- Hierarchical lookup services
- Topology-centric lookup service

Challenge: Locating Content



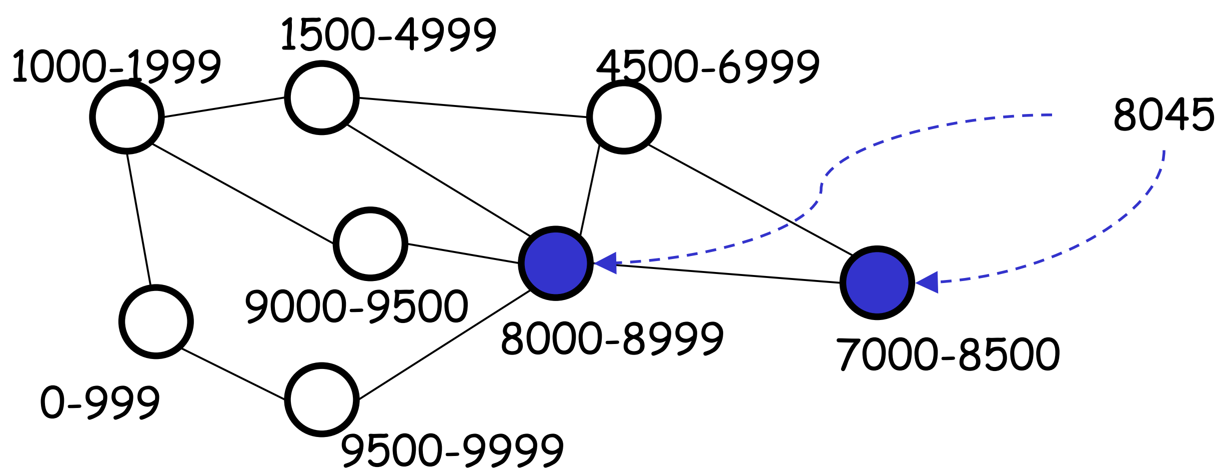
- Simplest strategy: expanding ring search
- If K of N nodes have copy, expected search cost *at least* N/K, i.e., O(N)
- Need many cached copies to keep search overhead small

Directed Searches

- Idea:
  - assign particular nodes to hold particular content (or pointers to it, like an information booth)
  - when a node wants that content, go to the node that is supposed to have or know about it
- Challenges:
  - Distributed: want to distribute responsibilities among existing nodes in the overlay
  - Adaptive: nodes join and leave the P2P overlay
    - distribute knowledge responsibility to joining nodes
    - redistribute responsibility knowledge from leaving nodes

DHT Step 1: The Hash

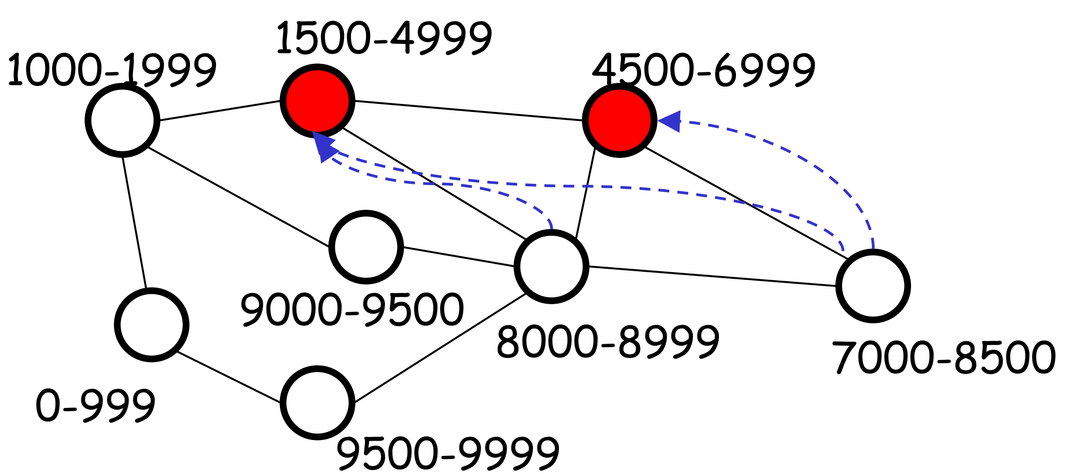
- Introduce a hash function to map the object being searched for to a unique identifier:
  - e.g., h("NGC'02 Tutorial Notes") → 8045
- Distribute the range of the hash function among all nodes in the network



- Each node must "know about" at least one copy of each object that hashes within its range (when one exists)

“Knowing about objects”

- Two alternatives
  - Node can cache each (existing) object that hashes within its range
  - Pointer-based: level of indirection - node caches pointer to location(s) of object



DHT Step 2: Routing

- For each object, node(s) whose range(s) cover that object must be reachable via a "short" path
- by the querier node (assumed can be chosen arbitrarily)
- by nodes that have copies of the object (when pointer-based approach is used)

- The different approaches (CAN,Chord,Pastry,Tapestry) differ fundamentally only in the routing approach
  - any "good" random hash function will suffice

DHT Routing: Other Challenges

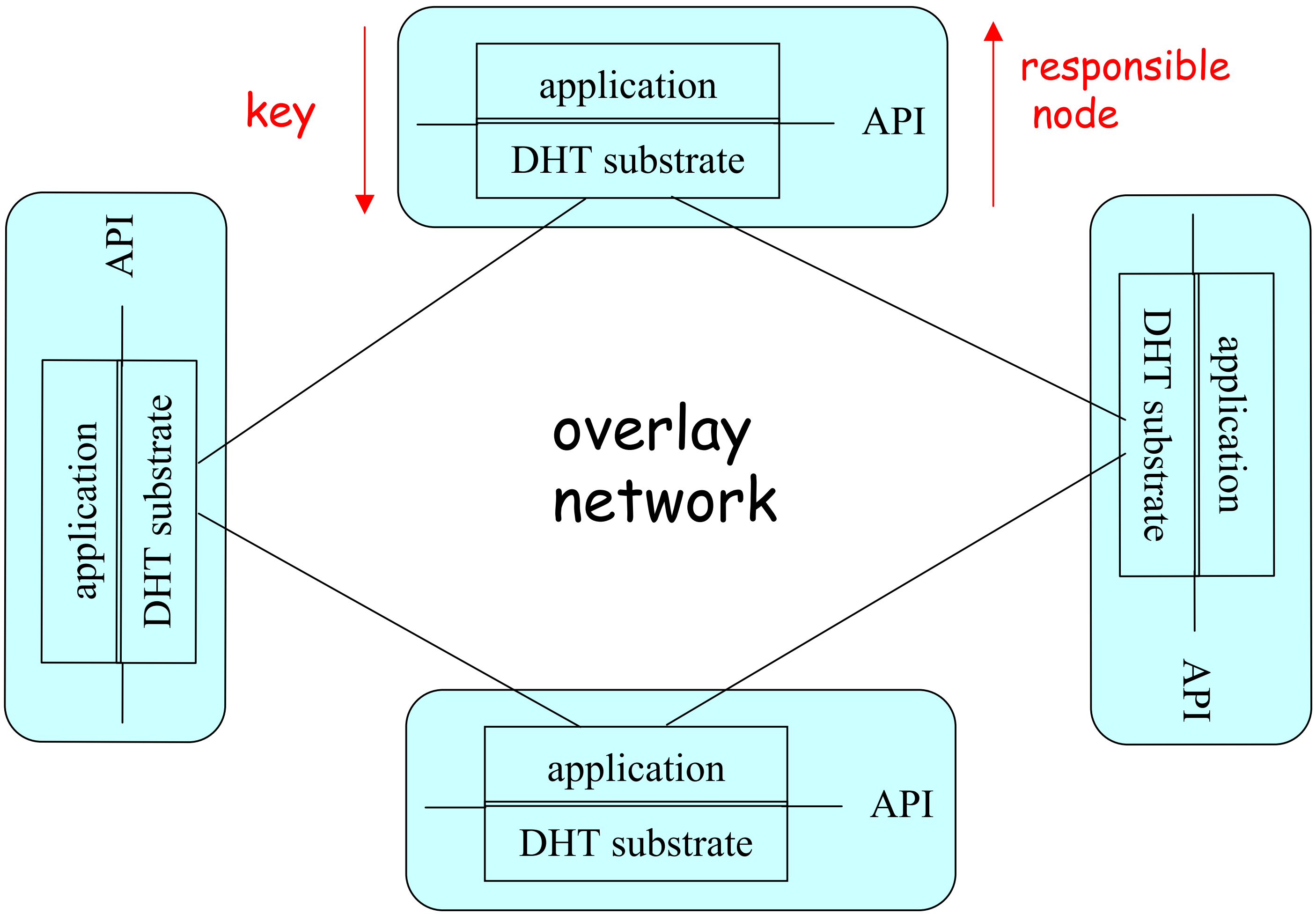
- # neighbors for each node should scale with growth in overlay participation (e.g., should not be O(N))
- DHT mechanism should be fully distributed (no centralized point that bottlenecks throughput or can act as single point of failure)
- DHT mechanism should gracefully handle nodes joining/leaving the overlay
  - need to repartition the range space over existing nodes
  - need to reorganize neighbor set
  - need bootstrap mechanism to connect new nodes into the existing DHT infrastructure

DHT API

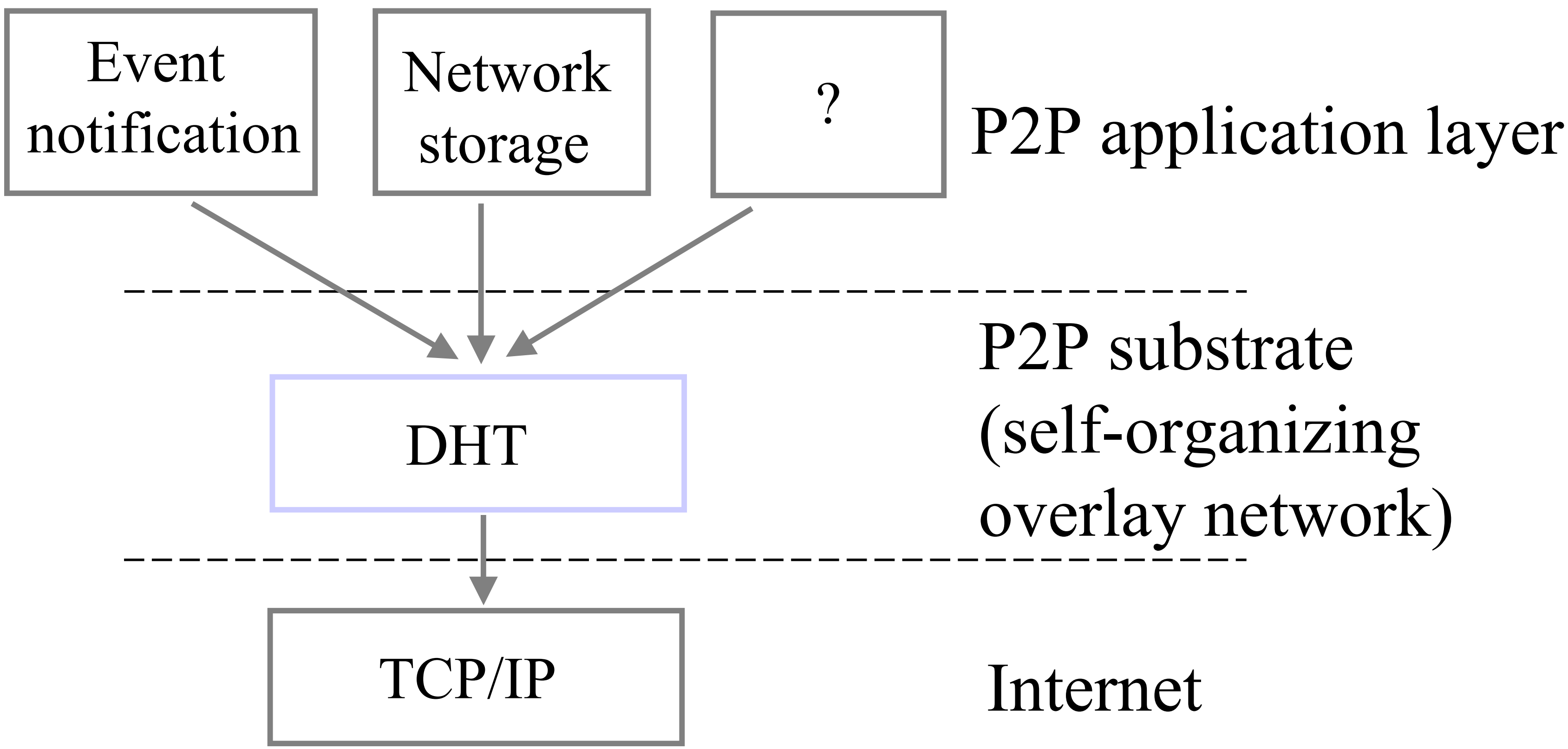
- each data item (e.g., file or metadata containing pointers) has a key in some ID space
- In each node, DHT software provides API:
  - Application gives API key k
  - API returns IP address of node that is responsible for k
- API is implemented with an underlying DHT overlay and distributed algorithms

DHT API

each data item (e.g., file or metadata pointing to file copies) has a key



DHT Layered Architecture



3. Structured P2P: DHT Approaches

- DHT service and issues
- CARP
- Consistent Hashing
- Chord
- CAN
- Pastry/Tapestry
- Hierarchical lookup services
- Topology-centric lookup service

CARP

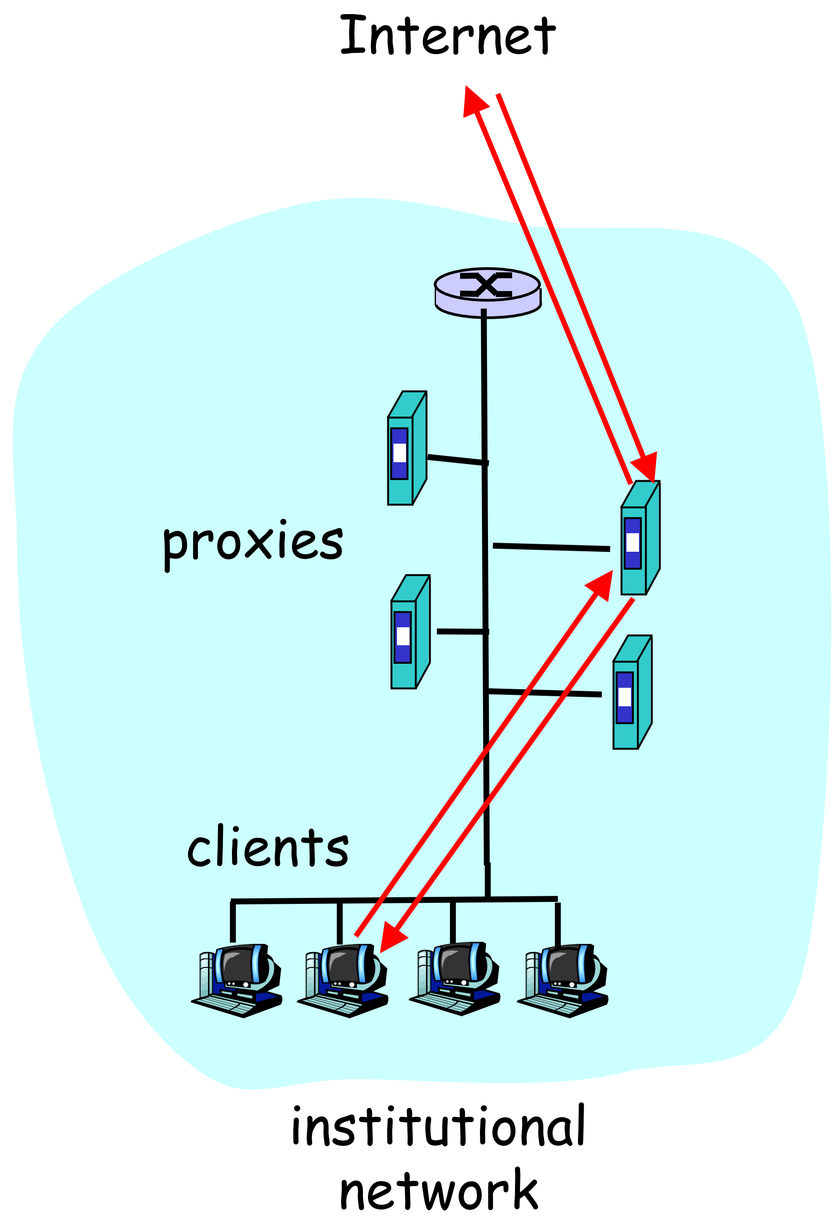
DHT for cache clusters

- Each proxy has unique name

key = URL = u

- calc h($proxy_n$, u) for all proxies
- assign u to proxy with highest h($proxy_n$, u)

if proxy added or removed, u is likely still in correct proxy



CARP (2)

- circa 1997
  - Internet draft: Valloppillil and Ross
- Implemented in Microsoft & Netscape products
- Browsers obtain script for hashing from proxy automatic configuration file (loads automatically)

Not good for P2P:

- Each node needs to know name of all other up nodes
- i.e., need to know O(N) neighbors
- But only O(1) hops in lookup

3. Structured P2P: DHT Approaches

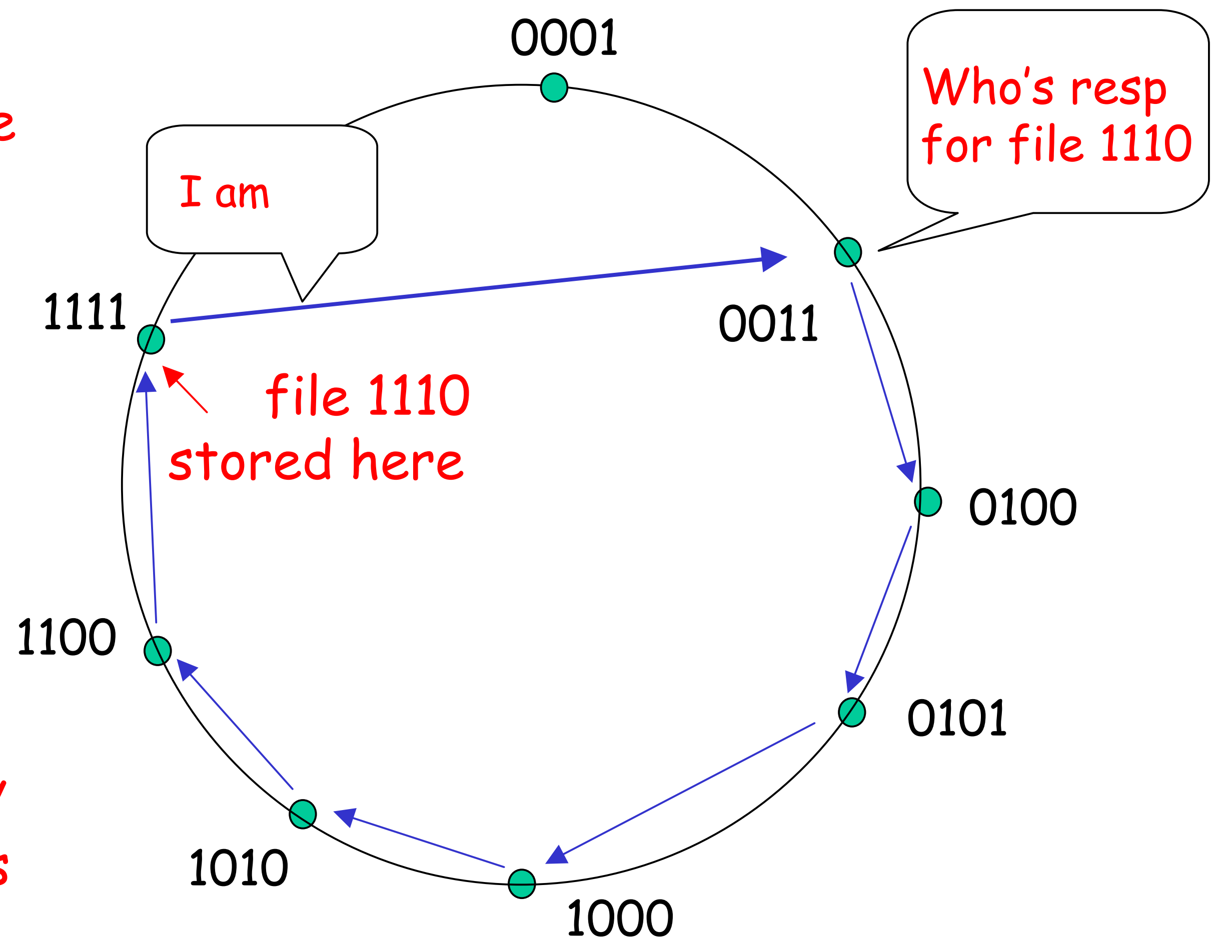
- DHT service and issues
- CARP
- Consistent Hashing
- Chord
- CAN
- Pastry/Tapestry
- Hierarchical lookup services
- Topology-centric lookup service

Consistent hashing (1)

- Overlay network is a circle
- Each node has randomly chosen id
  - Keys in same id space
- Node's successor in circle is node with next largest id
  - Each node knows IP address of its successor
- Key is stored in closest successor

Consistent hashing (2)

O(N) messages on avg to resolve query



Note: no locality among neighbors

Consistent hashing (3)

Node departures

- Each node must track $s \geq 2$ successors
- If your successor leaves, take next one
- Ask your new successor for list of its successors; update your s successors

Node joins

- You're new, node id k
- ask any node n to find the node n' that is the successor for id k
- Get successor list from n'
- Tell your predecessors to update their successor lists
- Thus, each node must track its predecessor

Consistent hashing (4)

- Overlay is actually a circle with small chords for tracking predecessor and k successors
- # of neighbors = s+1: O(1)
  - The ids of your neighbors along with their IP addresses is your "routing table"
- average # of messages to find key is O(N)

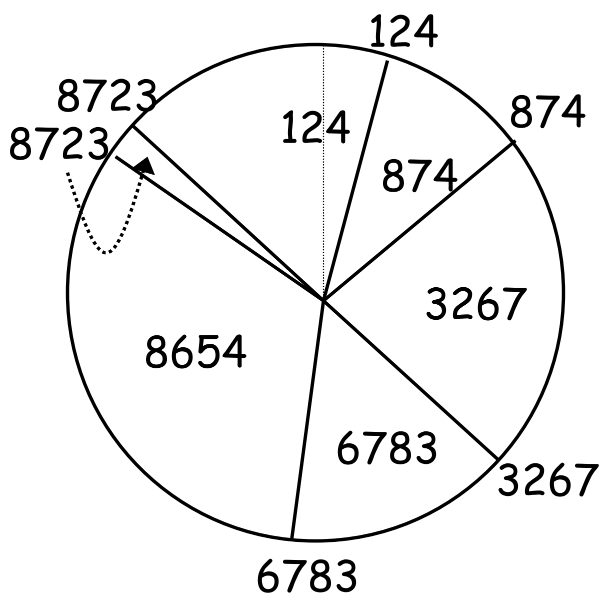
Can we do better?

3. Structured P2P: DHT Approaches

- DHT service and issues
- CARP
- Consistent Hashing
- Chord
- CAN
- Pastry/Tapestry
- Hierarchical lookup services
- Topology-centric lookup service

Chord

- Nodes assigned 1-dimensional IDs in hash space at random (e.g., hash on IP address)
- Consistent hashing: Range covered by node is from previous ID up to its own ID (modulo the ID space)



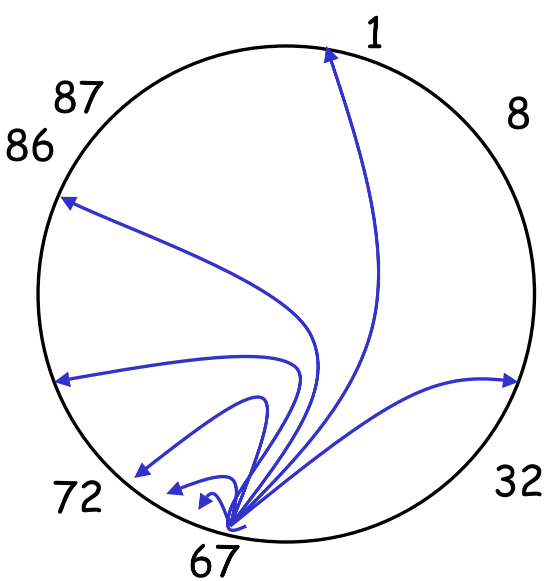
Chord Routing

- A node s's i^{th} neighbor has the ID that is equal to $s+2^i$ or is the next largest ID (mod ID space), $i \geq 0$
- To reach the node handling ID t, send the message to neighbor $\#\log_2(t-s)$
- Requirement: each node s must know about the next node that exists clockwise on the Chord (0^{th} neighbor)
- Set of known neighbors called a finger table

Chord Routing (cont'd)

- A node s is node t's neighbor if s is the closest node to $t+2^i$ mod H for some i. Thus,
  - each node has at most $\log_2$ N neighbors
  - for any object, the node whose range contains the object is reachable from any node in no more than $\log_2$ N overlay hops

    (each step can always traverse at least half the distance to the ID)
- Given K objects, with high probability each node has at most $(1 + \log_2 N) K / N$ in its range
- When a new node joins or leaves the overlay, O(K / N) objects move between nodes



i	Finger table for node 67
0	72
1	72
2	72
3	86
4	86
5	1
6	32

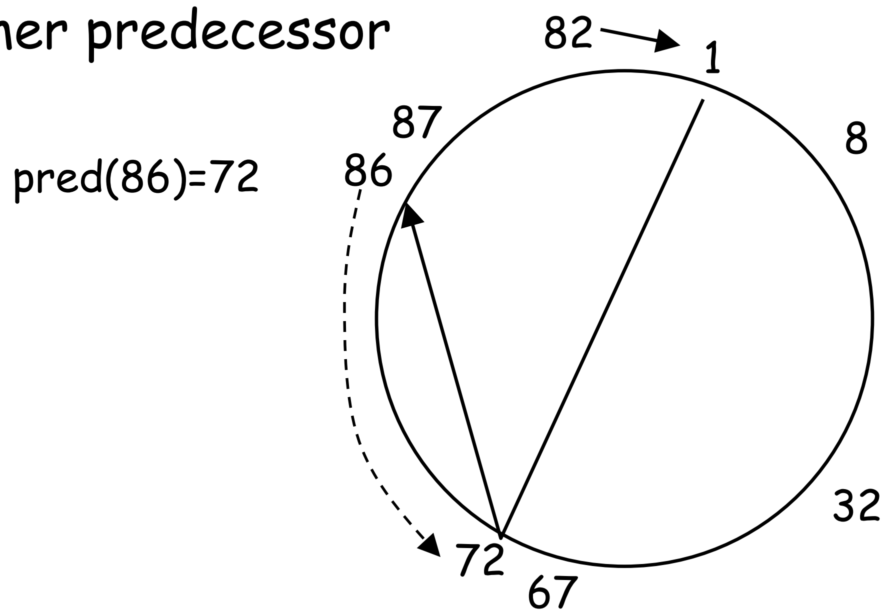
Closest node clockwise to

$67+2^i$ mod 100

Chord Node Insertion

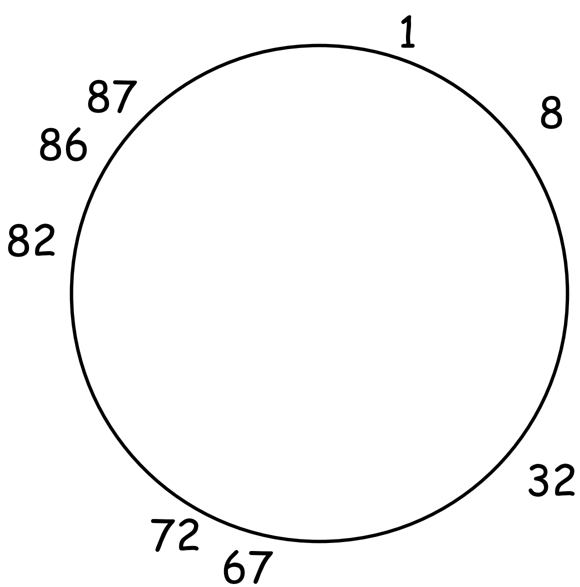
- One protocol addition: each node knows its closest counter-clockwise neighbor
- A node selects its unique (pseudo-random) ID and uses a bootstrapping process to find some node in the Chord
- Using Chord, the node identifies its successor in the clockwise direction
- An newly inserted node's predecessor is its successor's former predecessor

Example: Insert 82



Chord Node Insertion (cont'd)

- First: set added node s's fingers correctly
  - s's predecessor t does the lookup for each distance of 2^i from s



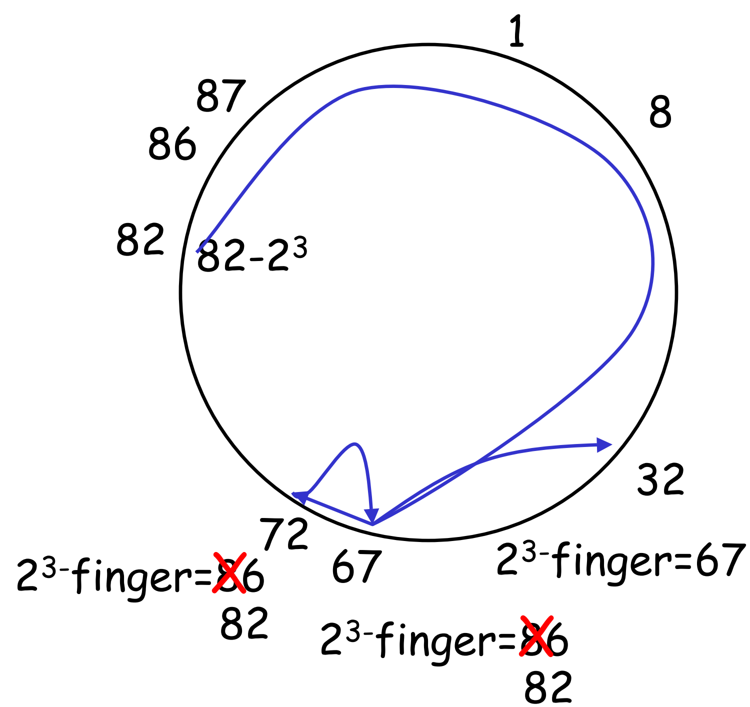
Lookups from node 72

Lookup(83) = 86 →
Lookup(84) = 86 →
Lookup(86) = 86 →
Lookup(90) = 1 →
Lookup(98) = 1 →
Lookup(14) = 32 →
Lookup(46) = 67 →

i	Finger table for node 82
0	86
1	86
2	86
3	1
4	1
5	32
6	67

Chord Node Insertion (cont'd)

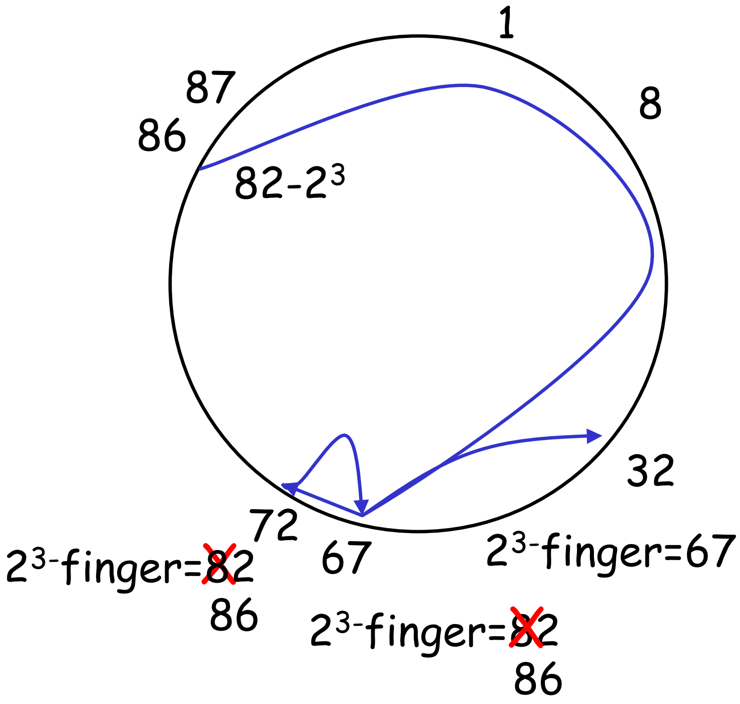
- Next, update other nodes' fingers about the entrance of s (when relevant). For each i:
  - Locate the closest node to s (counter-clockwise) whose 2^i-finger can point to s: largest possible is $s - 2^i$
  - Use Chord to go (clockwise) to largest node t before or at $s - 2^i$
    - route to $s - 2^i$, if arrived at a larger node, select its predecessor as t
  - If t's 2^i-finger routes to a node larger than s
    - change t's 2^i-finger to s
    - set t = predecessor of t and repeat
  - Else i++, repeat from top
- $O(\log^2 N)$ time to find and update nodes



e.g., for i=3

Chord Node Deletion

- Similar process can perform deletion



e.g., for i=3

3. Structured P2P: DHT Approaches

- DHT service and issues
- CARP
- Consistent Hashing
- Chord
- CAN
- Pastry/Tapestry
- Hierarchical lookup services
- Topology-centric lookup service

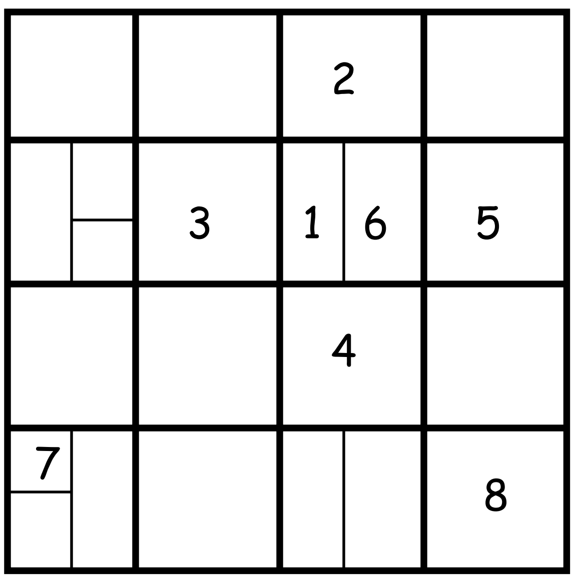
CAN

- hash value is viewed as a point in a D-dimensional cartesian space
- each node responsible for a D-dimensional "cube" in the space
- nodes are neighbors if their cubes "touch" at more than just a point

  (more formally, nodes s & t are neighbors when
  - s contains some

    $[\langle n_1, n_2, \ldots, n_i, \ldots, n_j, \ldots, n_D \rangle, \langle n_1, n_2, \ldots, m_i, \ldots, n_j, \ldots\, n_D \rangle]$
  - and t contains

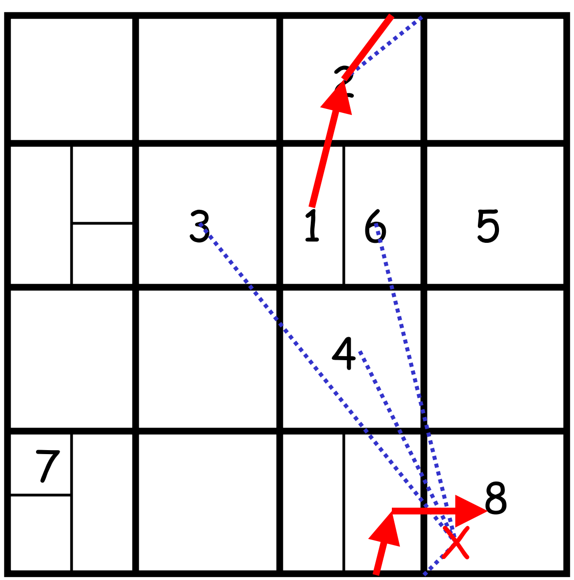
    $[\langle n_1, n_2, \ldots, n_i, \ldots, n_j + \delta, \ldots, n_D \rangle, \langle n_1, n_2, \ldots, m_i, \ldots, n_j + \delta, \ldots\, n_D \rangle]$)



- Example: D=2
- 1's neighbors: 2,3,4,6
- 6's neighbors: 1,2,4,5
- Squares "wrap around", e.g., 7 and 8 are neighbors
- expected # neighbors: O(D)

CAN routing

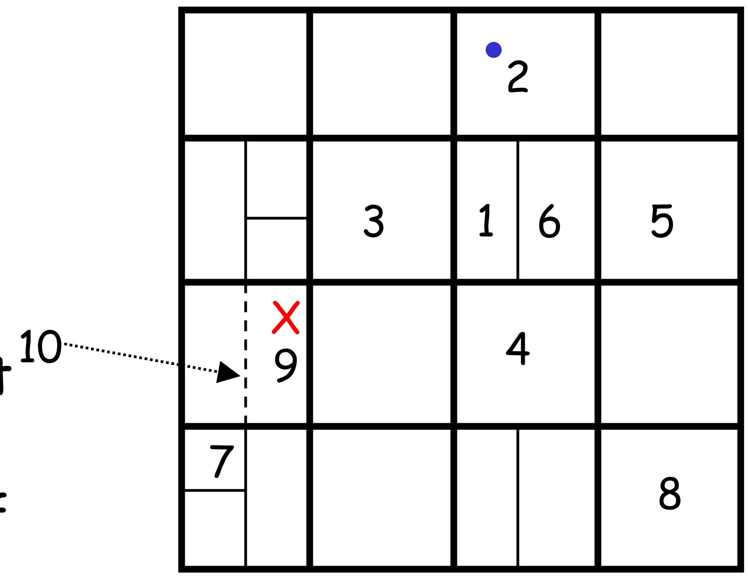
- To get to $\langle n_1, n_2, \ldots, n_D \rangle$ from $\langle m_1, m_2, \ldots, m_D \rangle$
  - choose a neighbor with smallest cartesian distance from $\langle m_1, m_2, \ldots, m_D \rangle$ (e.g., measured from neighbor's center)



- e.g., region 1 needs to send to node covering X
- checks all neighbors, node 2 is closest
- forwards message to node 2
- Cartesian distance monotonically decreases with each transmission
- expected # overlay hops: $(DN^{1/D})/4$

CAN node insertion

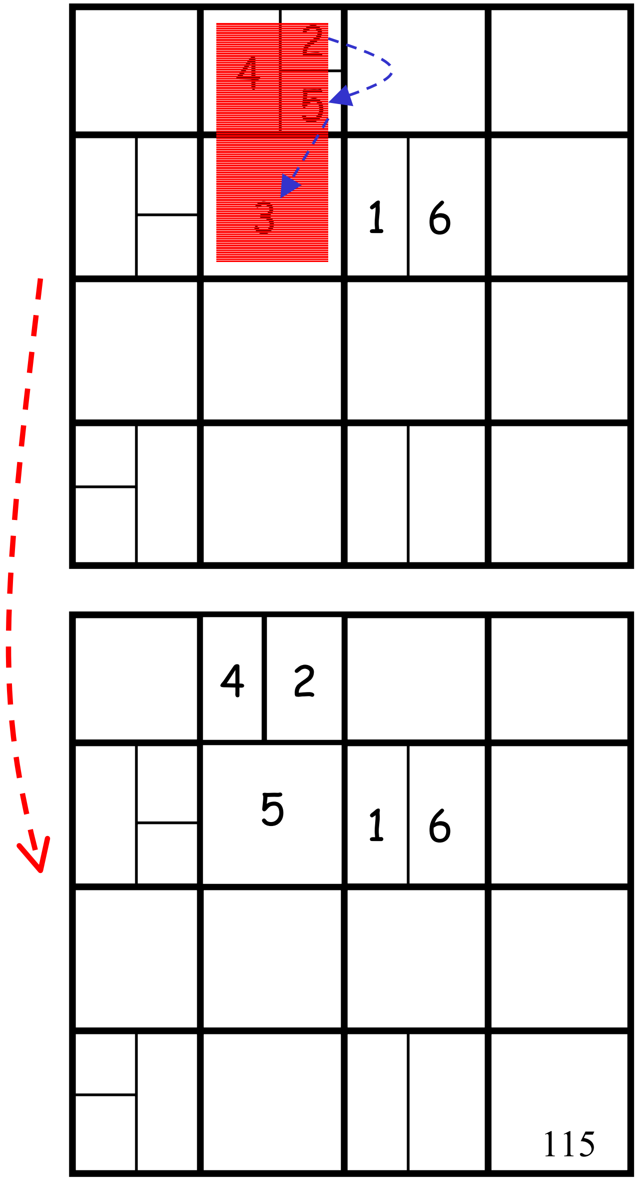
- To join the CAN:
  - find some node in the CAN (via bootstrap process)
  - choose a point in the space uniformly at random
  - using CAN, inform the node that currently covers the space
  - that node splits its space in half
    - 1st split along 1st dimension
    - if last split along dimension i < D, next split along i+1st dimension
    - e.g., for 2-d case, split on x-axis, then y-axis
  - keeps half the space and gives other half to joining node



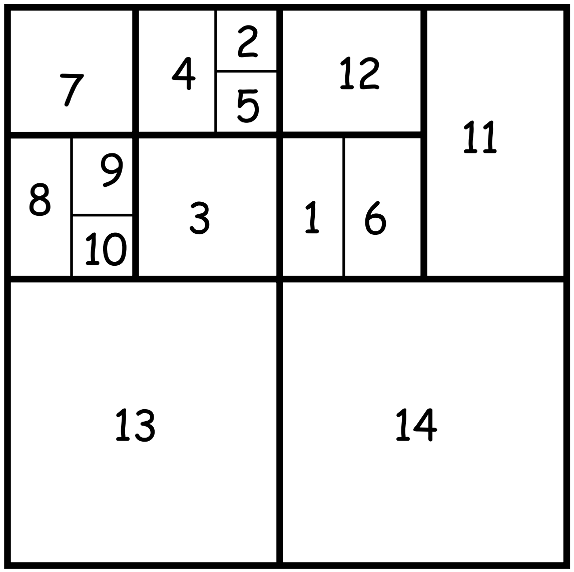
Observation: the likelihood of a rectangle being selected is proportional to it's size, i.e., big rectangles chosen more frequently

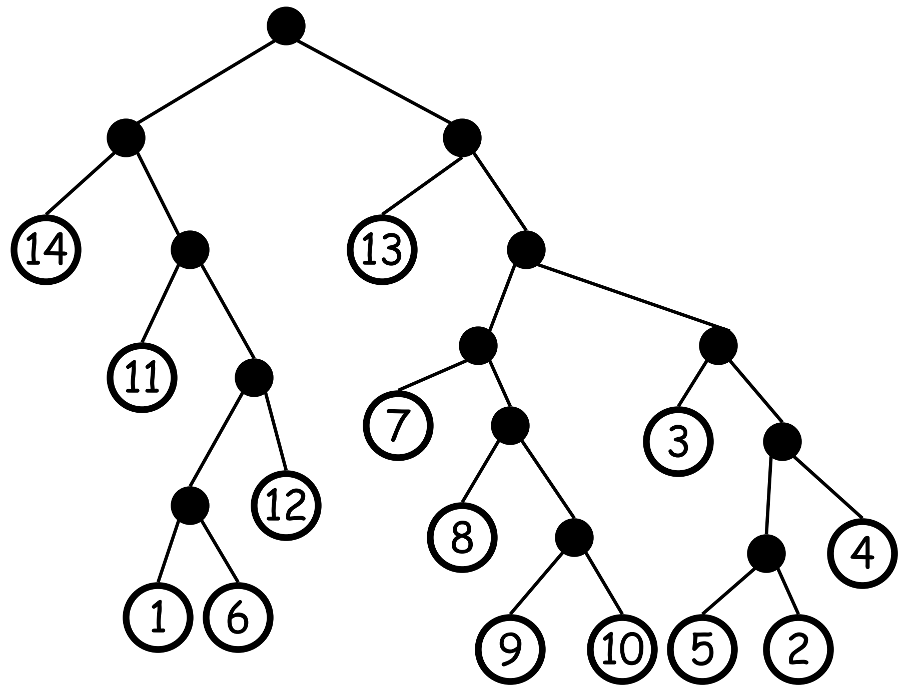
CAN node removal

- Underlying cube structure should remain intact
  - i.e., if the spaces covered by s & t were not formed by splitting a cube, then they should not be merged together
- Sometimes, can simply collapse removed node's portion to form bigger rectangle
  - e.g., if 6 leaves, its portion goes back to 1
- Other times, requires juxtaposition of nodes' areas of coverage
  - e.g., if 3 leaves, should merge back into square formed by 2,4,5
  - cannot simply collapse 3's space into 4 and/or 5
  - one solution: 5's old space collapses into 2's space, 5 takes over 3's space



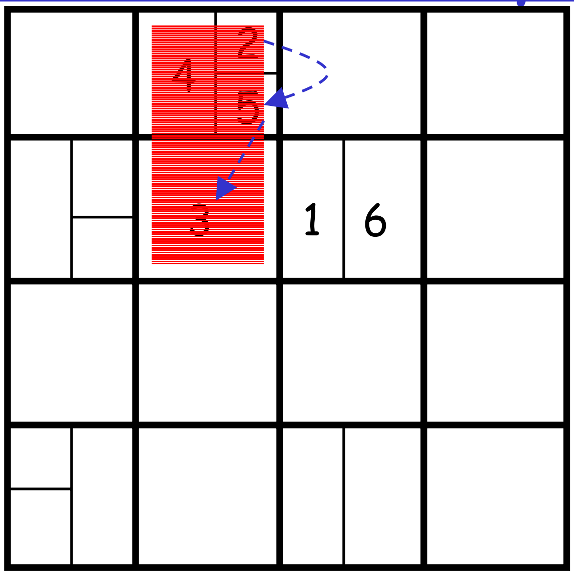
CAN (recovery from) removal process

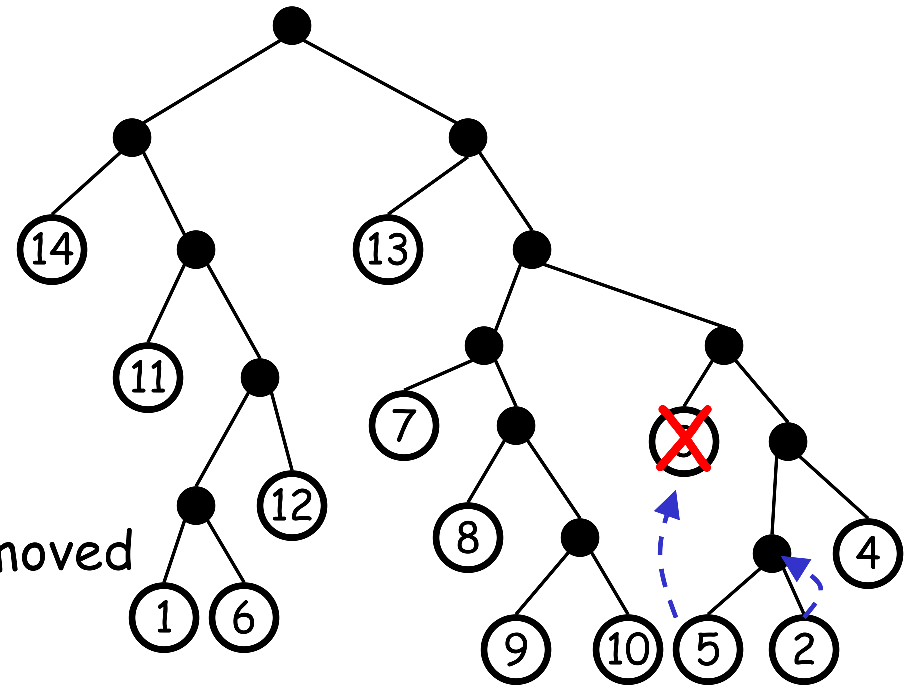




- View partitioning as a binary tree of
  - leaves represent regions covered by overlay nodes (labeled by node that covers the region)
  - intermediate nodes represent "split" regions that could be "reformed", i.e., a leaf can appear at that position
  - siblings are regions that can be merged together (forming the region that is covered by their parent)

CAN (recovery from) removal process





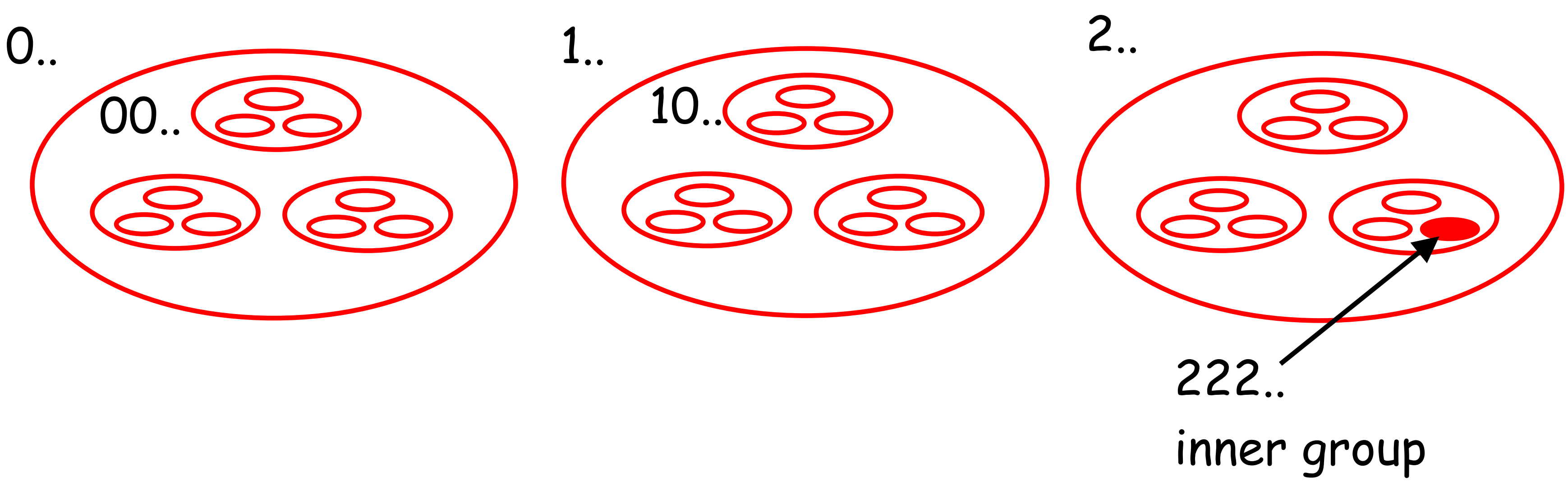
- Repair algorithm when leaf s is removed
  - Find a leaf node t that is either
    - s's sibling
    - descendant of s's sibling where t's sibling is also a leaf node
  - t takes over s's region (moves to s's position on the tree)
  - t's sibling takes over t's previous region
- Distributed process in CAN to find appropriate t w/ sibling:
  - current (inappropriate) t sends msg into area that would be covered by a sibling
  - if sibling (same size region) is there, then done. Else receiving node becomes t & repeat

3. Structured P2P: DHT Approaches

- DHT service and issues
- CARP
- Consistent Hashing
- Chord
- CAN
- Pastry
- Hierarchical lookup services
- Topology-centric lookup service

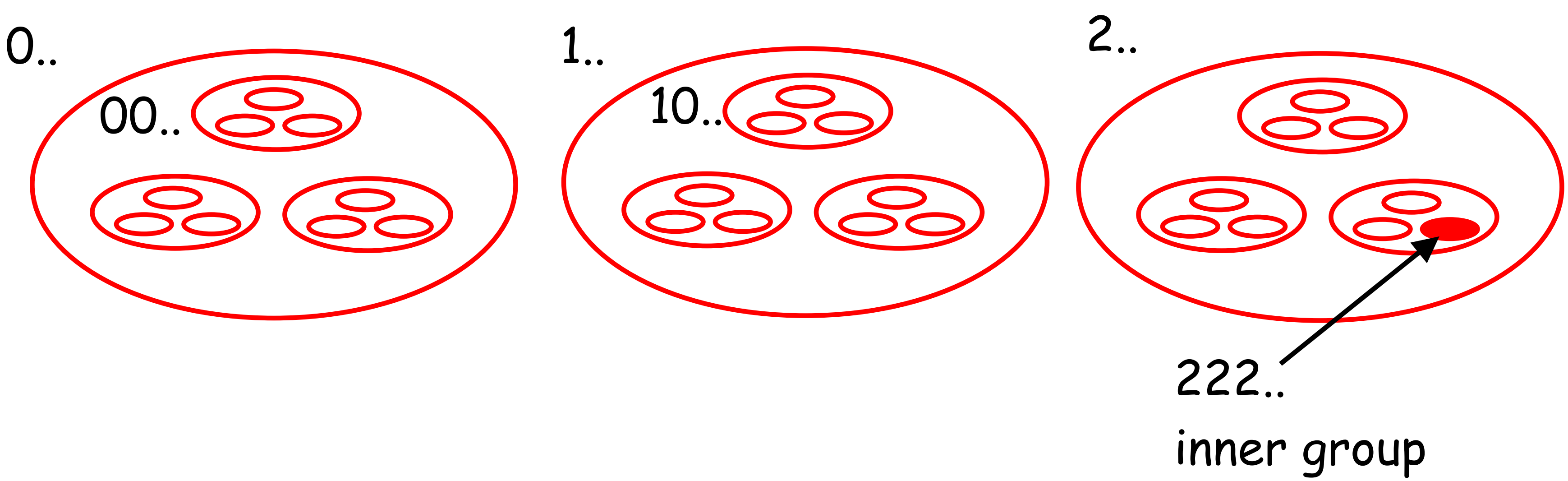
Pseudo-Pastry: basic idea

- Example: nodes & keys have n-digit base-3 ids, eg, 02112100101022
- Each key is stored in node with closest id
- Node addressing defines nested groups



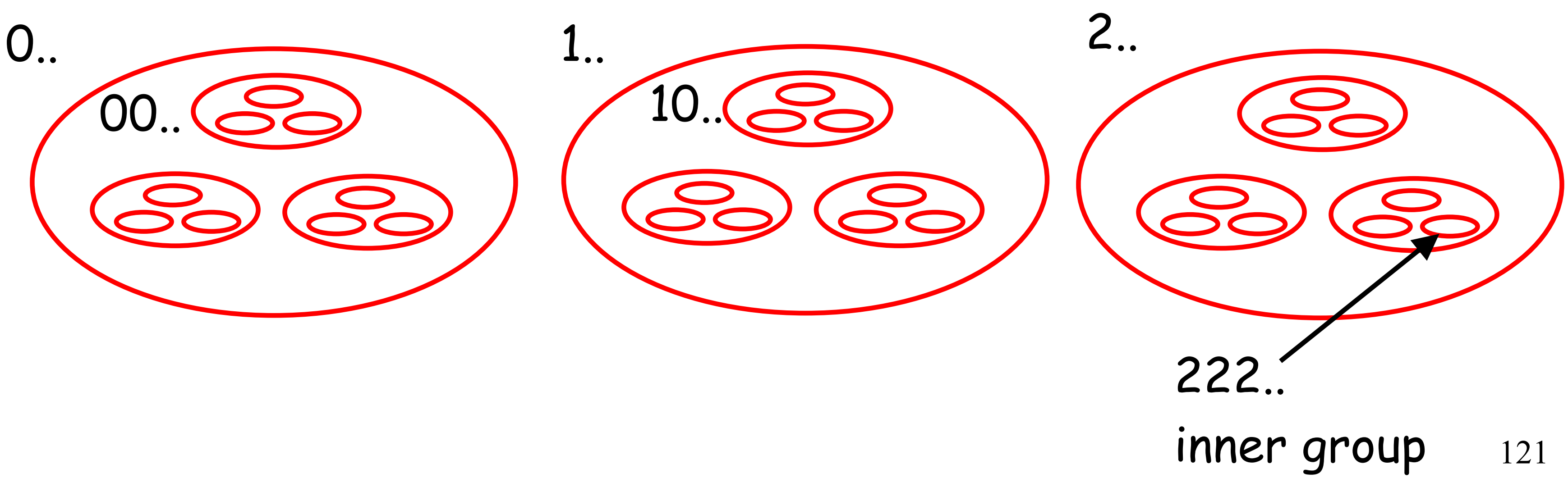
Pseudo-Pastry (2)

- Nodes in same inner group know each other's IP address
- Each node knows IP address of one delegate node in some of the other groups



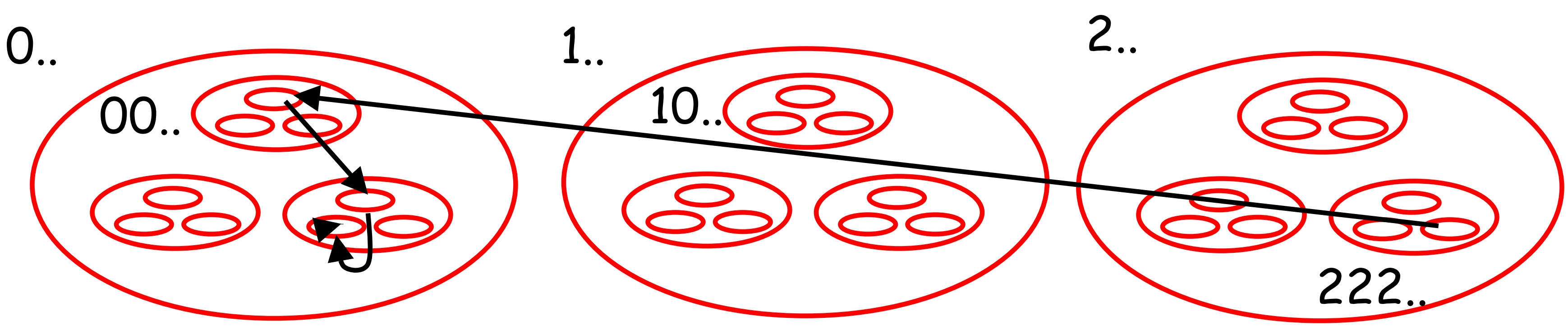
Pastry: basic idea (3)

- Each node needs to know the IP addresses of all up nodes in its inner group.
- Each node needs to know IP addresses of some delegate nodes. Which delegate nodes?
- Node in 222…: 0…, 1…, 20…, 21…, 220…, 221…
- Thus, 6 delegate nodes rather than 27



Pseudo-Pastry (4)

- Suppose node in group 222… wants to lookup key k= 02112100210. Divide and conquer
- Forward query to node node in 0…, then to node in 02…, then to node in 021…
- Node in 021… forwards to closest to key in 1 hop



Pastry (in truth)

- Nodes are assigned a 128-bit identifier
- The identifier is viewed in base 16
  - e.g., 65a1fc04
  - 16 subgroups for each group
- Each node maintains a routing table and a leaf set
  - routing table provides delegate nodes in nested groups
  - inner group idea flawed: might be empty or have too many nodes

Routing table (node: 65a1fc04)

Row 0	*0x*	*1x*	*2x*	*3x*	*4x*	*5x*		*7x*	*8x*	*9x*	*ax*	*bx*	*cx*	*dx*	*ex*	*fx*
Row 1	*60x*	*61x*	*62x*	*63x*	*64x*		*66x*	*67x*	*68x*	*69x*	*6ax*	*6bx*	*6cx*	*6dx*	*6ex*	*6fx*
Row 2	*650x*	*651x*	*652x*	*653x*	*654x*	*655x*	*656x*	*657x*	*658x*	*659x*		*65bx*	*65cx*	*65dx*	*65ex*	*65fx*
Row 3	*65a0x*		*65a2x*	*65a3x*	*65a4x*	*65a5x*	*65a6x*	*65a7x*	*65a8x*	*65a9x*	*65aax*	*65abx*	*65acx*	*65adx*	*65aex*	*65afx*

$\log_{16} N$ rows

Pastry: Routing procedure

if (destination is within range of our leaf set)
forward to numerically closest member
else
if (there's a longer prefix match in table)
forward to node with longest match
else
forward to node in table
(a) shares at least as long a prefix
(b) is numerically closer than this node

Pastry: Performance

Integrity of overlay/ message delivery:

- guaranteed unless *L/2* simultaneous failures of nodes with adjacent nodeIds

Number of routing hops:

- No failures: < $\log_{16} N$ expected
- During failure recovery:
  - *O(N)* worst case, average case much better

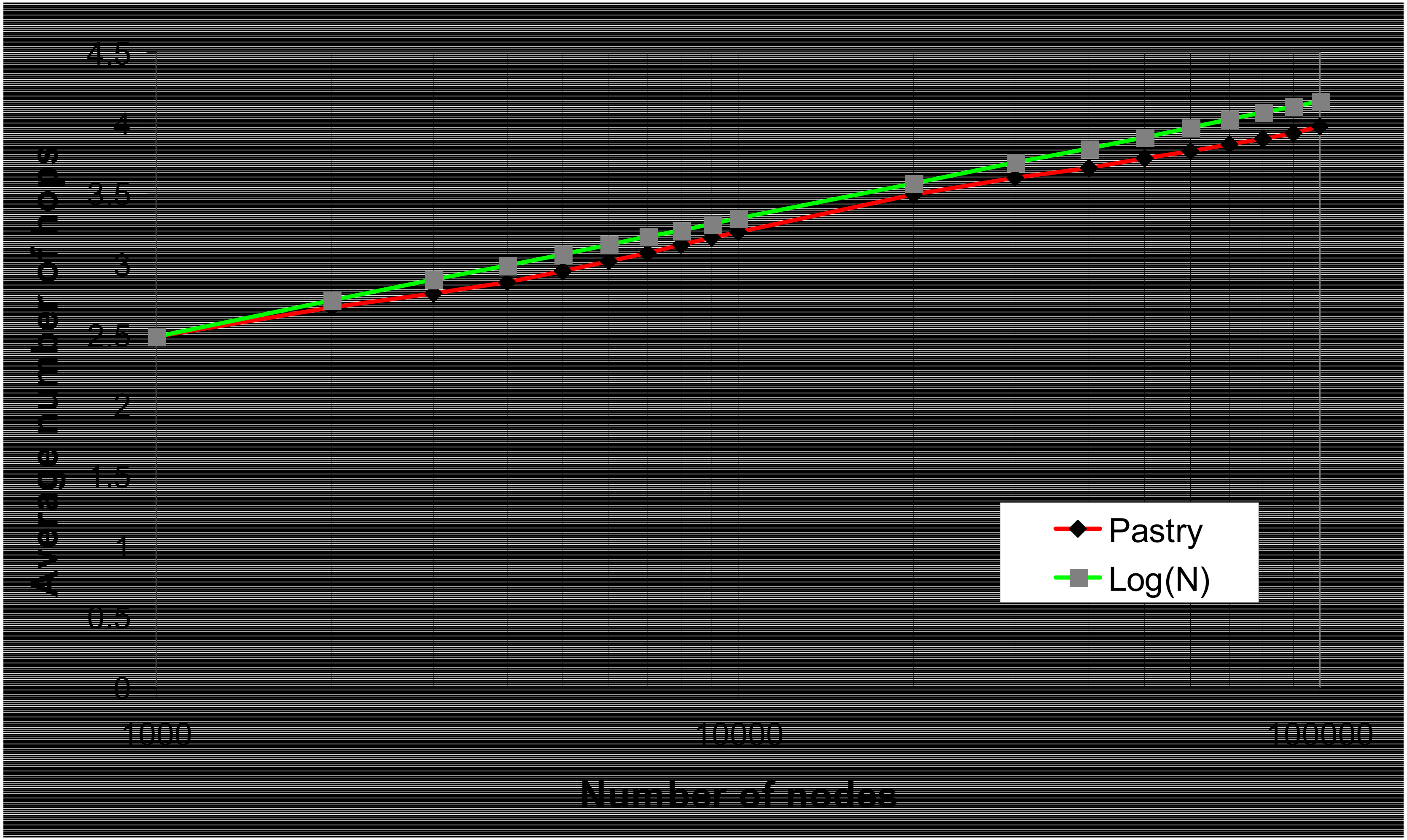
Pastry: Experimental results

Prototype

- implemented in Java
  - deployed testbed (currently ~25 sites worldwide)

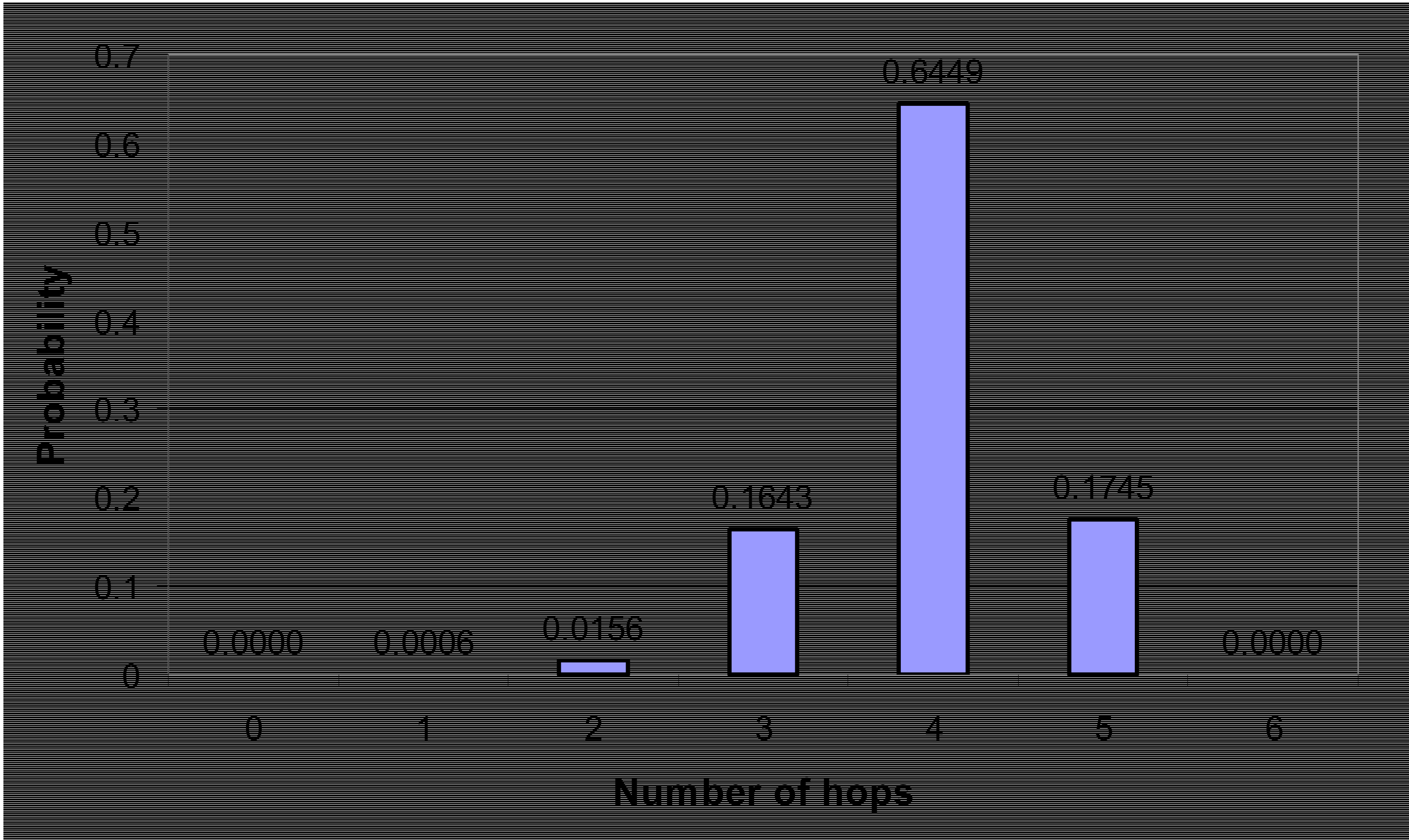
Simulations for large networks

Pastry: Average # of hops



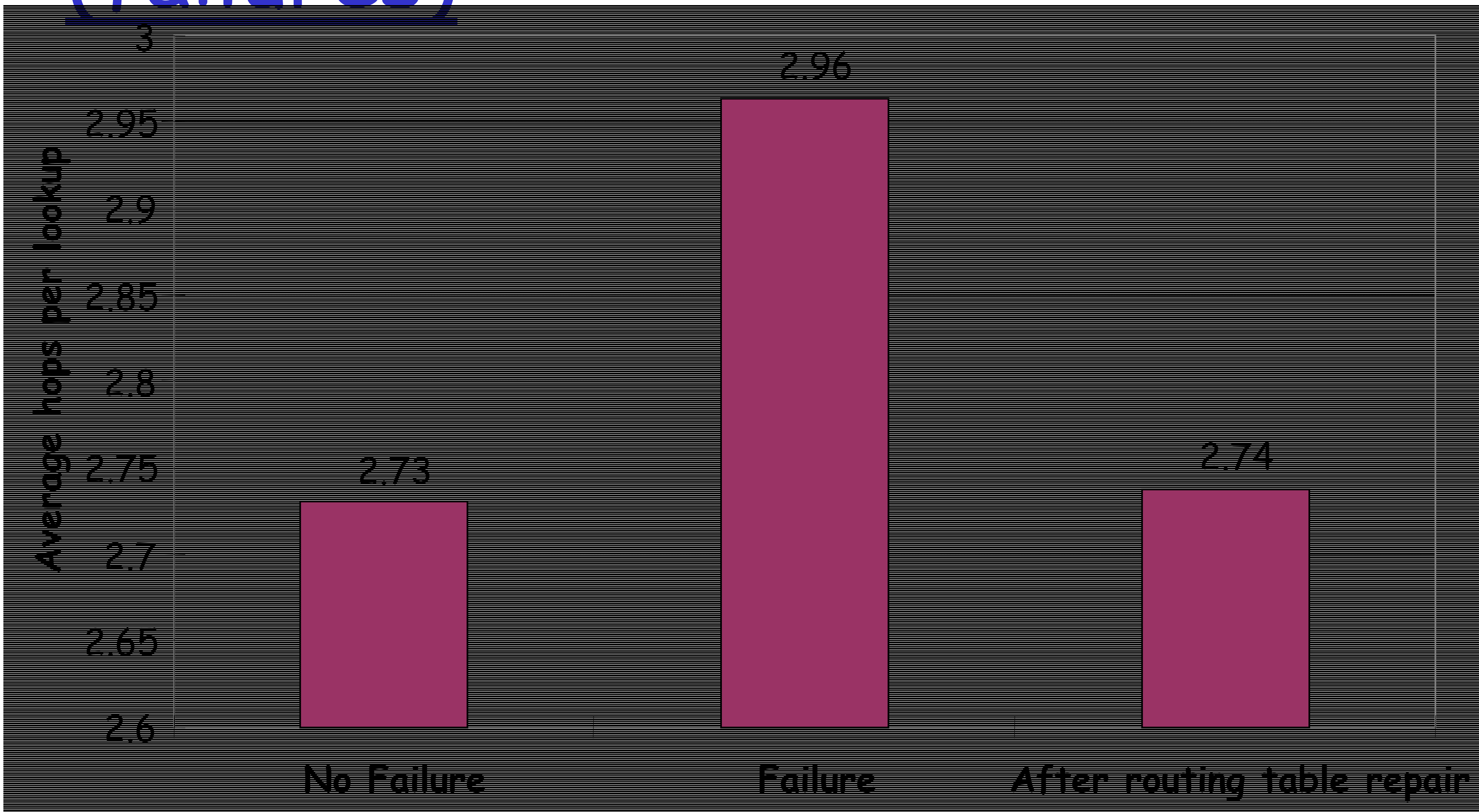
L=16, 100k random queries

Pastry: # of hops (100k nodes)



L=16, 100k random queries

Pastry: # routing hops (failures)



L=16, 100k random queries, 5k nodes, 500 failures

Pastry: Proximity routing

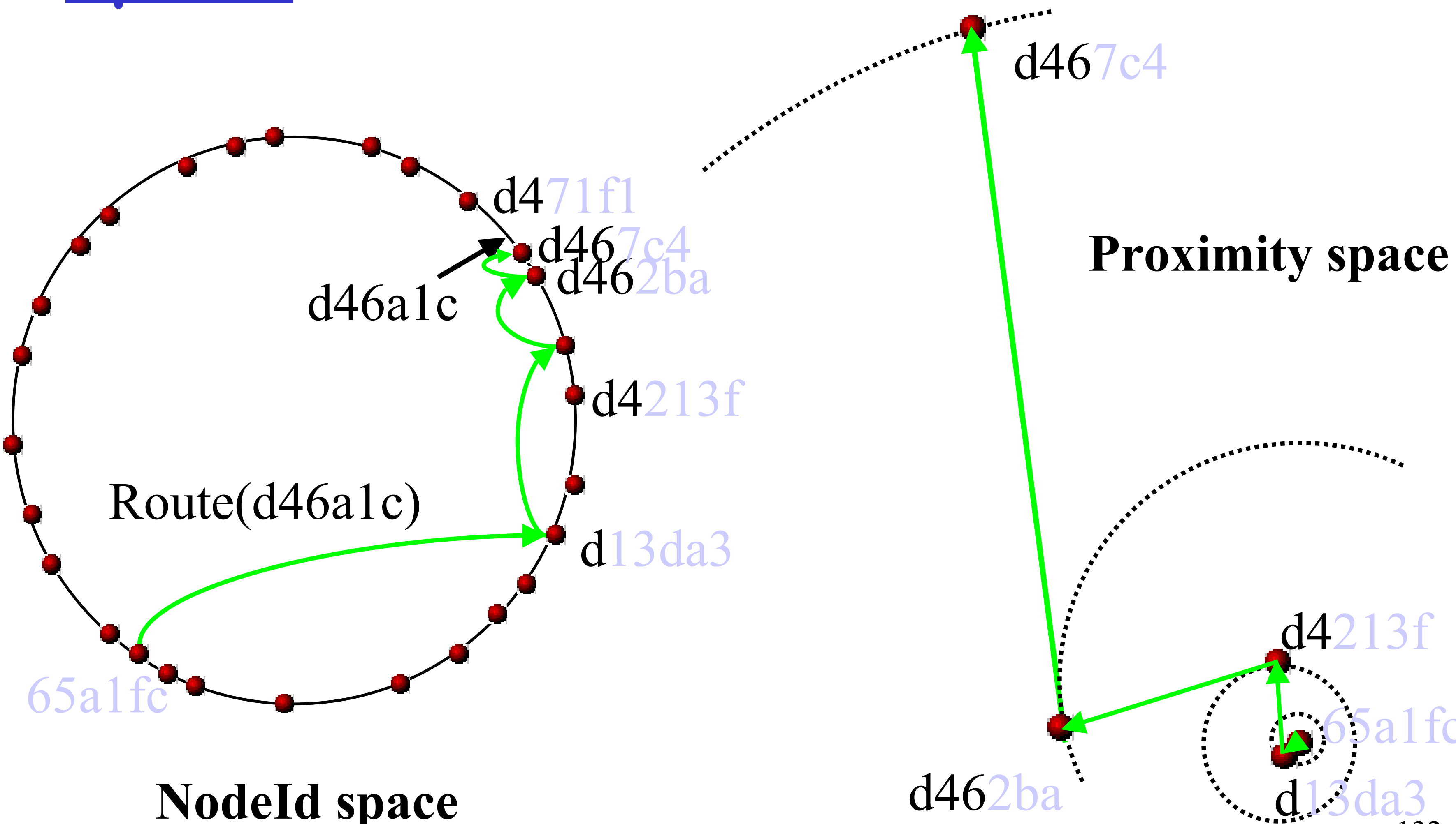
Assumption: scalar proximity metric

- e.g. ping delay, # IP hops
- a node can probe distance to any other node

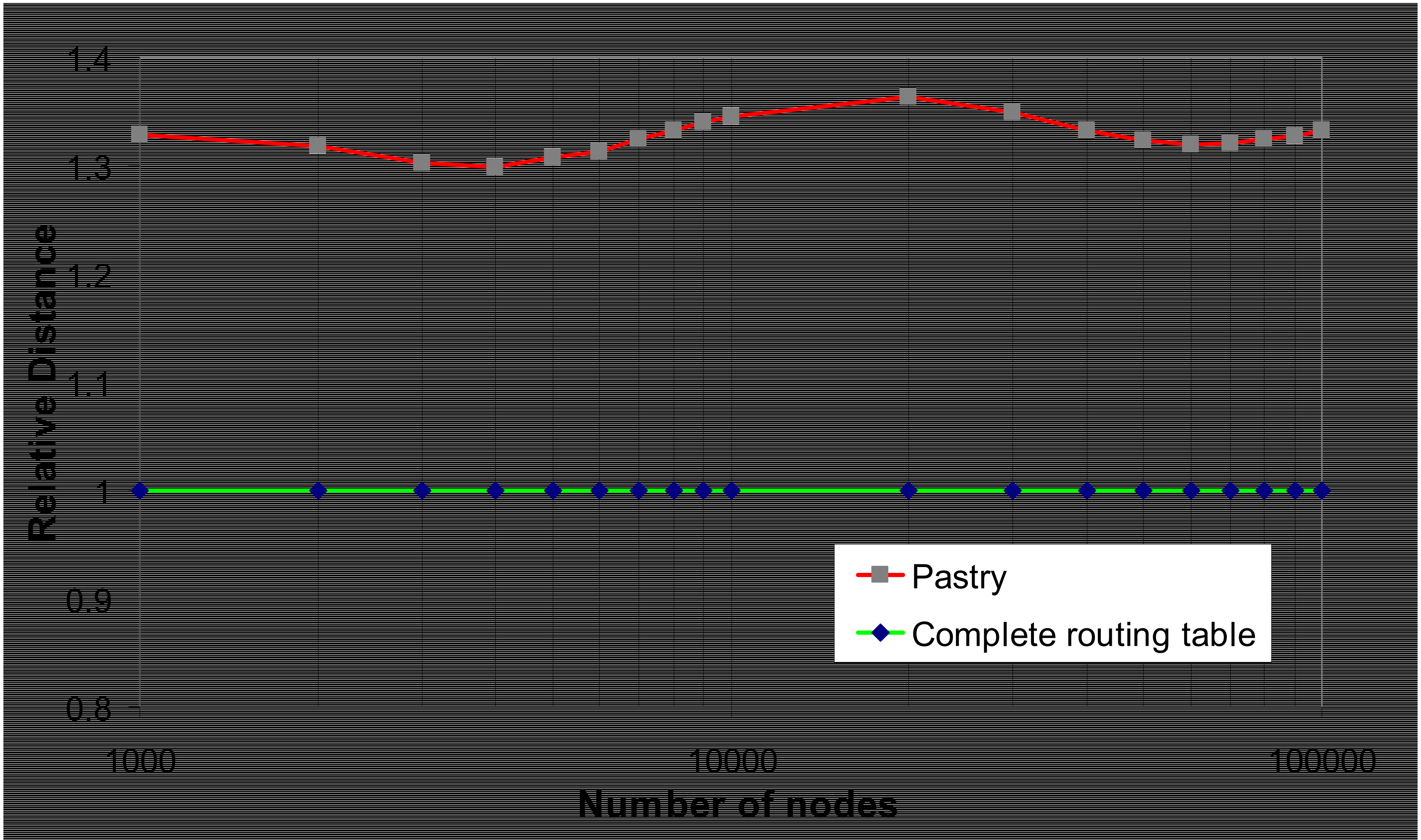
Proximity invariant:

Each routing table entry refers to a node close to the local node (in the proximity space), among all nodes with the appropriate nodeId prefix.

Pastry: Routes in proximity space



Pastry: Distance traveled

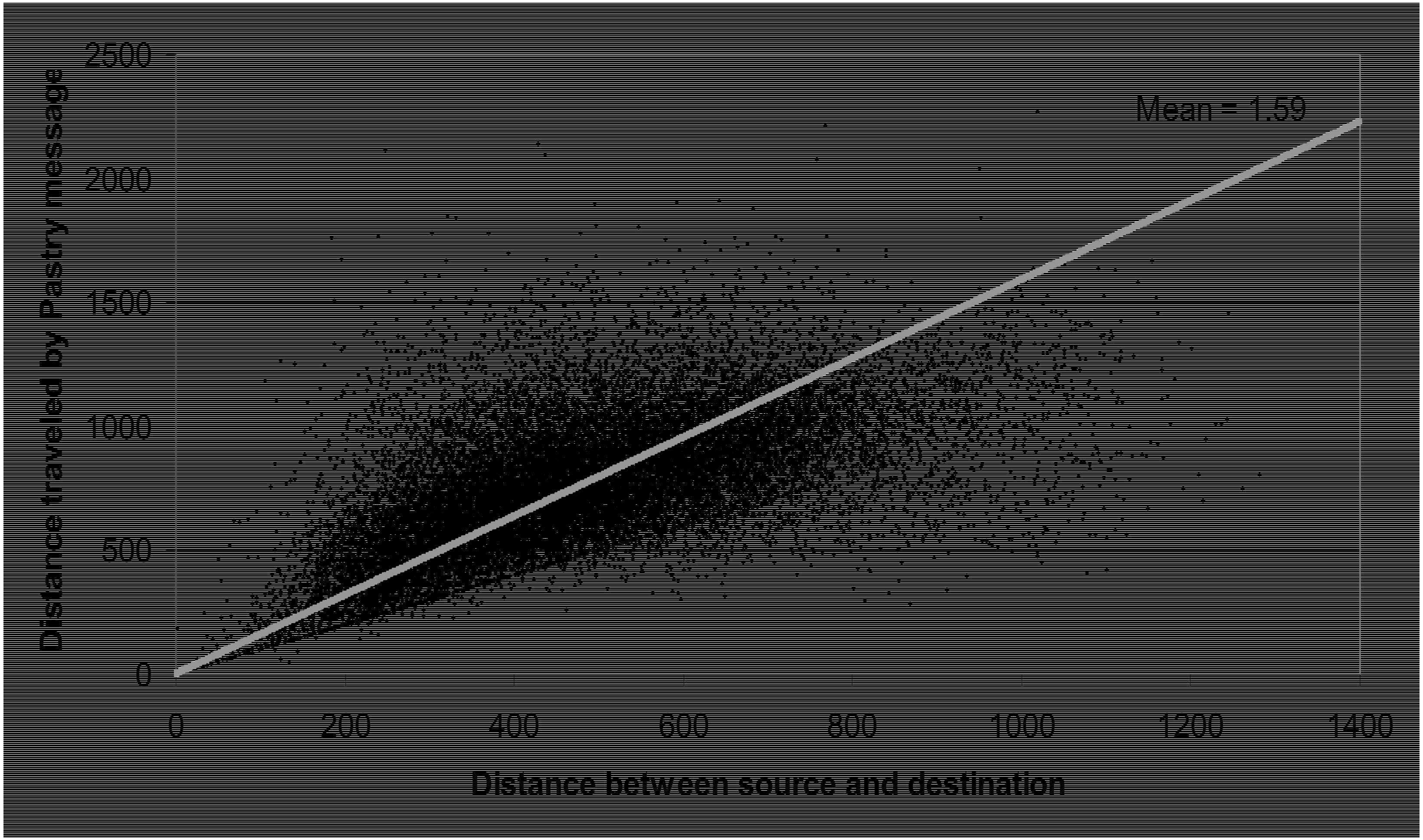


L=16, 100k random queries, Euclidean proximity space

Pastry: Locality properties

1) *Expected distance traveled by a message in the proximity space is within a small constant of the minimum*
2) *Routes of messages sent by nearby nodes with same keys converge at a node near the source nodes*
3) *Among k nodes with nodeIds closest to the key, message likely to reach the node closest to the source node first*

Pastry delay vs IP delay



GATech top., .5M hosts, 60K nodes, 20K random messages

Pastry: Summary

O(log N) routing steps (expected)

- *O(log N)* routing table size
- Network proximity routing

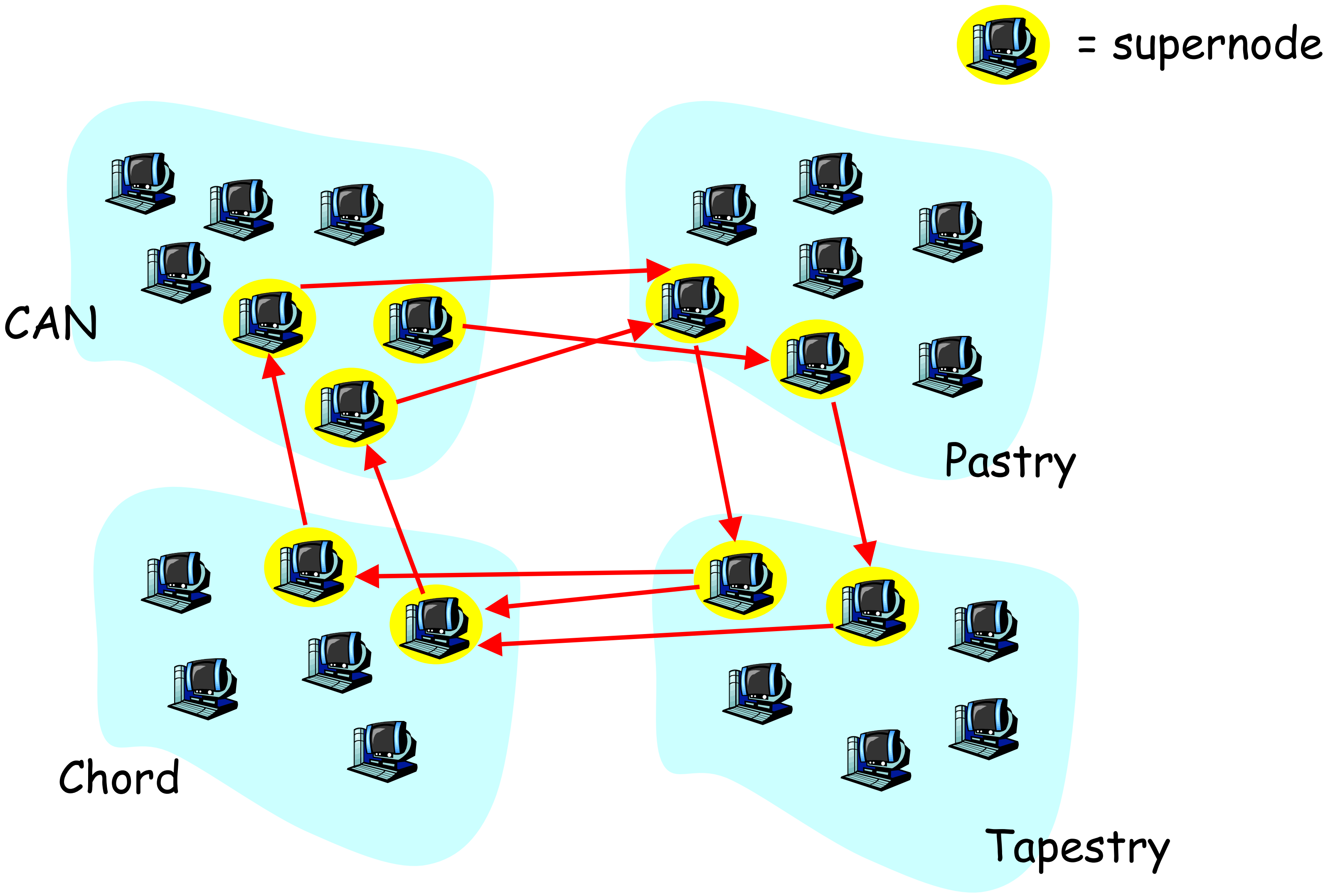
3. Structured P2P: DHT Approaches

- The DHT service and API
- CARP
- Consistent Hashing
- Chord
- CAN
- Pastry/Tapestry
- Hierarchical lookup services
- Topology-centric lookup service

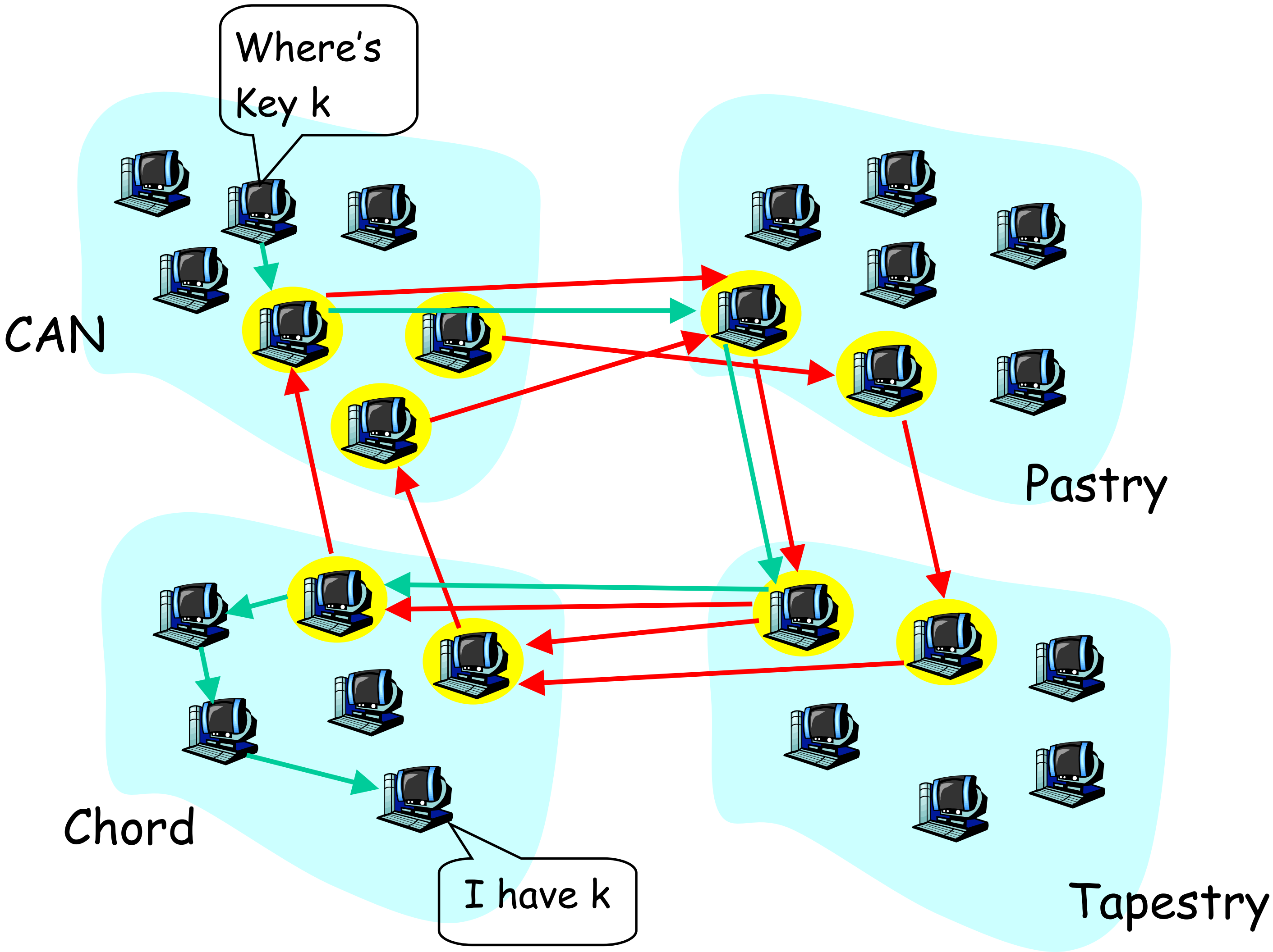
Hierarchical Lookup Service

- KaZaA is hierarchical but unstructured
- Routing in Internet is hierarchical
- Perhaps DHTs can benefit from hierarchies too?
  - Peers are organized into groups
  - Inter-group lookup, then intra-group lookup

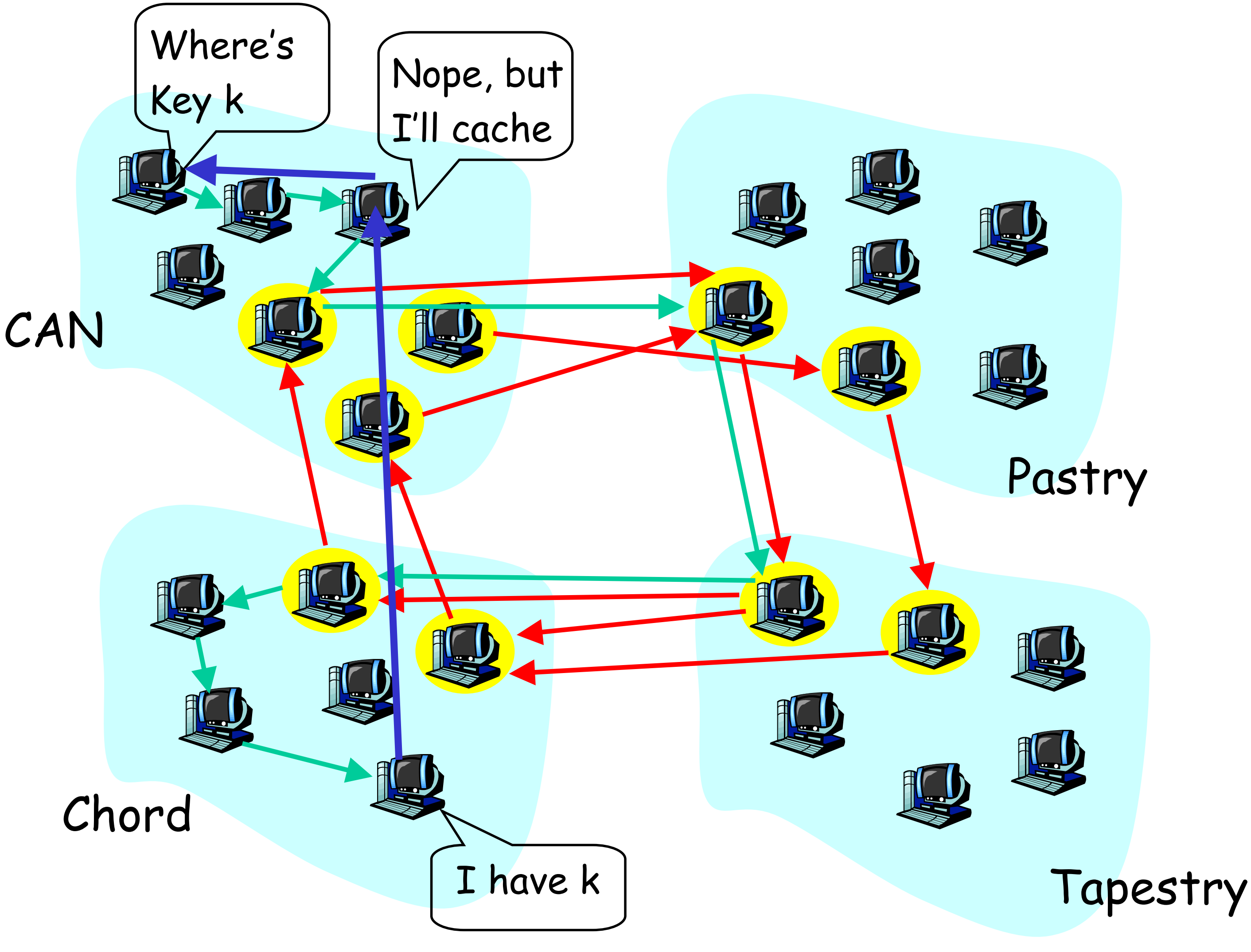
Hierarchical framework



Hierarchical Lookup



Cooperative group caching

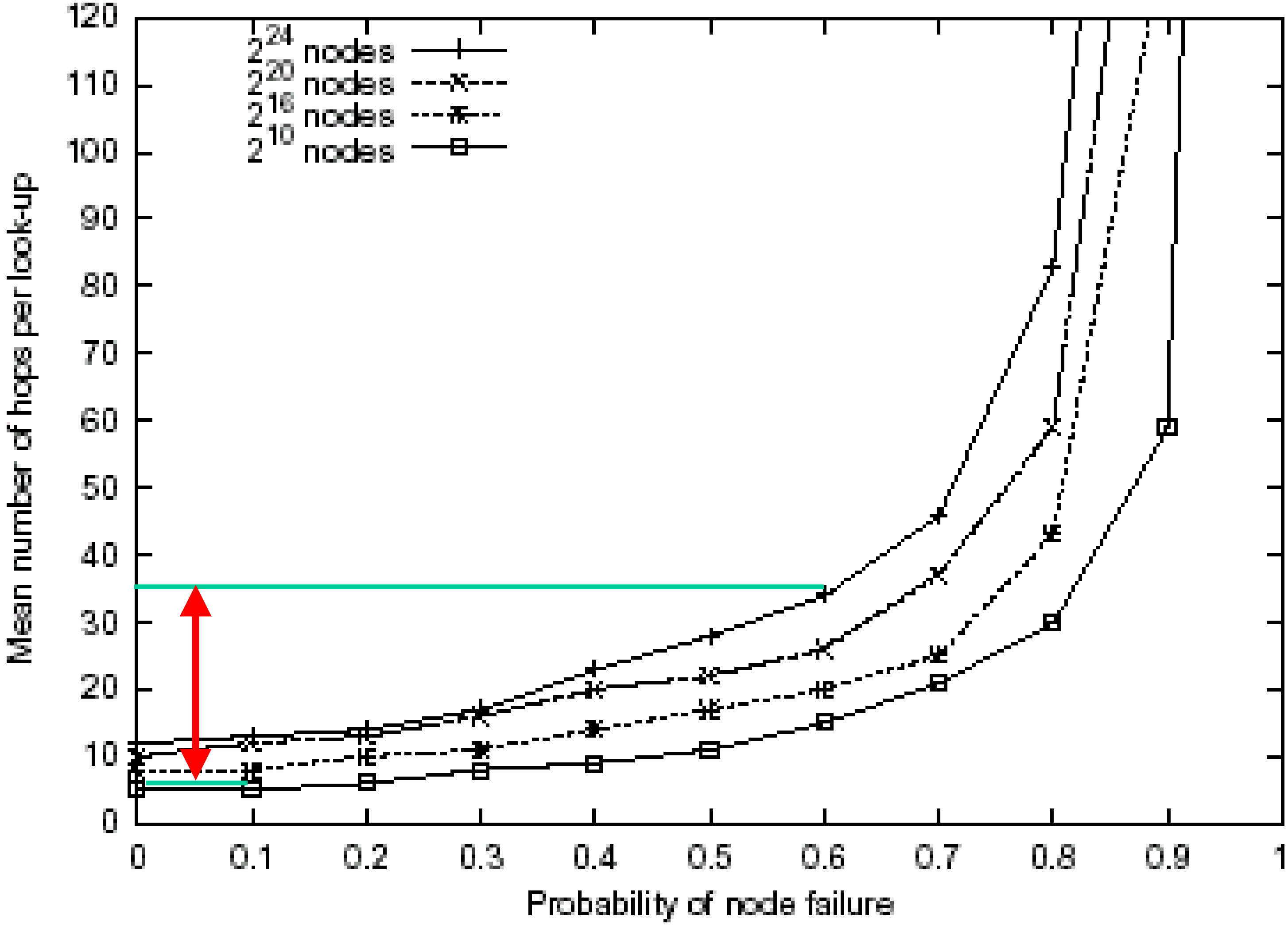


Hierarchical Lookup (2)

Benefits

- Can reduce number of hops, particularly when nodes have heterogeneous availabilities
- Groups can cooperatively cache popular files, reducing average latency
- Facilitates large scale deployment by providing administrative autonomy to groups
  - Each ISP can use it is own DHT protocol
  - Similar to intra-AS routing

Inter-lookup: Chord



3. Structured P2P: DHT Approaches

- The DHT service and API
- CARP
- Consistent Hashing
- Chord
- CAN
- Pastry/Tapestry
- Hierarchical lookup services
- Topology-centric lookup service

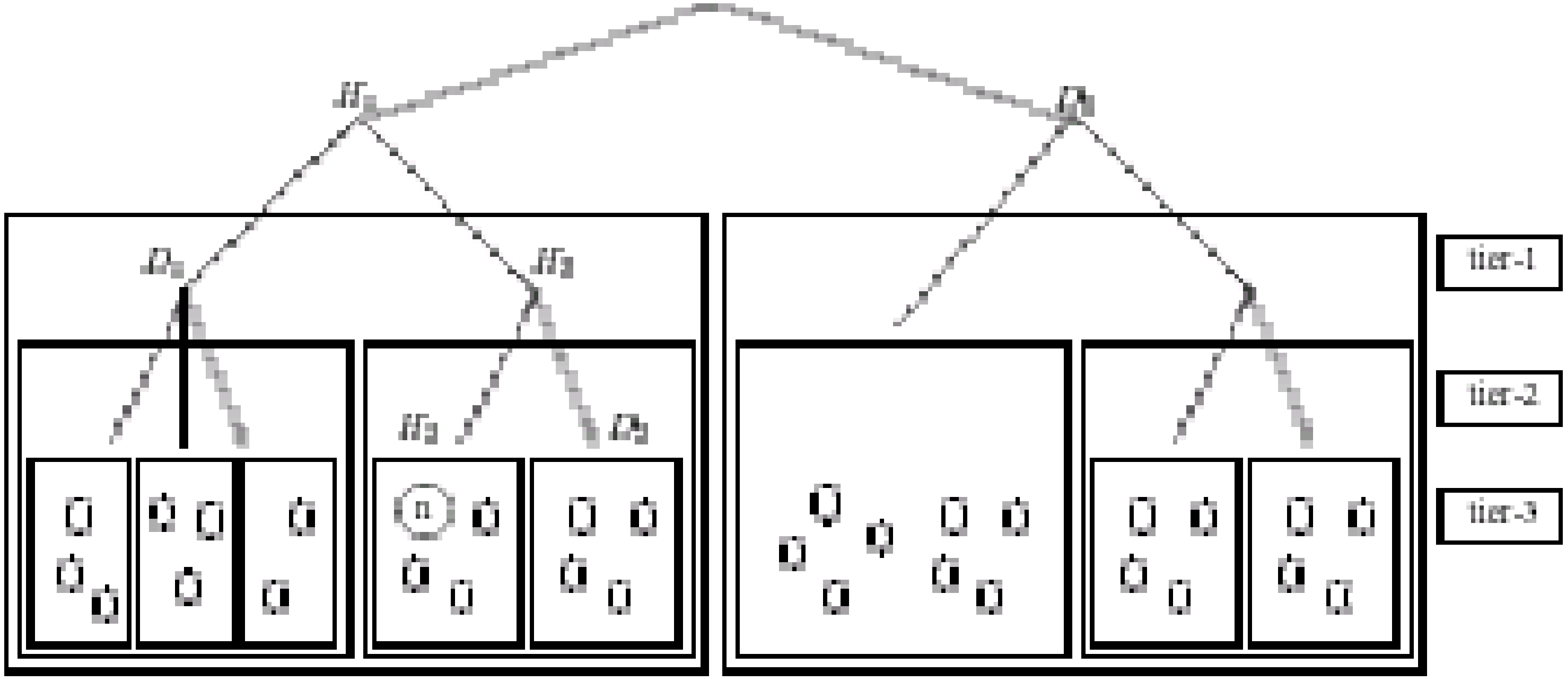
Topology Centric: TOPLUS

- Lookup delay ≈ IP delay (stretch =1)

Idea

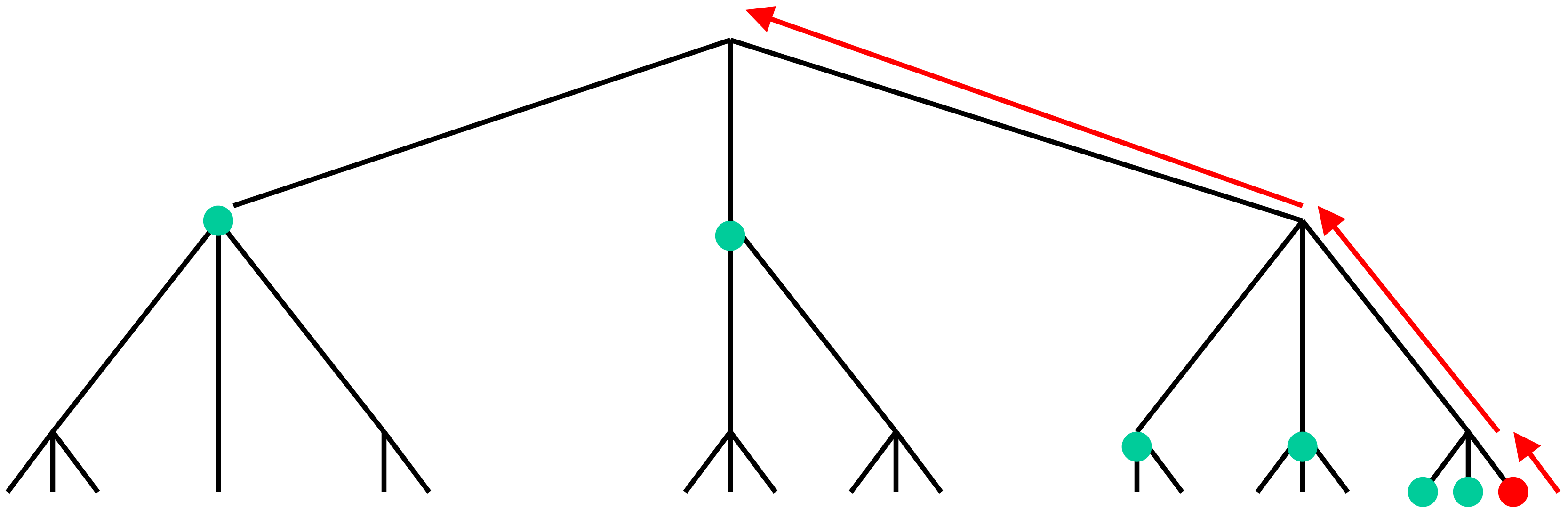
- Node ids are IP addresses
- Nested groups: unbalanced
- Get groups from prefixes in BGP routing tables
  - Each group is contiguous set of IP addresses of form w.x.y.z/n (e.g., 128.15.6/23)
  - massaging

Nested Groups



TOPLUS: Delegate Nodes

- Delegates: as you pass through groups towards root, take delegate from each descendant



TOPLUS

Node state:

- Other nodes in inner group
- "descendant" delegate

Routing:

- Longest IP-prefix match
- Use optimized techniques in IP routers
- Number of hops < H+1, H = height of tree
- Typically, big jumps made initially into destinations AS (opposite of Pastry)

TOPLUS

Caching

- A group $G = w.x.y.z/r$ agrees to be a cooperative regional cache
- When node in X in G wants to lookup k for file f, it creates k_G:
  - first r bits of k replaced with first r bits of w.x.y.z/r
- X discovers node Yin G that's responsible for k_G
- X requests f through Y

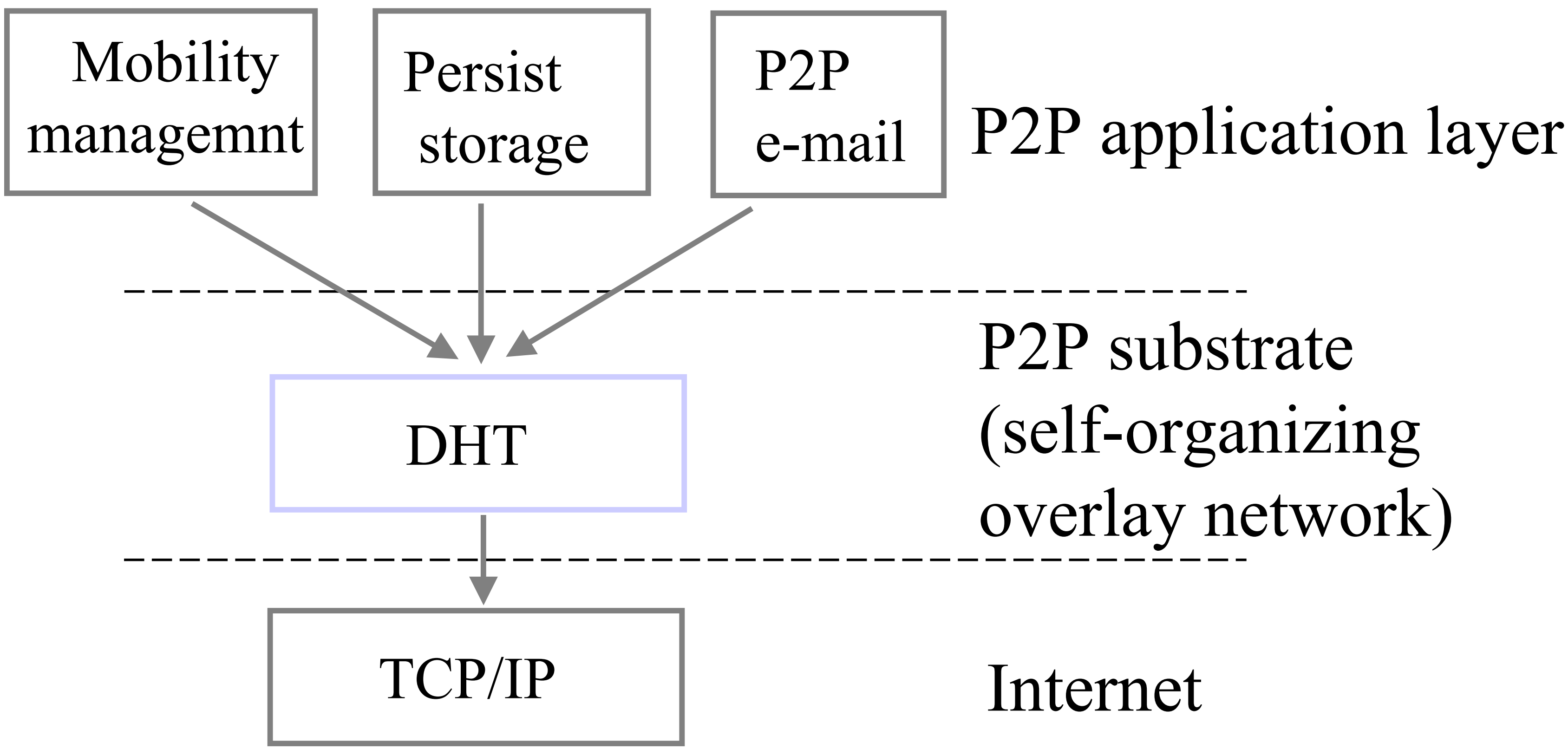
TOPLUS: some results

- 250,252 prefixes from BGP tables
  - 47,000 tier-1 groups
  - 10,000 of which have sub groups
  - 11 tiers
- Used King to estimate delay between arbitrary nodes
- Stretch: 1.17
- Aggregated groups: 8000 tier-1 groups, 40% having subgroups; stretch = 1.28

TOPLUS issues

- Inner group can be too big/small
  - Use XOR distance metric
- Non-uniform population of node id space
- Lack of virtual nodes
- Correlated node failures

4. Applications using DHTs



4. Applications using DHTs

- file sharing
  - Issues
  - Caching
  - Optimal replication theory
- persistent file storage
  - PAST
- mobility management
- SOS

File sharing using DHT

Advantages

- Always find file
- Quickly find file
- Potentially better management of resources

Challenges

- File replication for availability
- File replication for load balancing
- Keyword searches

There is at least one file sharing system using DHTs: Overnet, using Kademlia

File sharing: what's under key?

Data item is file itself

- Replicas needed for availability
- How to load balance?

Data item under key is list of pointers to file

- Must replicate pointer file
- Must maintain pointer files: consistency

File sharing: keywords

- Recall that unstructured file sharing provides keyword search
  - Each stored file has associated metadata, matched with queries
- DHT: Suppose key = h(artist, song)
  - If you know artist/song exactly, DHT can find node responsible for key
  - Have to get spelling/syntax right!
- Suppose you only know song title, or only artist name?

Keywords: how might it be done?

Each file has XML descriptor

```
<song>
<artist>David
  Bowie</artist>
<title>Changes</title>
<album>Hunky Dory</album>
<size>3156354</size>
</song>
```

Key is hash of descriptor: k = h(d)

Store file at node responsible for k

Plausible queries

q_1 = /song[artist/David Bowie][title/Changes] [album/Hunky Dory] [size/3156354]

q_2 = /song[artist/David Bowie][title/Changes]

q_3 = /song/artist/David Bowie

q_4 = /song/title/Changes

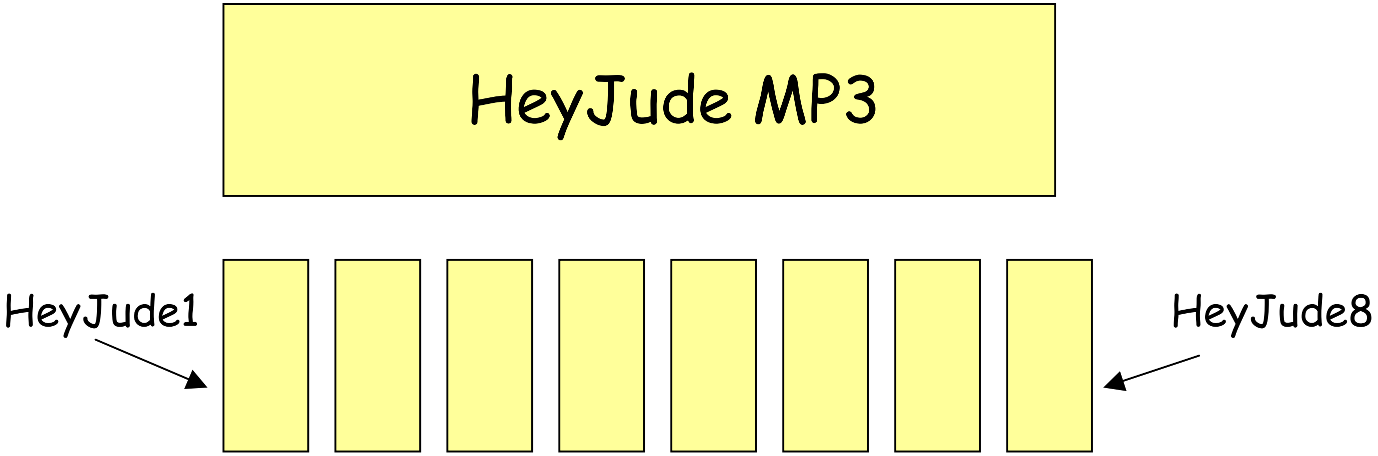
Create keys for each plausible query: $k_n = h(q_n)$

For each query key k_n, store descriptors d at node responsible for k_n

Keywords: continued

- Suppose you input `q4 = /song/title/Changes`
- Locally obtain key for q_4, submit key to DHT
- DHT returns node n responsible for q_4
- Obtain from n the descriptors of all songs called Changes
- You choose your song with descriptor d, locally obtain key for d, submit key to DHT
- DHT returns node n' responsible for desired song

Blocks



Each block is assigned to a different node

Blocks (2)

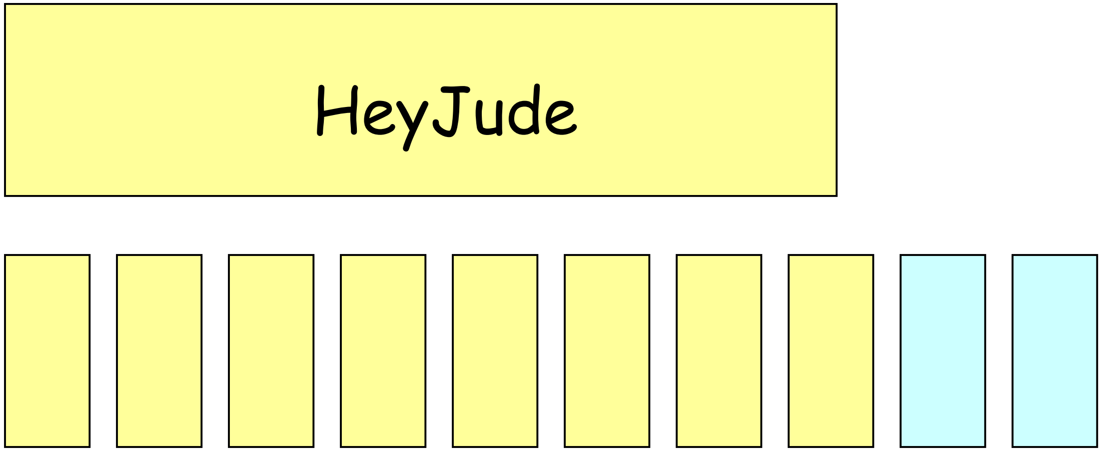
Benefits

- Parallel downloading
  - Without wasting global storage
- Load balancing
  - Transfer load for popular files distributed over multiple nodes

Drawbacks

- Must locate all blocks
- Must reassemble blocks
- More TCP connections
- If one block is unavailable, file is unavailable

Erasures (1)



- Reconstruct file with any m of r pieces
- Increases storage overhead by factor r/m

Erasures (2)

Benefits

- Parallel downloading
  - Can stop when you get the first m pieces
- Load balancing
- More efficient copies of blocks
  - Improved availability for same amount of global storage

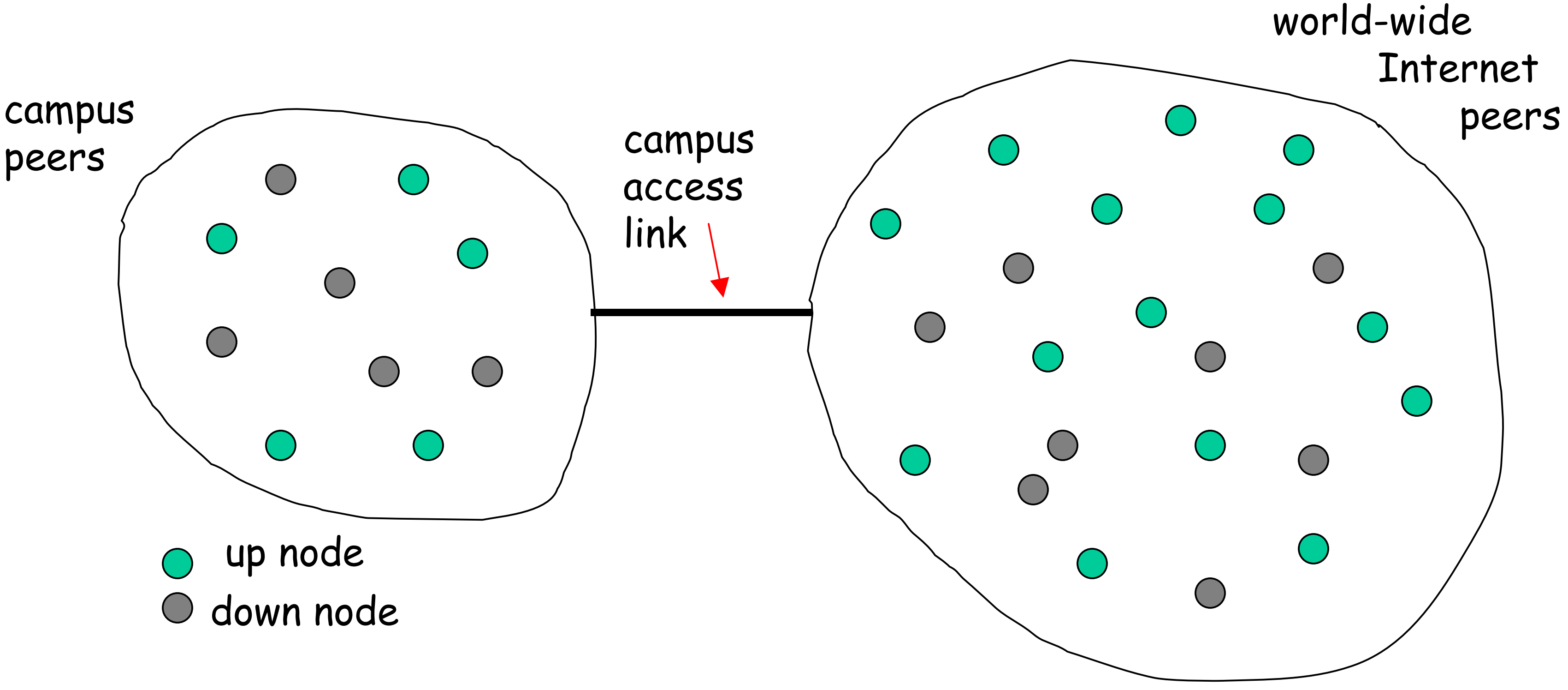
Drawbacks

- Must reassemble blocks
- More TCP connections

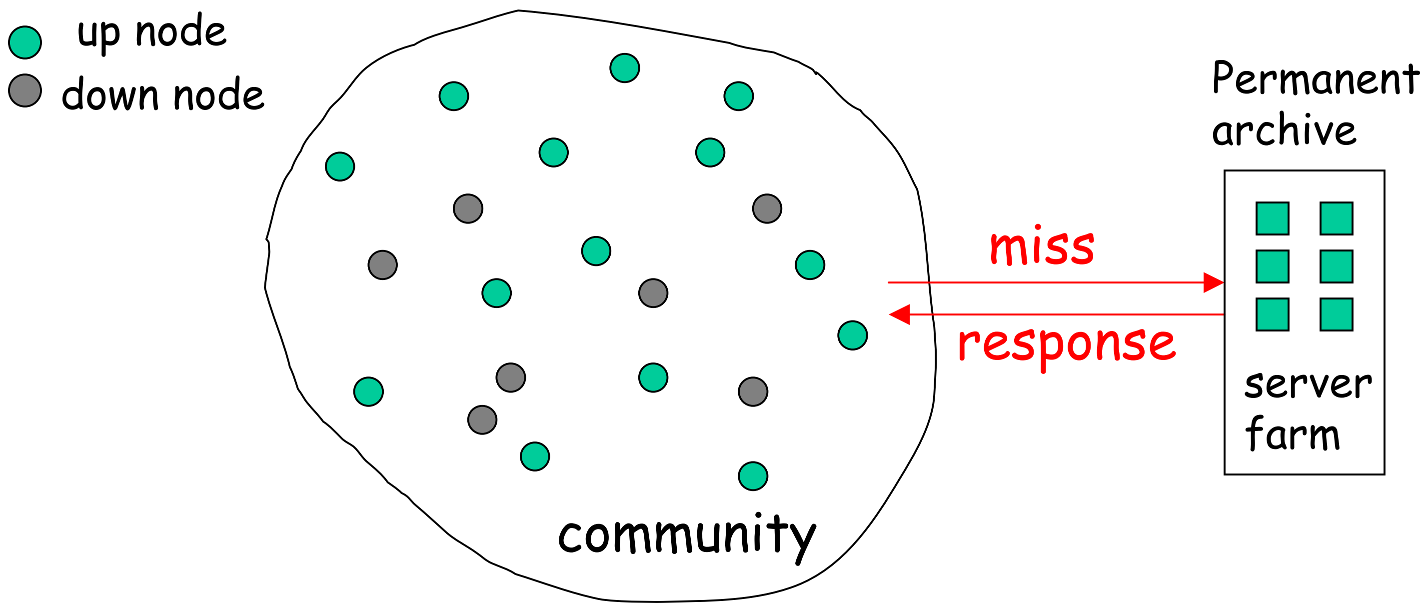
4. Applications for DHTs

- file sharing
  - Issues
  - Caching
  - Optimal replication theory
- persistent file storage
  - PAST
- mobility management
  - I3
- SOS

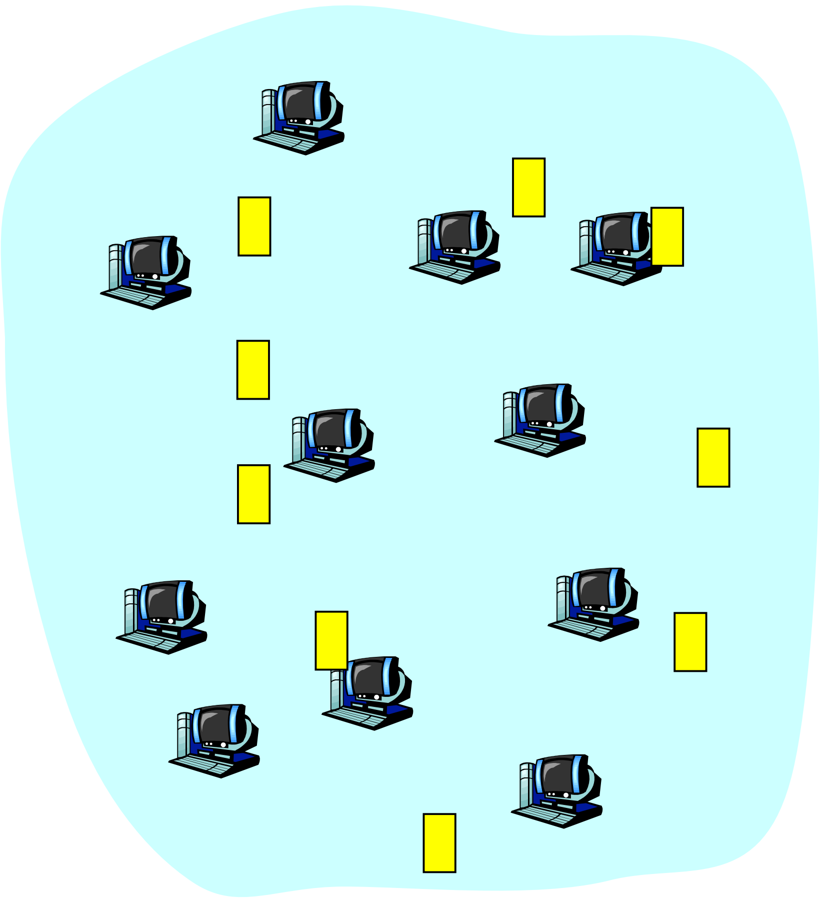
Cache: campus community



Cache: distribution engine



Replication in cache: The problem



- Lot's of nodes
  - Which go up & down
- Lot's of files
  - Some more popular than others
  - Popularities changing
- How many copies of each file? Where?
- Want to optimize availability

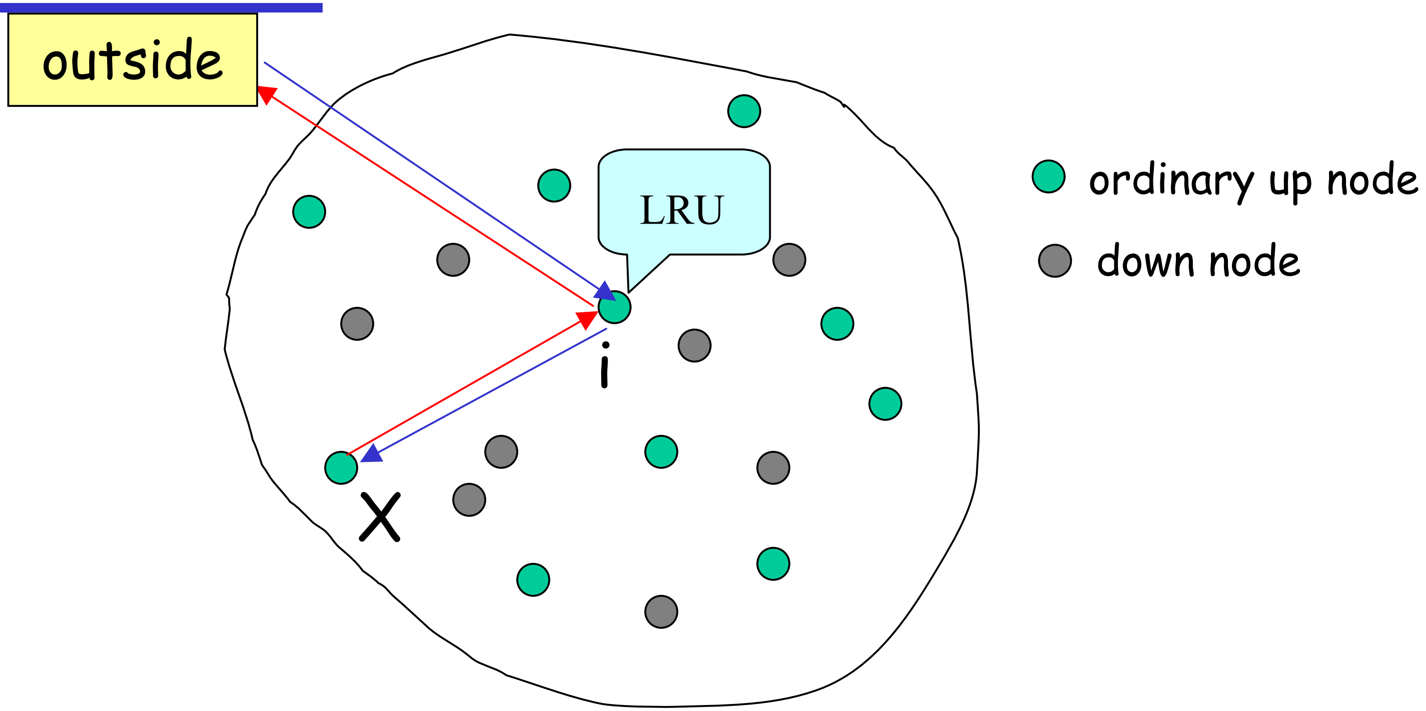
DHT: Recall

- For a given key, DHT returns current up node responsible for key
- Can extend API to provide 1st place winner, 2nd place winner, etc.

Desirable properties of content management algorithm

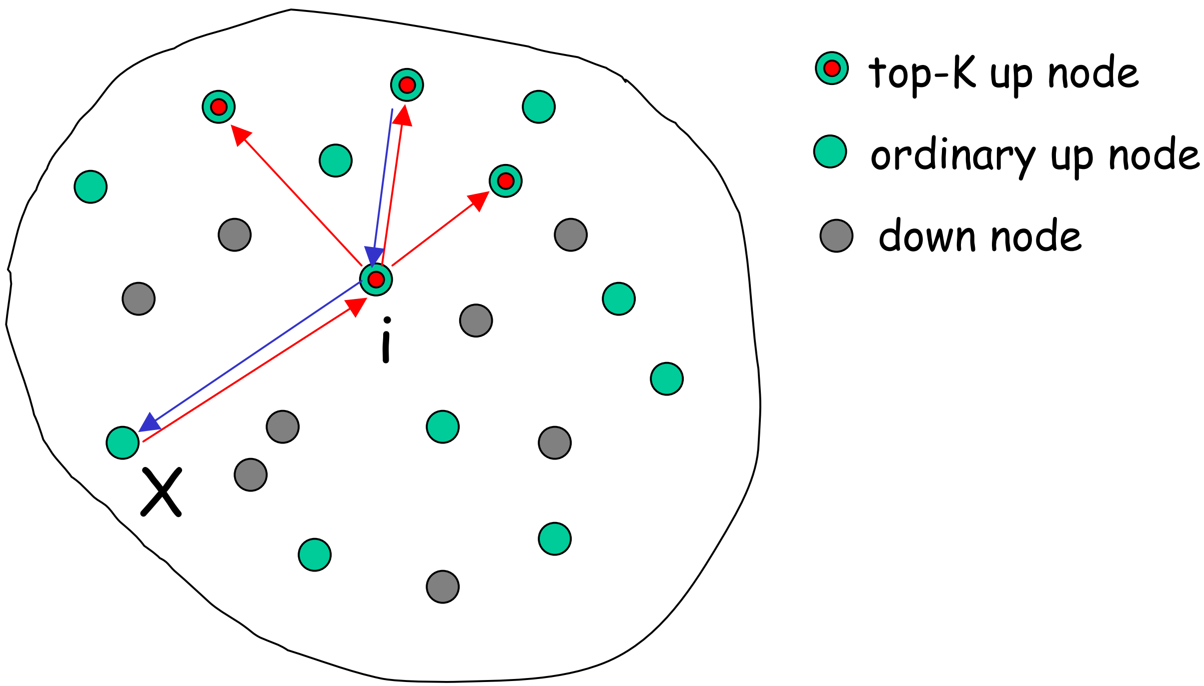
- Distributed
- Adaptive
  - No a priori knowledge about file popularities, node availabilities
  - New files inserted continually
- High hit rate performance
- Replicate while satisfying requests

Adaptive algorithm: simple version



Problem: Can miss even though object is in an up node in the community

Top-K algorithm

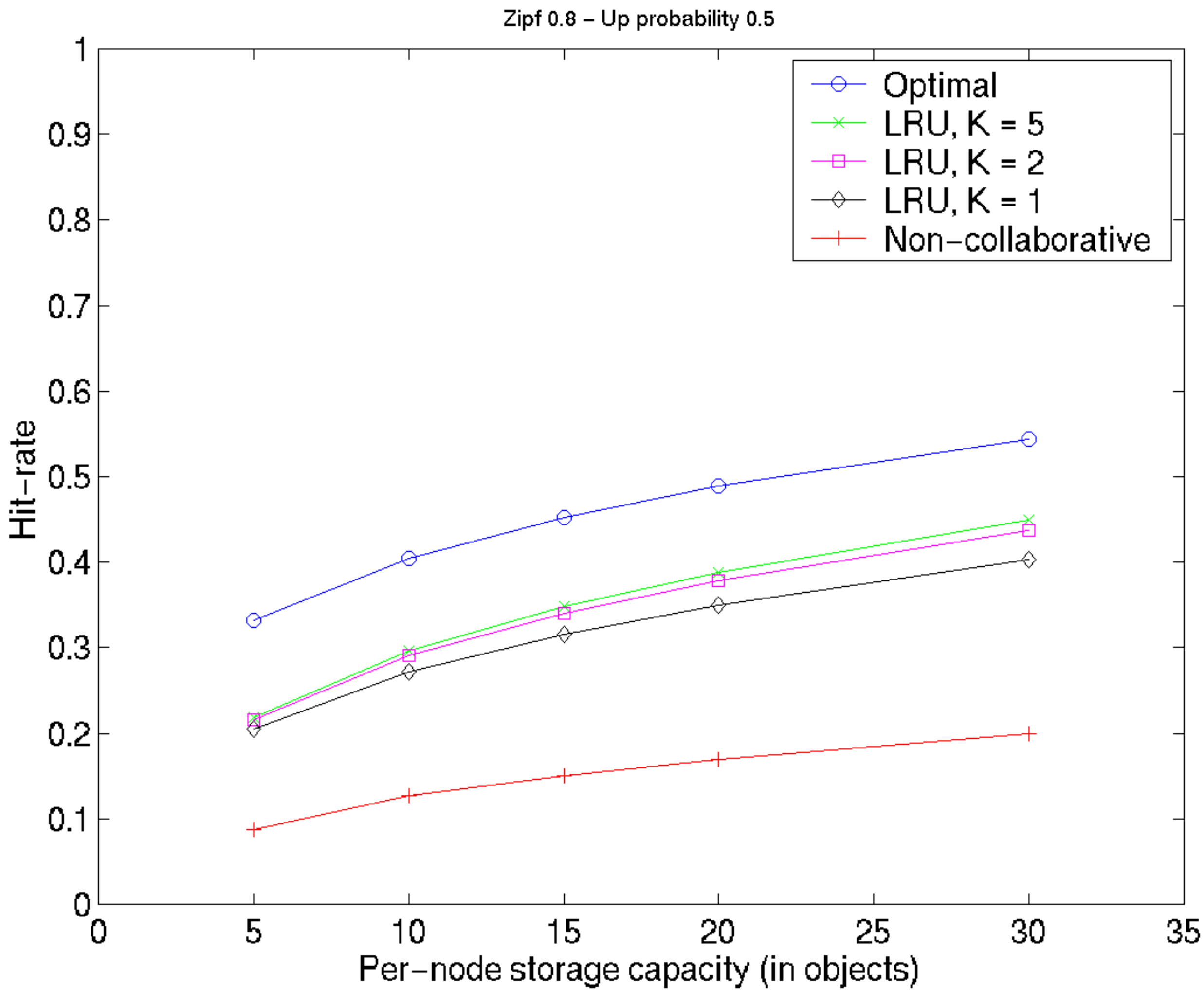


- If i doesn't have o, i pings top-K winners.
- i retrieves o from one of the top K if present.
- If none of the top K has o, i retrieves o from outside.

Simulation

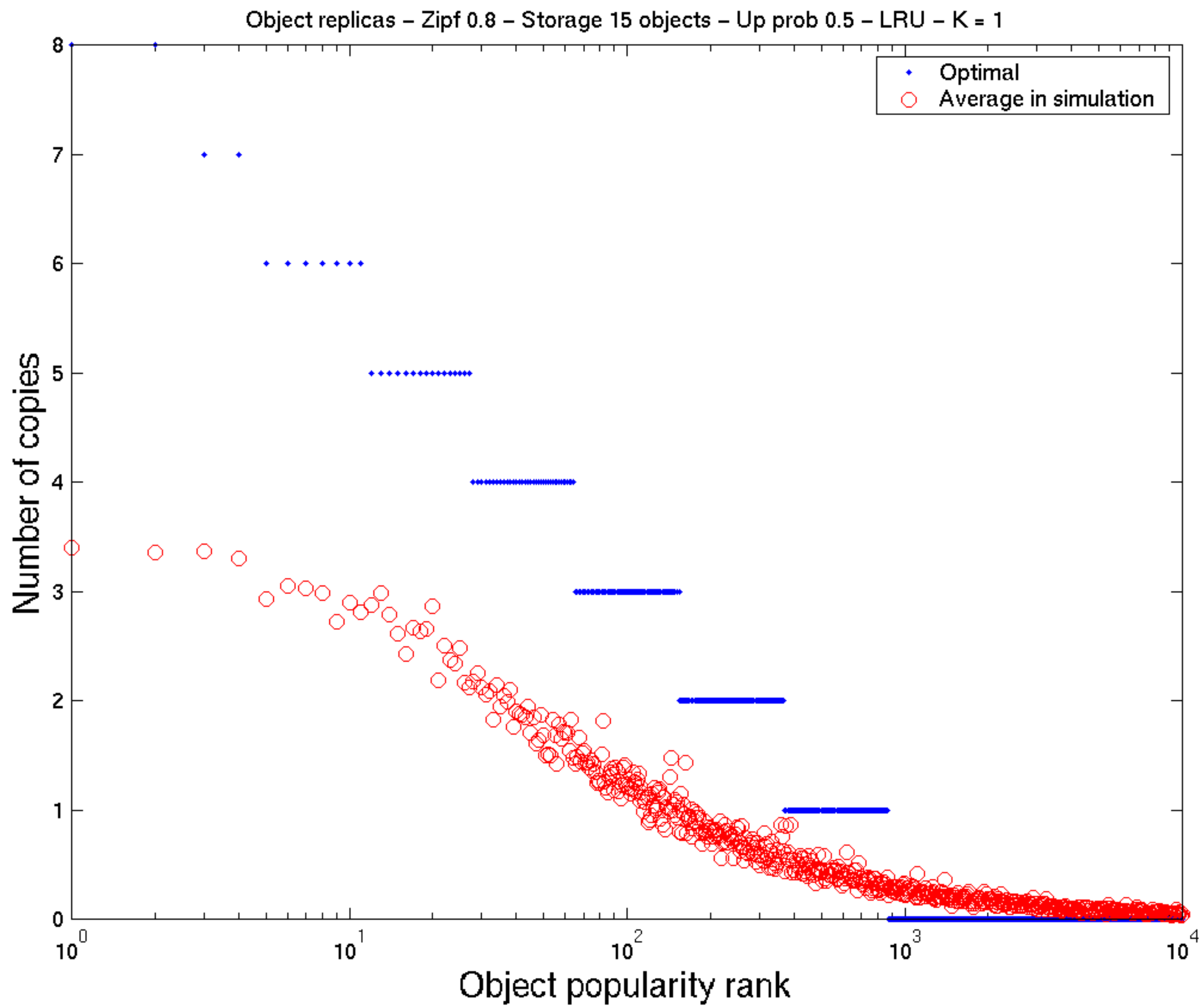
- Adaptive and optimal algorithms
- 100 nodes, 10,000 objects
- Zipf = 0.8, 1.2
- Storage capacity 5-30 objects/node
- All objects the same size
- Up probs 0.2, 0.5, and 0.9
- Top K with K = {1,2, 5}

Hit-probability vs. node storage



p = P(up)
= .5

Number of replicas



p = P(up)
= .5

15 objects per node

K = 1

Most frequently requested (MFR)

- Each peer estimates local request rate for each object.
  - denote $\lambda_o(i)$ for rate at peer i for object o
- Peer only stores the most requested objects.
  - packs as many objects as possible

Suppose i receives a request for o:

- i updates $\lambda_o(i)$
- If i doesn't have o & MFR says it should:

  i retrieves o from the outside

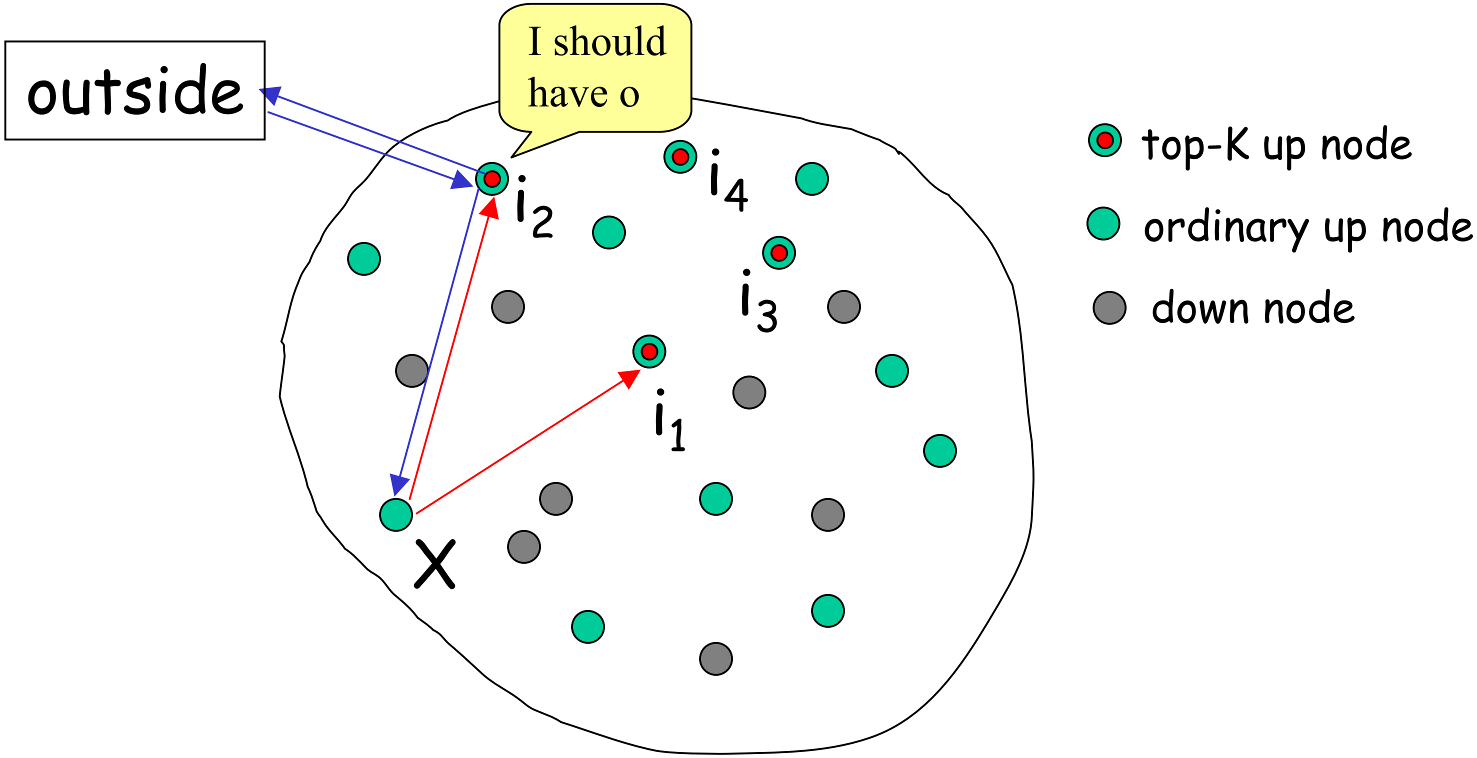
Influence the rates

- $\lambda_o(i)$ should reflect the "place" of node i

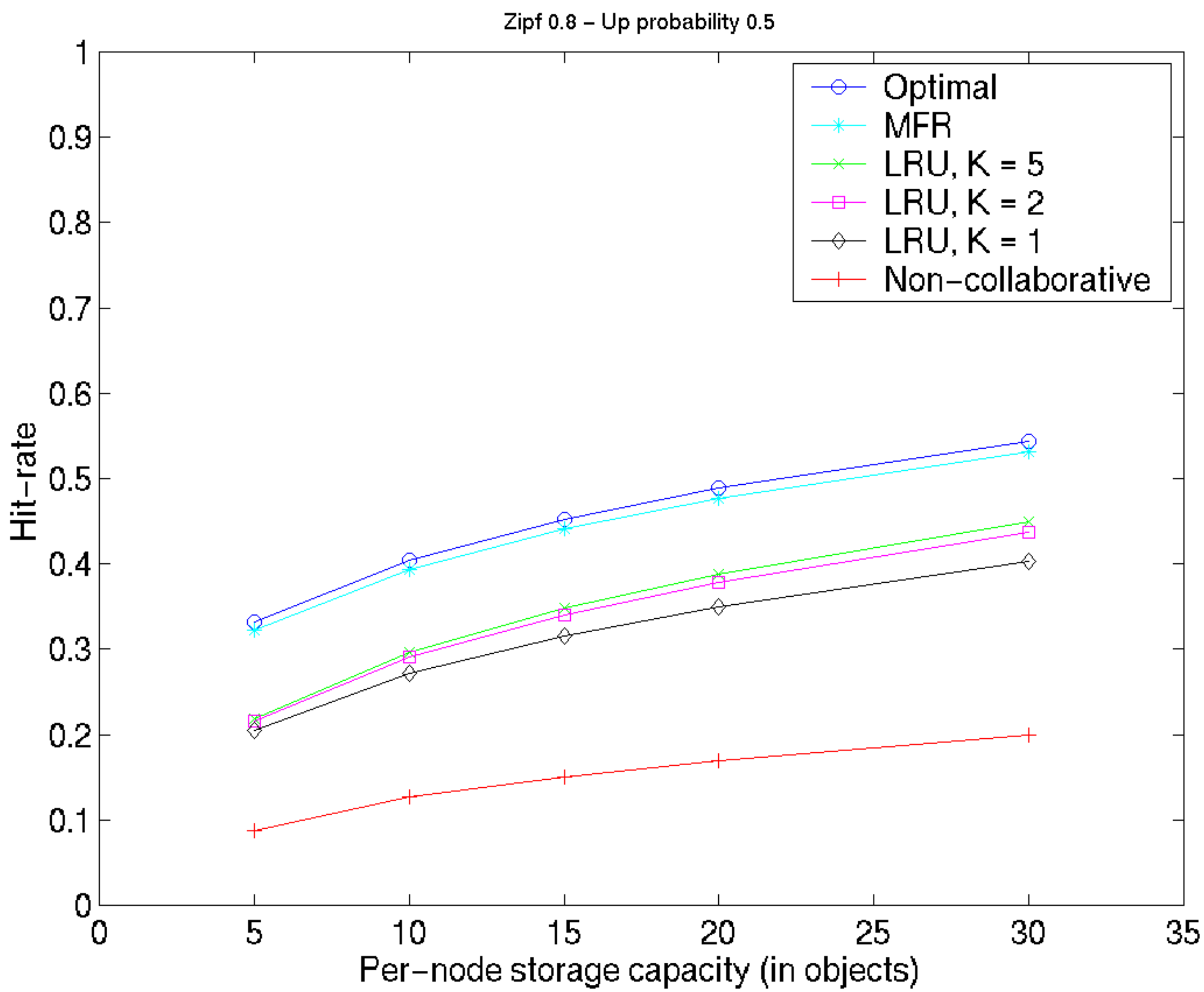
  ➡ Don't request o from top-K in parallel
  ➡ Instead, sequentially request

- Resulting $\lambda_o(i)$ are thinned by previous winners

Most-frequently-requested top-K algorithm



Hit-probability vs. node storage



p = P(up)
= .5

MFR: K=5

Top-I MFR vs. optimal

- Perf of Top-I MFR can be evaluated by a reduced-load iterative procedure
- 30 cases, each with 100 nodes & $b_j = 1$
- 28 out of 30 cases:
  - Top-I MFR converges to optimal!
- But there are counter examples

Summary: MFR top-K algorithm

Implementation

- layers on top of location substrate
- decentralized
- simple: each peer keeps track of a local MFR table

Performance

- provides near-optimal replica profile

4. Applications for DHTs

- file sharing
  - Issues
  - Caching
  - Optimal replication theory
- persistent file storage
  - PAST
- mobility management
- SOS

Optimization theory

- J objects, I peers in community
- object j
  - requested with probability q_j
  - size b_j
- peer i
  - up with probability p_i
  - storage capacity S_i
- decision variable
  - x_{ij} = 1 if a replica of j is put in i; 0 otherwise
- Goal: maximize hit probability (availability)

Optimization problem

Minimize $$\sum_{j=1}^{J} q_j \prod_{i=1}^{I} (1-p_i)^{x_{ij}}$$

subject to $$\sum_{j=1}^{J} b_j x_{ij} \le S_i, \quad i = 1,\ldots,I$$

$$x_{ij} \in \{0,1\}, \quad i = 1,\ldots,I, \quad j = 1,\ldots,J$$

Special case of Integer programming problem: NP

Homogeneous up probabilities

Suppose $p_i = p$

Let $n_j = \sum_{i=1}^{I} x_{ij}$ = number of replicas of object *j*

Let S = total group storage capacity

Minimize $\sum_{j=1}^{J} q_j (1-p)^{n_j}$

subject to: $\sum_{j=1}^{J} b_j n_j \le S$

Can be solved by dynamic programming

Erasures

- Each object consists of R_j erasure packets
- Need M_j erasure packets to reconstruct object
- Size of erasure packet is b_j/M_j

- Potentially improves hit probability
- Potentially abate hot-spot problem

Continuous optimization

$$f_j(z) = q_j \sum_{m=M_j}^{R_j} \binom{R_j}{m} [1-(1-p)^{c_j z}]^m [(1-p)^{c_j z}]^{R_j - m} \qquad c_j = M_j / b_j R_j$$

Theorem: Following optimization problem provides upper bound on hit probability:

Minimize $\sum_{j=1}^{J} f_j(z_j)$

subject to $\displaystyle\sum_{j=1}^{J} z_j = S \quad z_j \geq 0,\ j = 1, \ldots, J$

Easy to solve!

Steps in proof:

- Allow for continuous number of copies of each erasure packet
- Allocate space z_j to each object
- For given object j, use Shur convexity to show that optimal solution achieved when there's the same number of copies of each of the R_j erasure packets
- Optimize over possible z_j allocations

No erasures

(1) Order objects according to q_j/b_j, largest to smallest

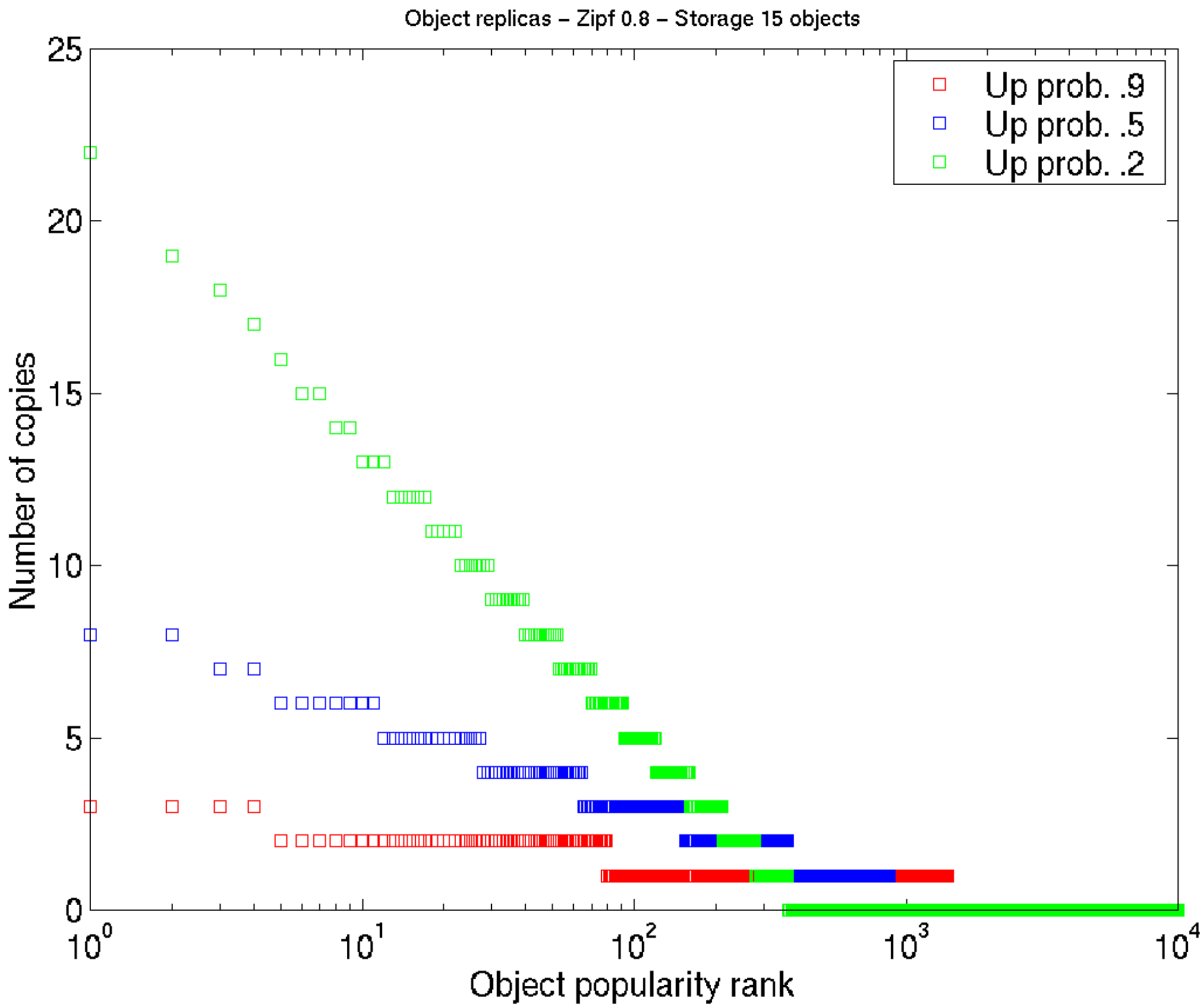
(2) There is an L such that $n_j^* = 0$ for all j > L.

(3) For j <= L , "logarithmic assignment rule":

$$n_j^* = \frac{S}{B_L} + \frac{\sum_{l=1}^{L} b_l \ln(q_l / b_l)}{B_L \ln\left(1 - p\right)} + \frac{\ln(q_j / b_j)}{\ln(1/(1-p))}$$

$$= K_1 + K_2 \ln(q_j / b_j)$$

Logarithmic behavior



- up prob = .9
- up prob = .5
- up prob = .2

4. Applications for DHTs

- file sharing
  - Issues
  - Caching
  - Optimal replication theory
- persistent file storage
  - PAST
- mobility management
- SOS

Persistent file storage

- PAST layered on Pastry
- CFS layered on Chord

P2P Filesystems

- Oceanstore
- FarSite

PAST: persistence file storage

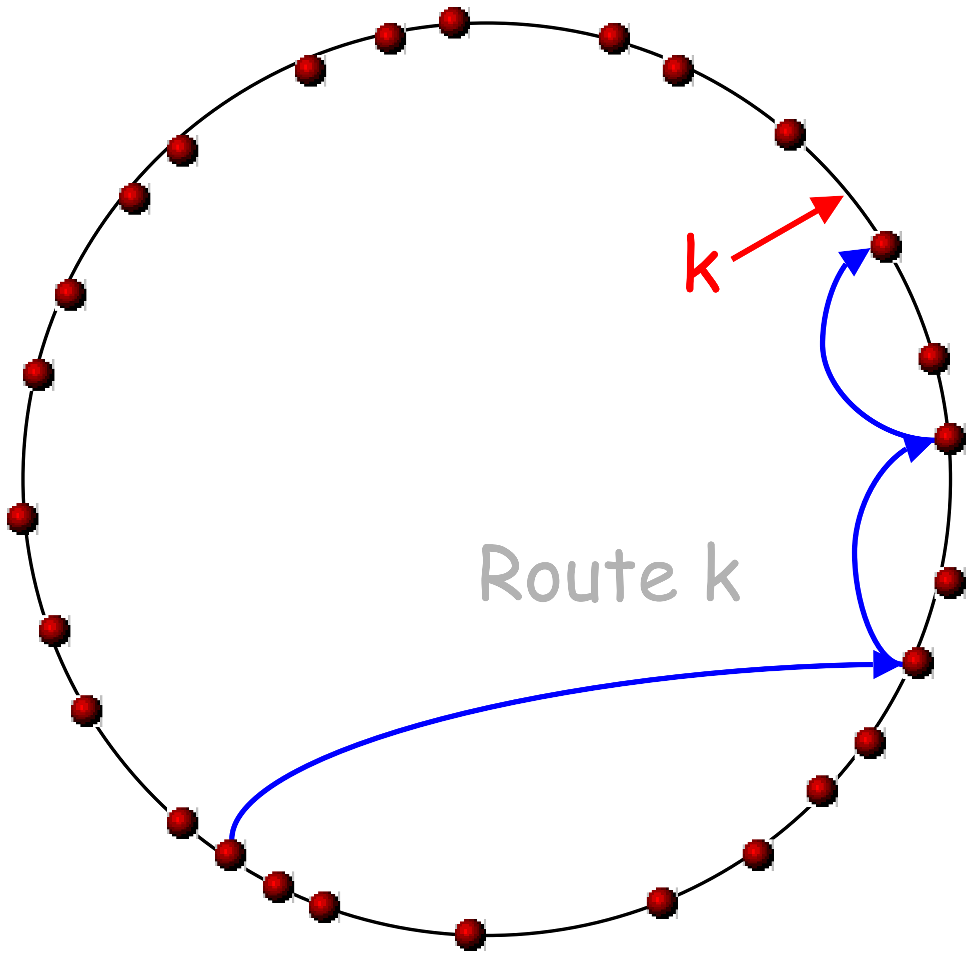
Goals

- Strong persistence
- High availability
- Scalability
  - nodes, files, queries, users
- Efficient use of pooled resources

Benefits

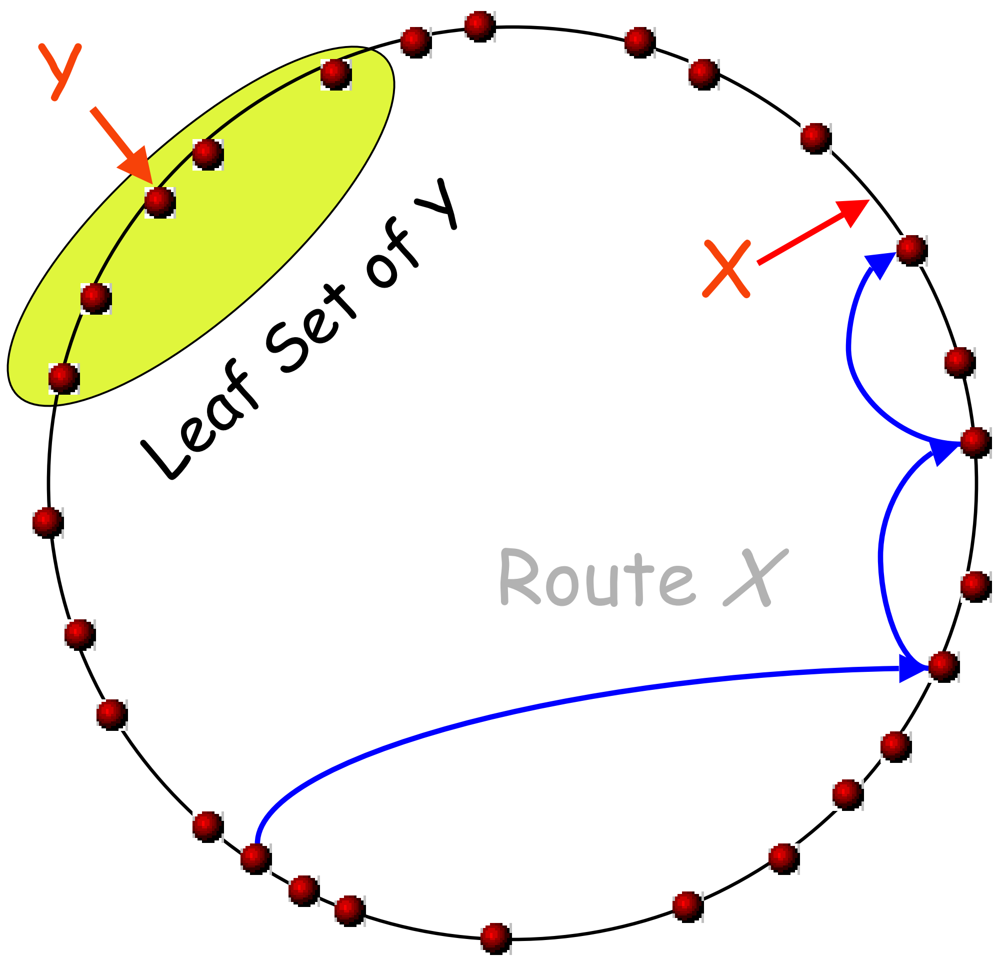
- Provides powerful backup and archiving service
- Obviates need for explicit mirroring

Pastry: A self-organizing P2P overlay network



Msg with key k is routed to live node with nodeId closest to k

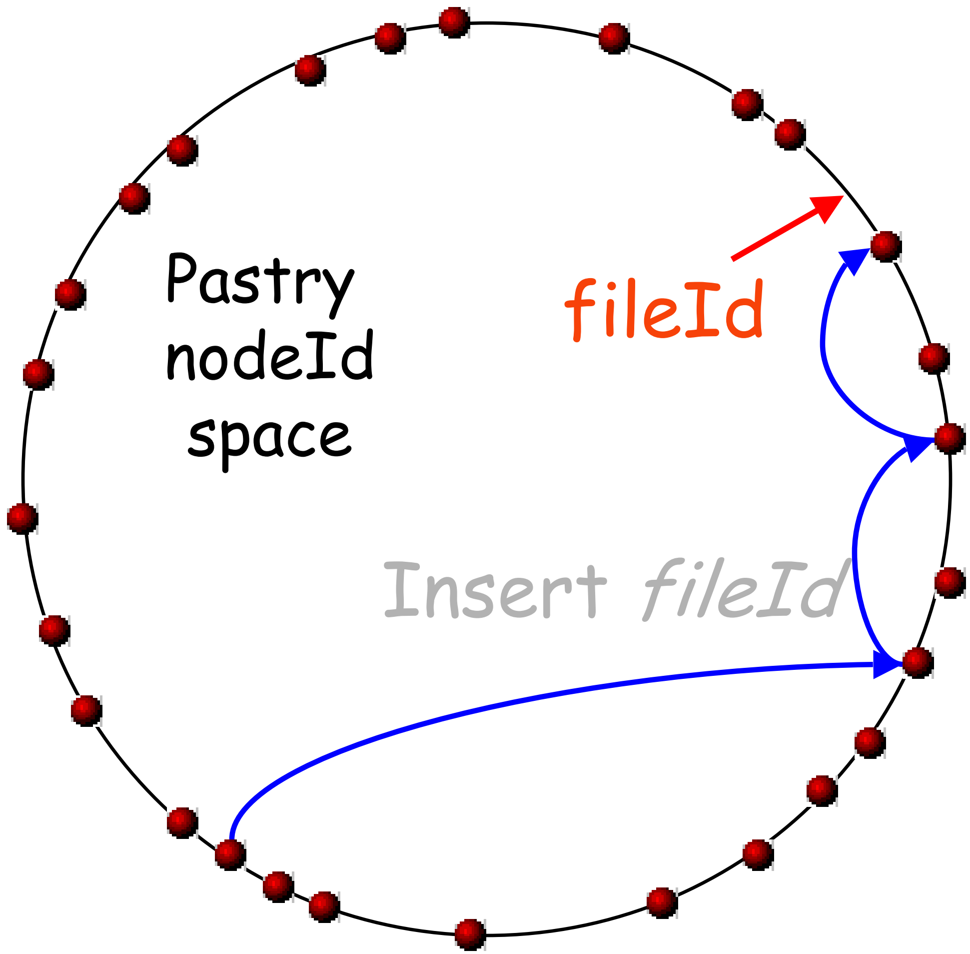
Pastry: Properties



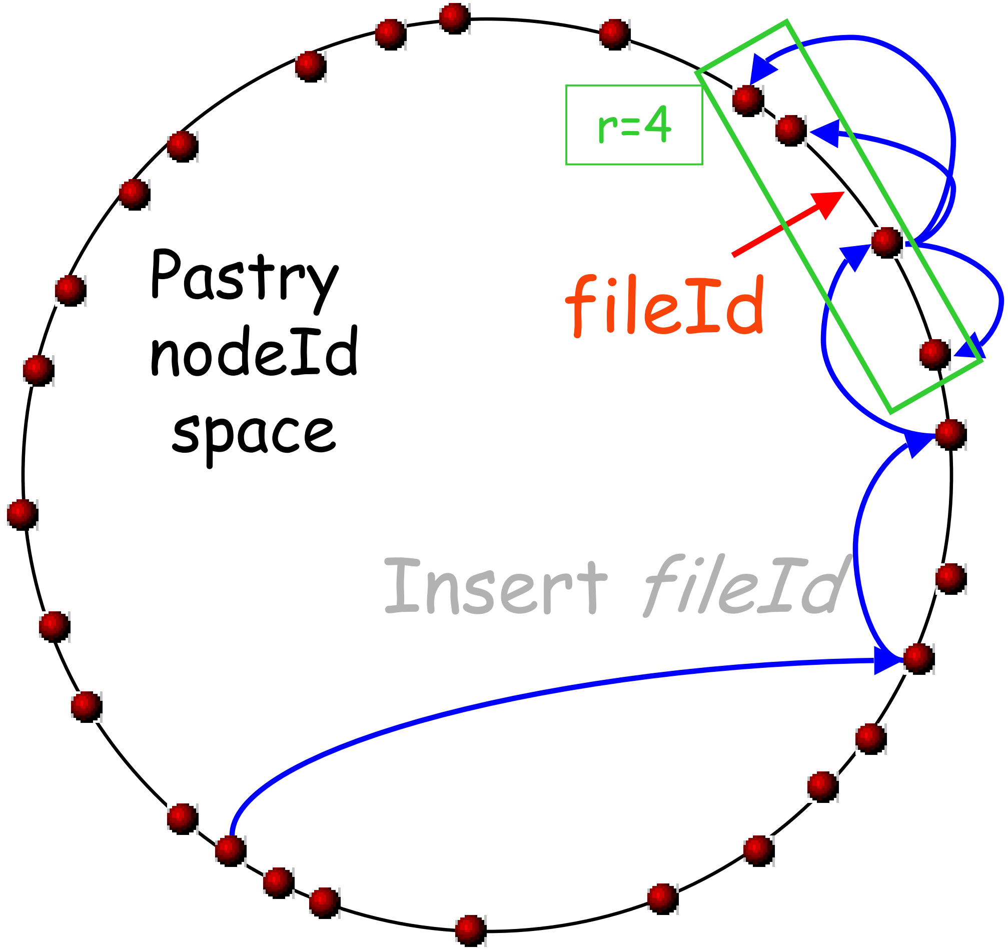
Properties

- $\log_{16}$ N steps
- O(log N) state
- leaf sets
  - diversity
- network locality

PAST: File storage



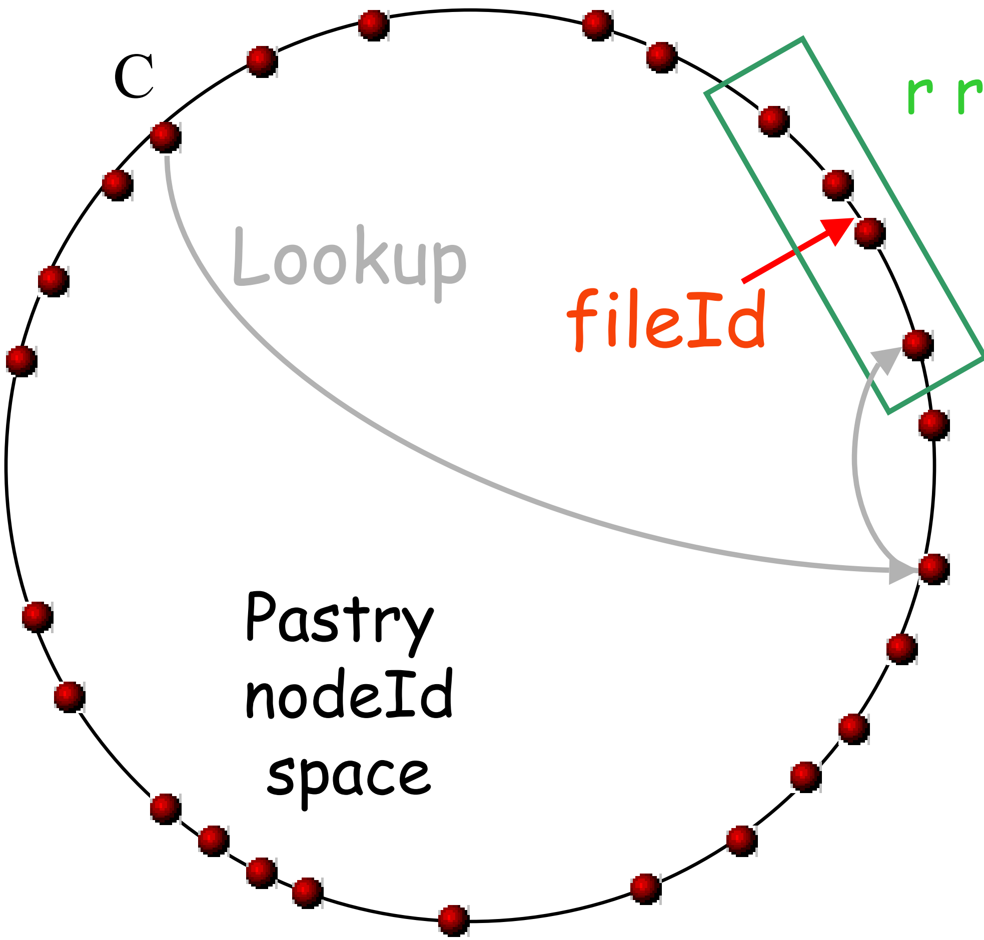
PAST: File storage



Storage Invariant:
File "replicas" are stored on r nodes with nodeIds closest to fileId

(r is bounded by the leaf set size)

PAST: File Retrieval



- file located in $\log_{16}$ N steps
- usually locates replica nearest client C

Maintaining the storage invariant

Pastry maintains leaf set membership notifies PAST of changes

- Node arrival
  - refer to existing replicas ("replica diversion")
  - lazy fetch
- Node failure
  - re-replicate file on k closest nodeIds

4. Applications for DHTs

- file sharing
  - Issues
  - Caching
  - Optimal replication theory
- persistent file storage
  - PAST
- mobility management
- SOS

Mobility management

- Alice wants to contact bob smith
  - Instant messaging
  - IP telephony
- But what is bob's current IP address?
  - DHCP
  - Switching devices
  - Moving to new domains

Mobility Management (2)

- Bob has a unique identifier:
  - bob.smith@foo.com
  - k =h(bob.smith@foo.com)
- Closest DHT nodes are responsible for k
- Bob periodically updates those nodes with his current IP address
- When Alice wants Bob's IP address, she sends query with k =h(bob.smith@foo.com)

Mobility management (3)

- Obviates need for SIP servers/registrars
- Can apply the same idea to DNS
- Can apply the same idea to any directory service
  - e.g., P2P search engines

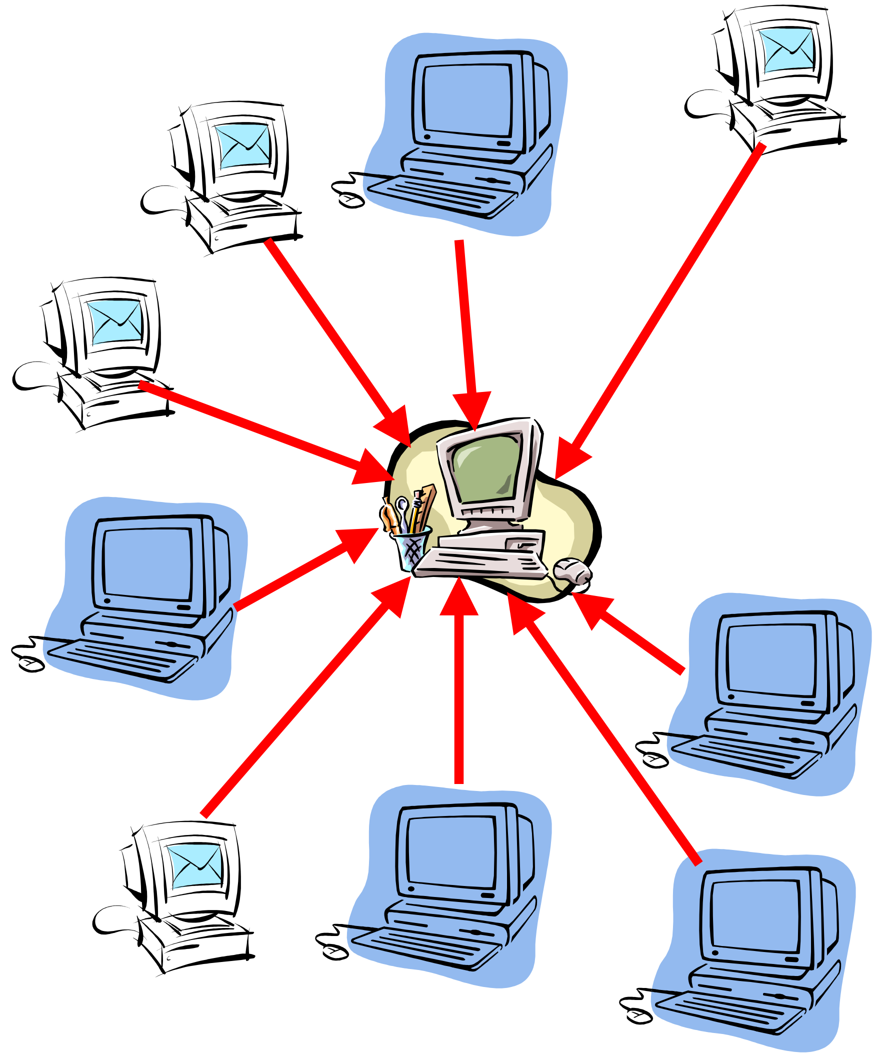
4. Applications for DHTs

- file sharing
  - Issues
  - Caching
  - Optimal replication theory
- persistent file storage
  - PAST
- mobility management
- SOS

SOS: Preventing DoS Attacks

To perform a DoS Attack:

1. Select Target to attack
2. Break into accounts (around the network)
3. Have these accounts send packets toward the target
4. Optional: Attacker "spoofs" source address (origin of attacking packets)



Goals of SOS

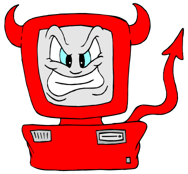
- Allow moderate number of legitimate users to communicate with a target destination, where
  - DoS attackers will attempt to stop communication to the target
  - target difficult to replicate (e.g., info highly dynamic)
  - legitimate users may be mobile (source IP address may change)
- Example scenarios
  - FBI/Police/Fire personnel in the field communicating with their agency's database
  - Bank users' access to their banking records
  - On-line customer completing a transaction

SOS: The Players

- Target: the node/end-system/server to be protected from DOS attacks

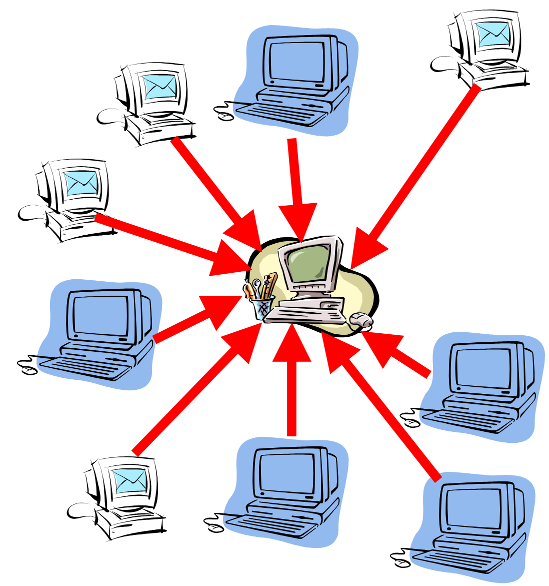
- Legitimate (Good) User: node/end-system/user that is authenticated (in advance) to communicate with the target

- Attacker (Bad User): node/end-system/user that wishes to prevent legitimate users' access to targets



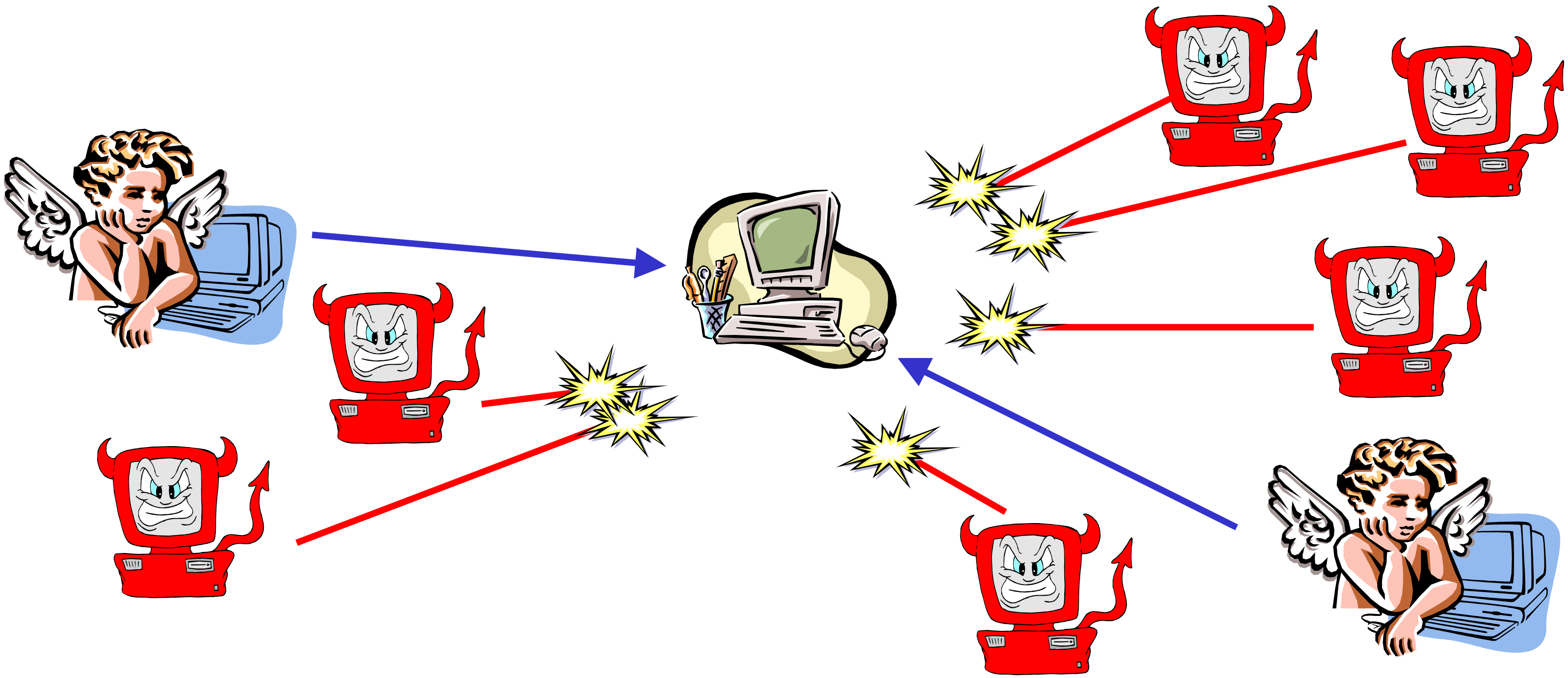
SOS: The Basic Idea

- DoS Attacks are effective because of their many-to-one nature: many attack one
- SOS Idea: Send traffic across an overlay:
  - Force attackers to attack many overlay points to mount successful attack
  - Allow network to adapt quickly: the "many" that must be attacked can be changed



Goal

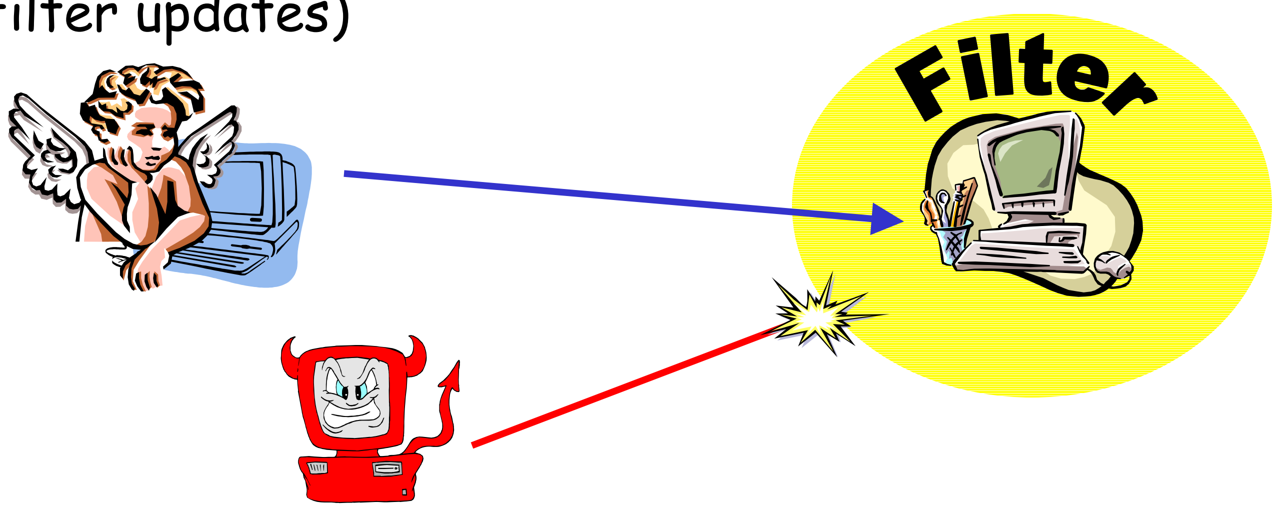
- Allow pre-approved legitimate users to communicate with a target
- Prevent illegitimate attackers' packets from reaching the target
- Want a solution that
  - is easy to distribute: doesn't require mods in all network routers
  - does not require high complexity (e.g., crypto) ops at/near the target



Assumption: Attacker cannot deny service to core network routers and can only simultaneously attack a bounded number of distributed end-systems

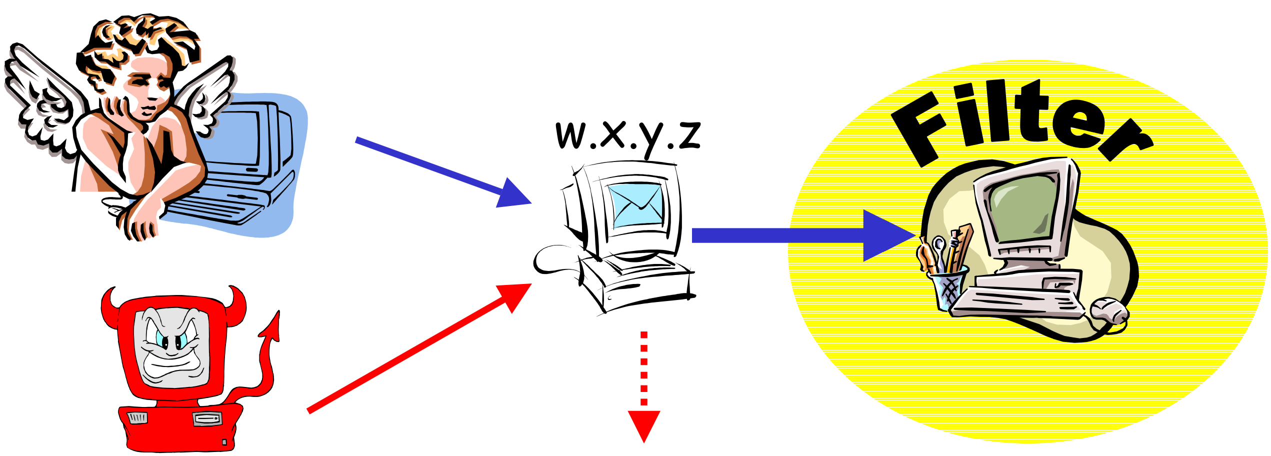
SOS: Step 1 - Filtering

- Routers "near" the target apply simple packet filter based on IP address
  - legitimate users' IP addresses allowed through
  - illegitimate users' IP addresses aren't
- Problems: What if
  - good and bad users have same IP address?
  - bad users know good user's IP address and spoofs?
  - good IP address changes frequently (mobility)? (frequent filter updates)



SOS: Step 2 - Proxies

- Step 2: Install Proxies outside the filter whose IP addresses are permitted through the filter
  - proxy only lets verified packets from legitimate sources through the filter

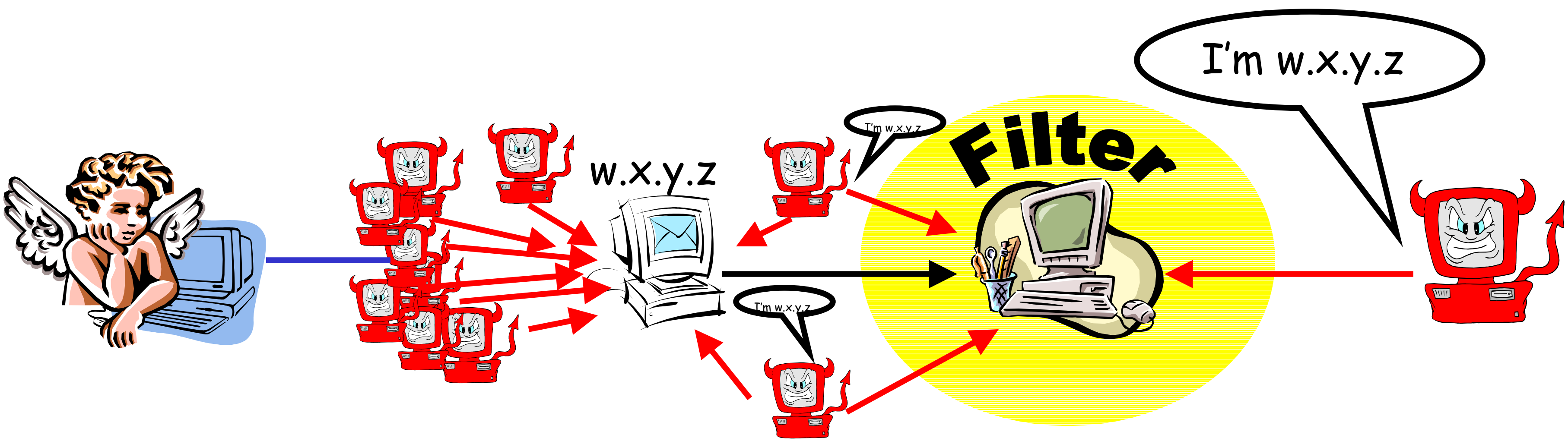


not
done
yet...

Problems with a known Proxy

Proxies introduce other problems

- Attacker can breach filter by attacking with spoofed proxy address
- Attacker can DoS attack the proxy, again preventing legitimate user communication



SOS: Step 3 - Secret Servlets

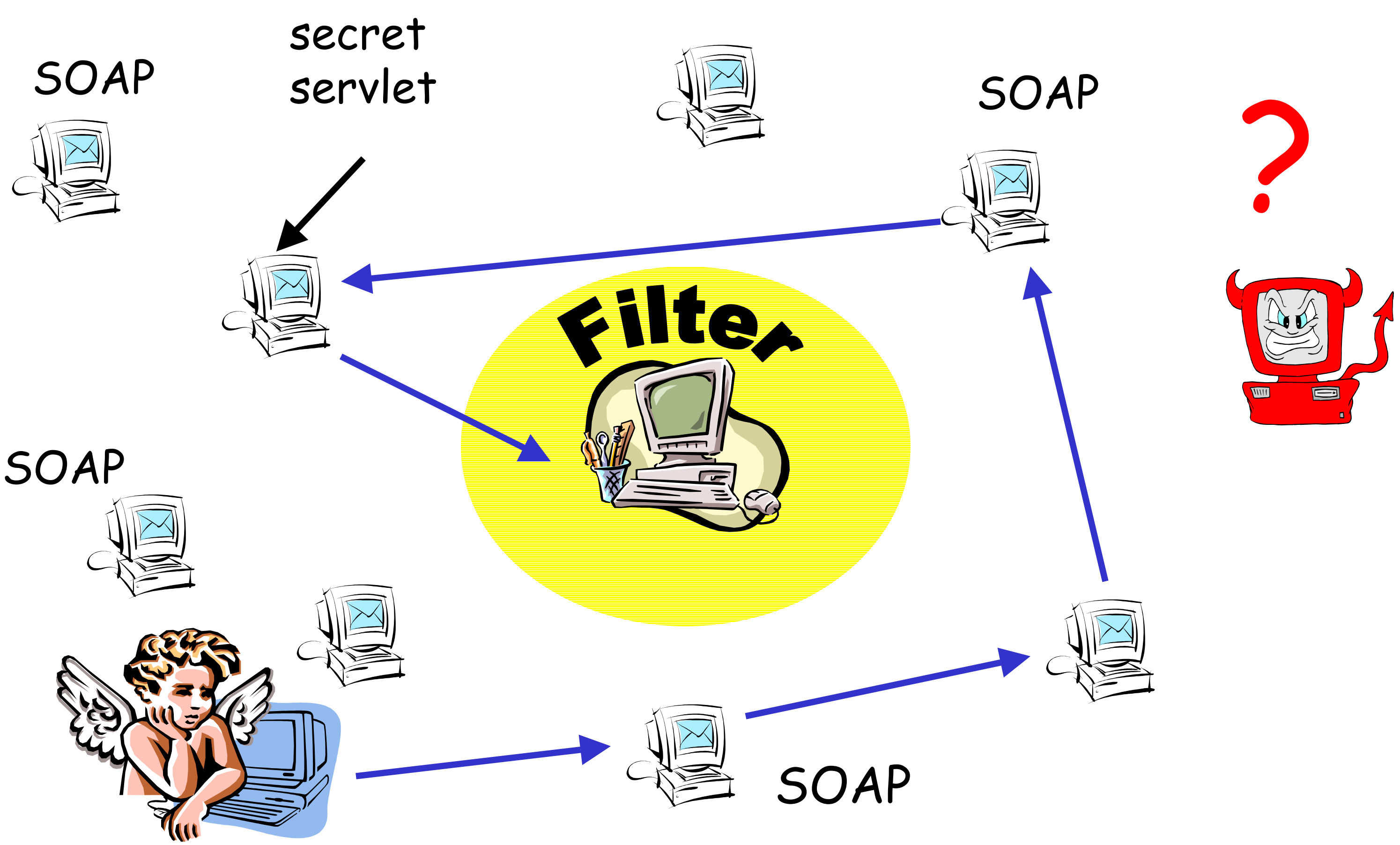
- Step 3: Keep the identity of the proxy "hidden"
  - hidden proxy called a **Secret Servlet**
  - only target, the secret servlet itself, and a few other points in the network know the secret servlet's identity (IP address)

SOS: Steps 4&5 - Overlays

- Step 4: Send traffic to the secret servlet via a network overlay
  - nodes in virtual network are often end-systems
  - verification/authentication of "legitimacy" of traffic can be performed at each overlay end-system hop (if/when desired)
- Step 5: Advertise a set of nodes that can be used by the legitimate user to access the overlay
  - these access nodes participate within the overlay
  - are called **Secure Overlay Access Points (SOAPs)**

User → SOAP → across overlay → Secret Servlet → (through filter) → target

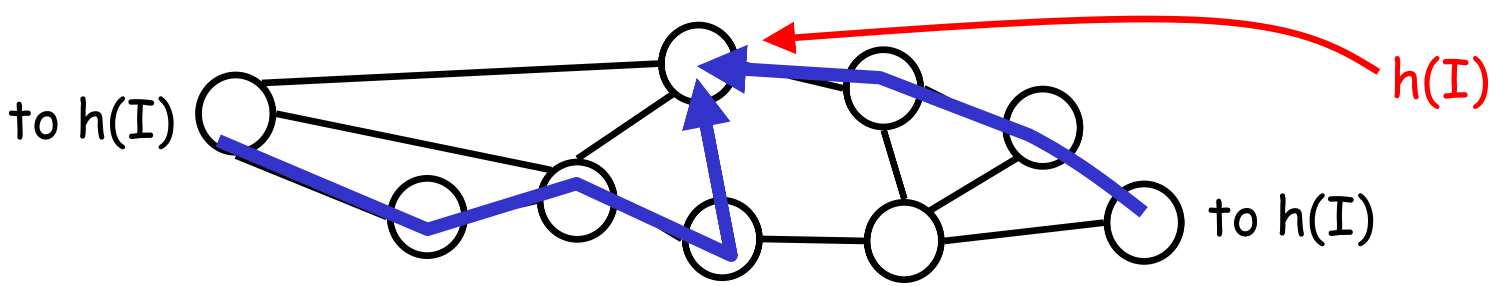
SOS with "Random" routing



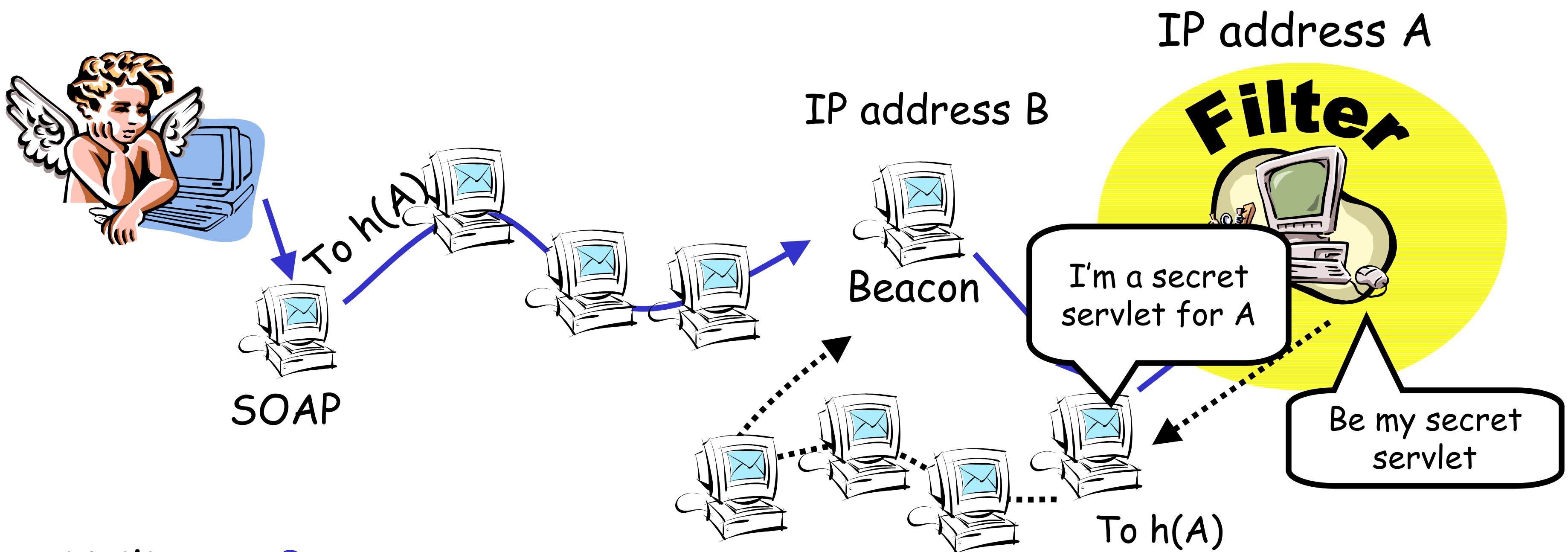
- With filters, multiple SOAPs, and hidden secret servlets, attacker cannot "focus" attack

Better than "Random" Routing

- Must get from SOAP to Secret Servlet in a "hard-to-predict manner": But random routing routes are long (O(n))
- Routes should not "break" as nodes join and leave the overlay (i.e., nodes may leave if attacked)
- Current proposed version uses DHT routing (e.g., Chord, CAN, PASTRY, Tapestry). We consider Chord:
  - Recall: A distributed protocol, nodes are used in homogeneous fashion
  - identifier, I, (e.g., filename) mapped to a unique node h(I) = B in the overlay
  - Implements a route from any node to B containing O(log N) overlay hops, where N = # overlay nodes



Step 5A: SOS with Chord



- Utilizes a Beacon to go from overlay to secret servlet
- Using target IP address A, Chord will deliver packet to a Beacon, B, where h(A) = B
- Secret Servlet chosen by target (arbitrarily)

SOS protected data packet forwarding

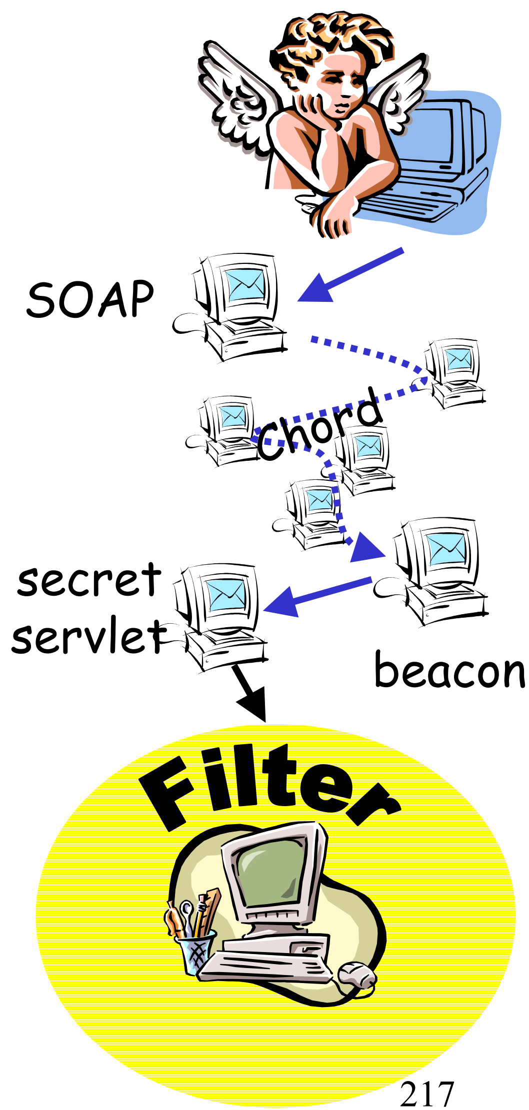
1. Legitimate user forwards packet to SOAP
2. SOAP forwards verified packet to Beacon (via Chord)
3. Beacon forwards verified packet to secret servlet
4. Secret Servlet forwards verified packet to target

Adding Redundancy in SOS

- Each special role can be duplicated if desired
  - Any overlay node can be a SOAP
  - The target can select multiple secret servlets
  - Multiple Beacons can be deployed by using multiple hash functions
- An attacker that successfully attacks a SOAP, secret servlet or beacon brings down only a subset of connections, and only while the overlay detects and adapts to the attacks

Why attacking SOS is difficult

- Attack the target directly (without knowing secret servlet ID): filter protects the target
- Attack secret servlets:
  - Well, they're hidden...
  - Attacked servlets "shut down" and target selects new servlets
- Attack beacons: beacons "shut down" (leave the overlay) and new nodes become beacons
  - attacker must continue to attack a "shut down" node or it will return to the overlay
- Attack other overlay nodes: nodes shut down or leave the overlay, routing self-repairs



Attack Success Analysis

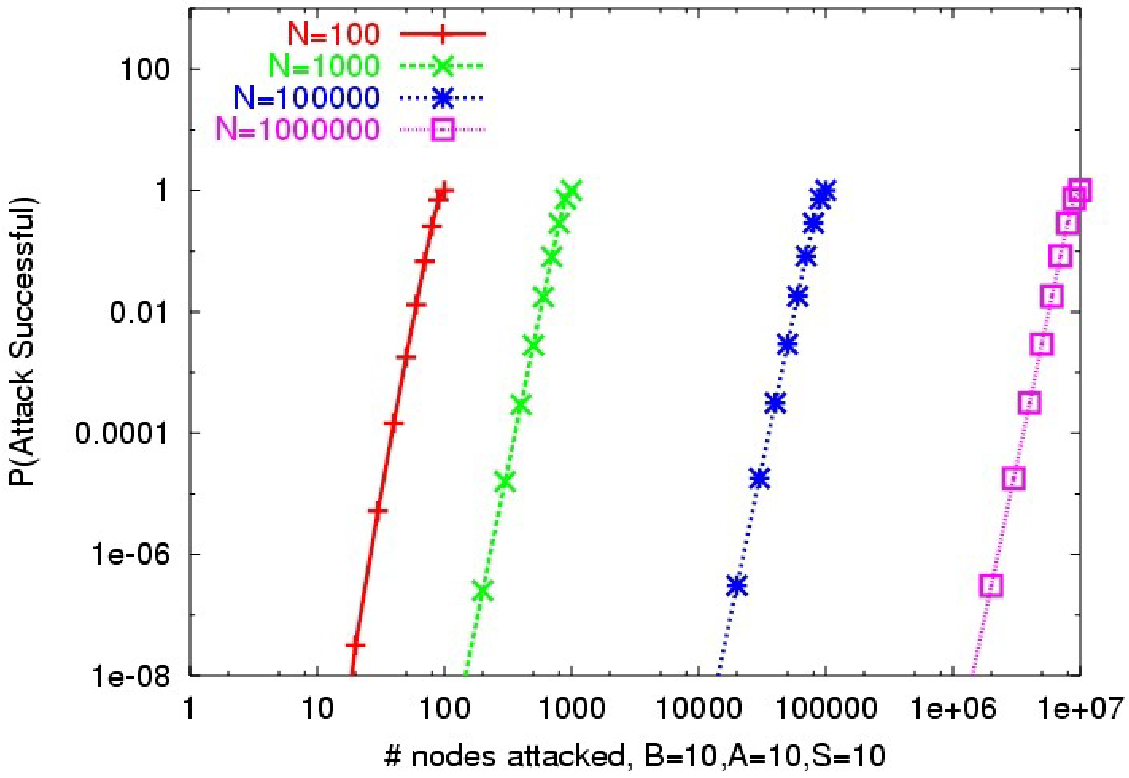
- N nodes in the overlay
- For a given target
  - S = # of secret servlet nodes
  - B = # of beacon nodes
  - A = # of SOAPs

Node jobs are assigned independently (same node can perform multiple jobs)

- Static attack: Attacker chooses M of N nodes at random and focuses attack on these nodes, shutting them down
- What is $P_{static}(N,M,S,B,A)$ = P(attack prevents communication with target)
- P(n,b,c) = P(set of b nodes chosen at random (uniform w/o replacement) from n nodes contains a specific set of c nodes)
- $P(n,b,c) = \binom{n-c}{b-c} / \binom{n}{b} = \binom{b}{c} / \binom{n}{c}$

Attack Success Analysis cont'd

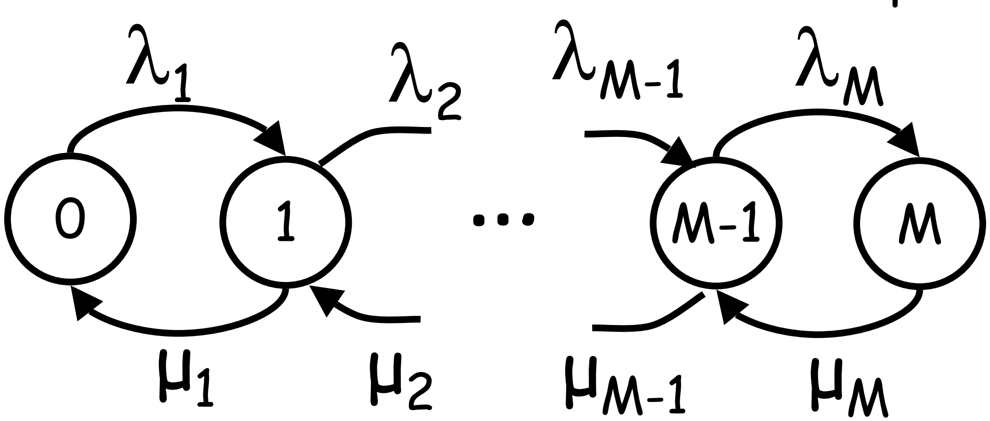
- Pstatic(N,M,S,B,A) = 1 - (1 - P(N,M,S))(1 – P(N,M,B))(1 – P(N,M,A))



Almost all overlay nodes must be attacked to achieve a high likelihood of DoS

Dynamic Attacks

- Ongoing attack/repair battle:
  - SOS detects & removes attacked nodes from overlay, repairs take time T_R
  - Attacker shifts from removed node to active node, detection/shift takes time T_A (freed node rejoins overlay)
- Assuming T_A and T_R are exponentially distributed R.V.'s, can be modeled as a birth-death process



M = Max # nodes simultaneously attacked

π_i = P(i attacked nodes currently in overlay)

$$P_{dynamic} = \sum_{0 \le i \le M} (\pi_i \cdot P_{static}(N-M+i, i, S, B, A))$$

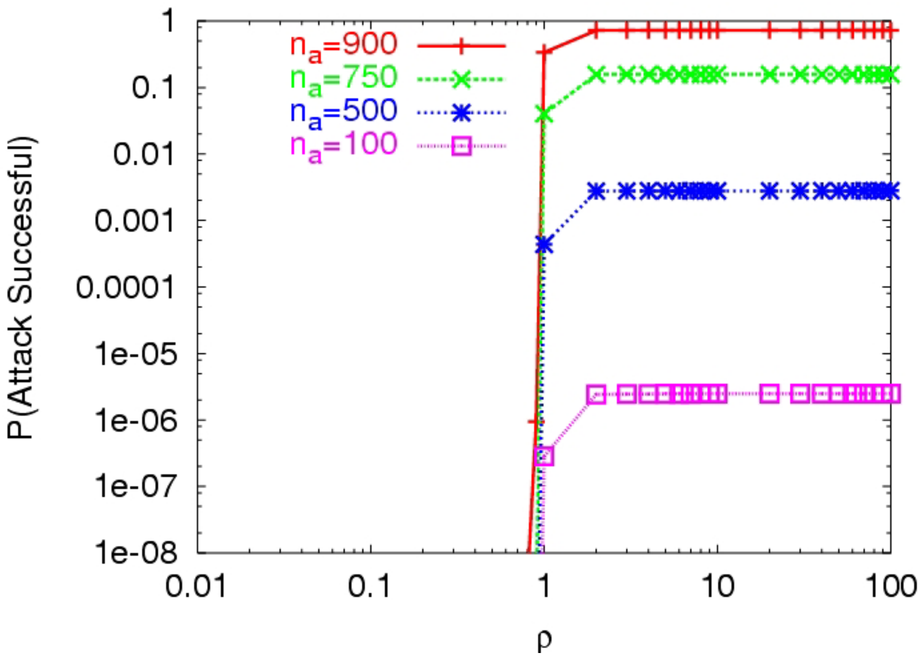
Centralized attack: $\lambda_i = \lambda$

Distributed attack: $\lambda_i = (M-i)\lambda$

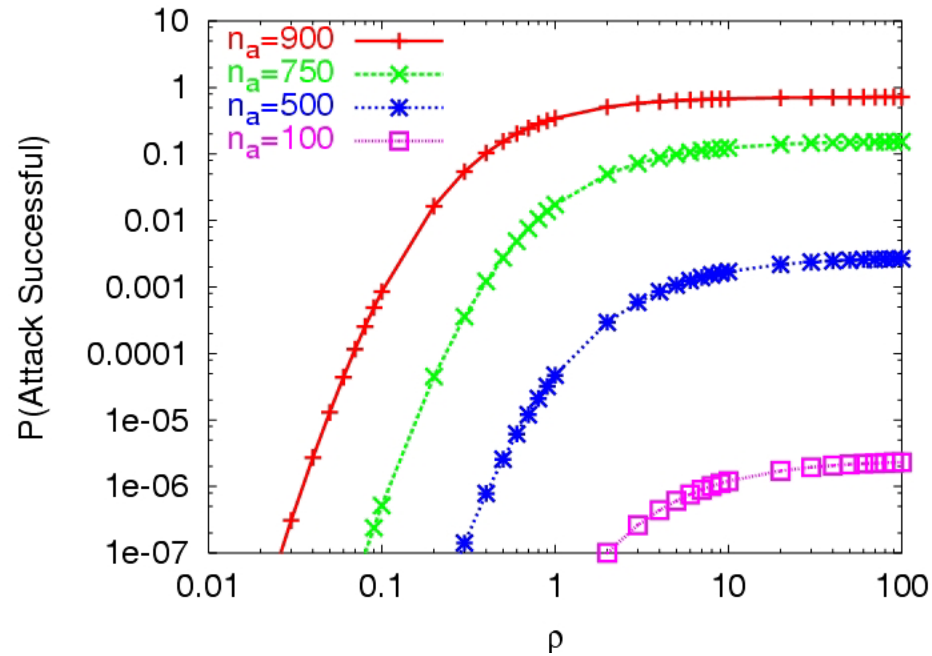
Centralized repair: $\mu_i = \mu$

Distributed repair: $\mu_i = i\mu$

Dynamic Attack Results



centralized attack and repair



distributed attack and repair

- 1000 overlay nodes, 10 SOAPs, 10 secret servlets, 10 beacons
- If repair faster than attack, SOS is robust even against large attacks (especially in centralized case)

SOS Summary

- SOS protects a target from DoS attacks
  - lets legitimate (authenticated) users through
- Approach
  - Filter around the target
  - Allow "hidden" proxies to pass through the filter
  - Use network overlays to allow legitimate users to reach the "hidden" proxies
- Preliminary Analysis Results
  - An attacker without overlay "insider" knowledge must attack majority of overlay nodes to deny service to target

5. Security in Structured P2P Systems

- Structured Systems described thusfar assume all nodes "behave"
  - Position themselves in forwarding structure to where they belong (based on ID)
  - Forward queries to appropriate next hop
  - Store and return content they are assigned when asked to do so
- How can attackers hinder operation of these systems?
- What can be done to hinder attacks?

Attacker Assumptions

- The attacker(s) participate in the P2P group
- Cannot view/modify packets not sent to them
- Can collude

Classes of Attacks

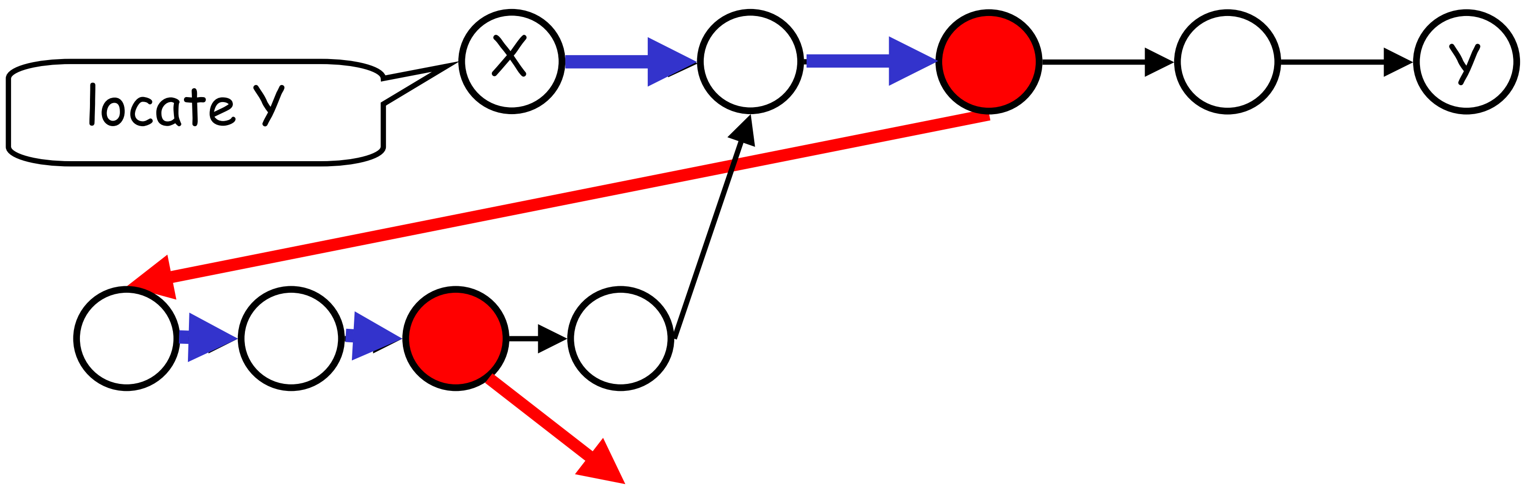
- Routing Attacks: re-route traffic in a “bad’ direction
- Storage/Retrieval Attacks: prevent delivery of requested data
- Miscellaneous
  - DoS (overload) nodes
  - Rapid joins/leaves

Identity Spoofing

- Problem:
  - Node claims to have an identity that belongs to other node
  - Node delivers bogus content
- Solution:
  - Nodes have certificates signed by trusted authority
  - Preventing spoofed identity: base identity on IP address, send query to verify the address.

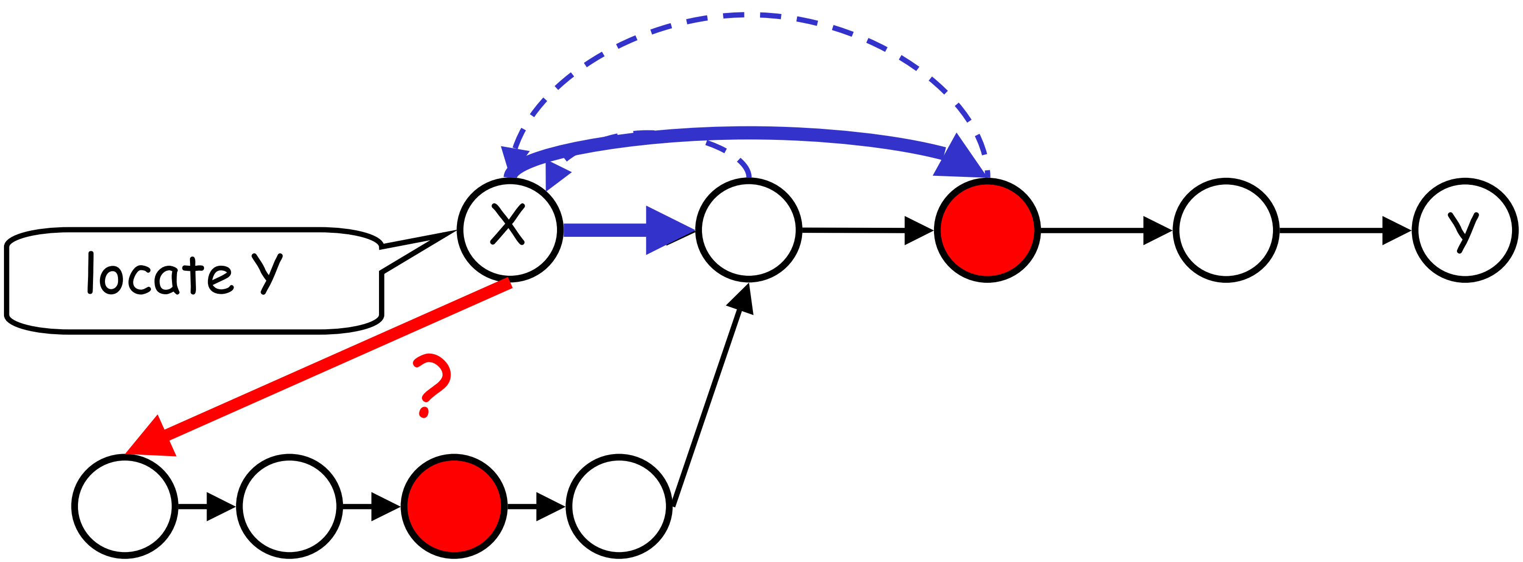
Routing Attacks 1: redirection

- Malicious node redirects queries in wrong direction or to non-existent nodes (drops)



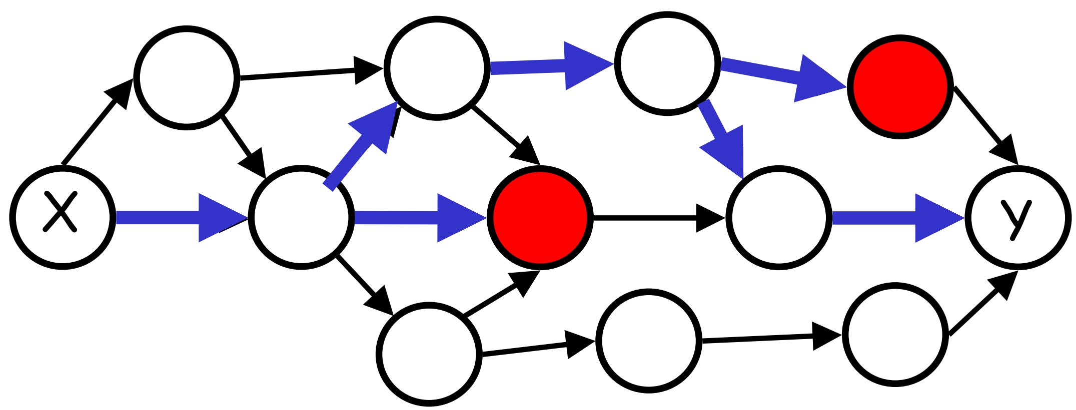
Suggested Solution: Part I

- Use iterative approach to reach destination.
  - verify that each hop moves closer (in ID space) to destination



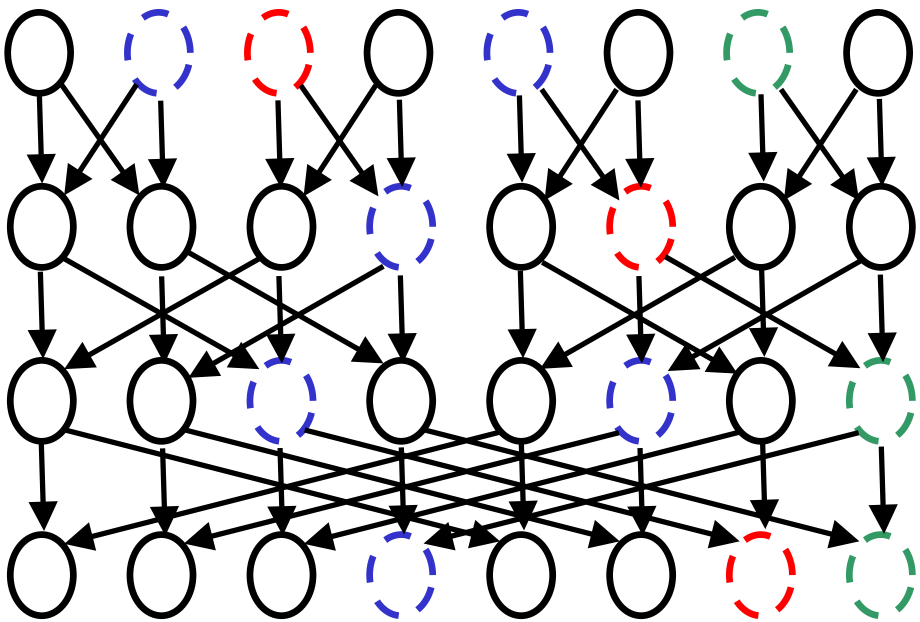
Suggested Solution: Part II

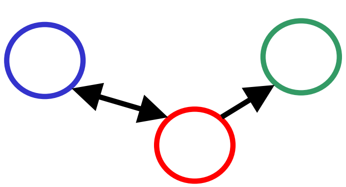
- Provide multiple paths to "re-route" around attackers



Choosing the Alternate paths: e.g., a CAN enhancement

- Use a butterfly network of virtual nodes w/ depth log n – log log n



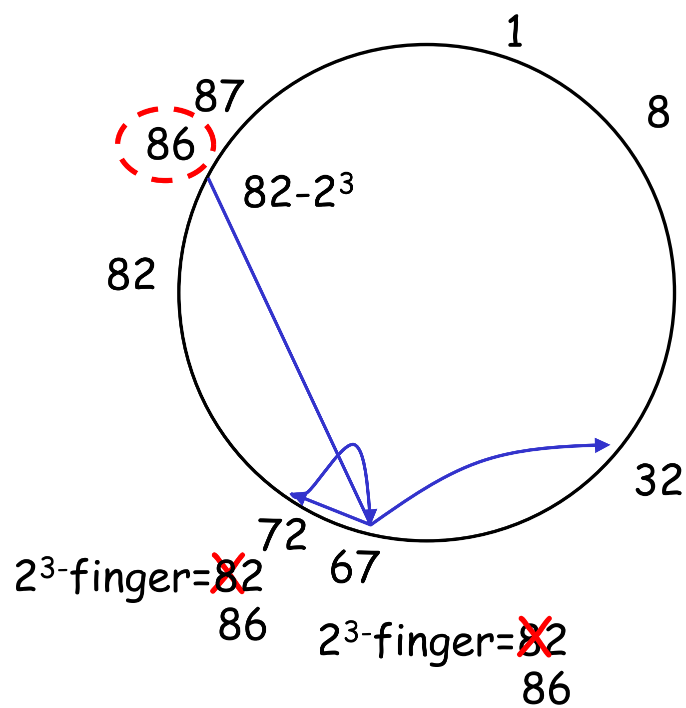


- Use:
  - Each real node maps to a set of virtual nodes
  - If edge (A,B) exists in Butterfly network, then form (A,B) in actual P2P overlay
  - "Flood" requests across the edges that form the butterfly
- Results: For any ε, there are constants such that
  - search time is O(log n)
  - insertion is O(log n)
  - # search messages is $O(\log^2 n)$
  - each node stores $O(\log^3 n)$ pointers to other nodes and O(log n) data items
  - All but a fraction ε of peers can access all but a fraction ε of content

Routing Attack 2: Misleading updates

- An attacker could trick nodes into thinking other nodes have left the system
- Chord Example: node "kicks out" other node
- Similarly, could claim another (non-existent) node has joined
- Proposed solution: random checks of nodes in P2P overlay, exchange of info among "trusted" nodes

Malicious node 86 "kicks out" node 82



e.g., for i=3

Routing Attack 3: Partition

- A malicious bootstrap node sends newcomers to a P2P system that is disjoint from (no edges to) the main P2P system
- Solutions:
  - Use a trusted bootstrap server
  - Cross-check routing via random queries, compare with trusted neighbors (found outside the P2P ring)

Storage/Retrieval Attacks

- Node is responsible for holding data item D. Does not store or deliver it as required
- Proposed solution: replicate object and make available from multiple sites

Miscellaneous Attacks

- Problem: Inconsistent Behavior - Node sometimes behaves, sometimes does not
- Solution: force nodes to "sign" all messages. Can build body of evidence over time
- Problem: Overload, i.e., DoS attack
- Solution: replicate content and spread out over network
- Problem: Rapid Joins/Leaves
- Solutions: ?

5. Anonymnity

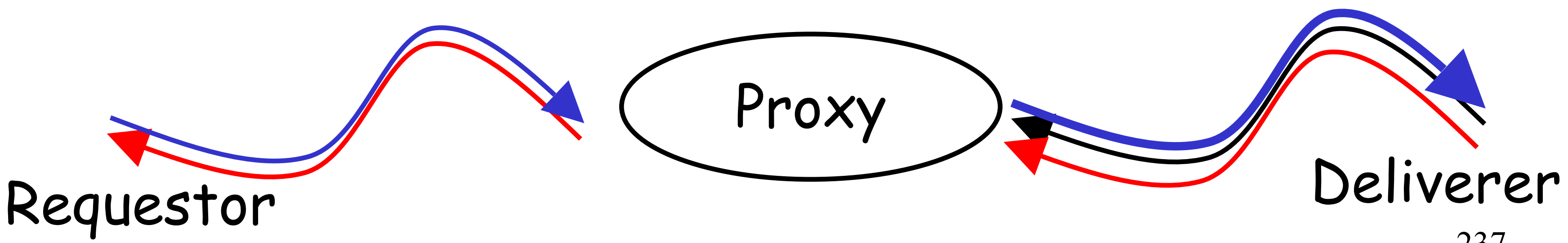
- Suppose clients want to perform anonymous communication
  - requestor wishes to keep its identity secret
  - deliverer wishes to also keep identity secret

Onion Routing

- A Node N that wishes to send a message to a node M selects a path (N, V_1, V_2, …, V_k, M)
  - Each node forwards message received from previous node
  - N can encrypt both the message and the next hop information recursively using public keys: a node only knows who sent it the message and who it should send to
- N's identity as originator is not revealed

Anonymnity on both sides

- A requestor of an object receives the object from the deliverer without these two entities exhanging identities
- Utilizes a proxy
  - Using onion routing, deliverer reports to proxy (via onion routing) the info it can deliver, but does not reveal its identity
  - Nodes along this onion-routed path, A, memorize their previous hop
  - Requestor places request to proxy via onion-routing, each node on this path, B, memorize previous hop
  - Proxy→Deliverer follows "memorized" path A
  - Deliverer sends article back to proxy via onion routing
  - Proxy→Requestor via "memorized" path B



6. P2P Graph Structure

- What are “good” P2P graphs and how are they built?
- Graphs we will consider
  - Random (Erdos-Renyi)
  - Small-World
  - Scale-free

“Good” Unstructured P2P Graphs

- Desirable properties
  - each node has small to moderate degree
  - expected # of hops needed to go from a node u to a node v is small
  - easy to figure out how to find the right path
  - difficult to attack the graph (e.g., by knocking out nodes)
  - don’t need extensive modifications when nodes join/leave (e.g., like in Chord, CAN, Pastry)
- Challenge: Difficult to enforce structure

Random (Erdos-Renyi) Graphs

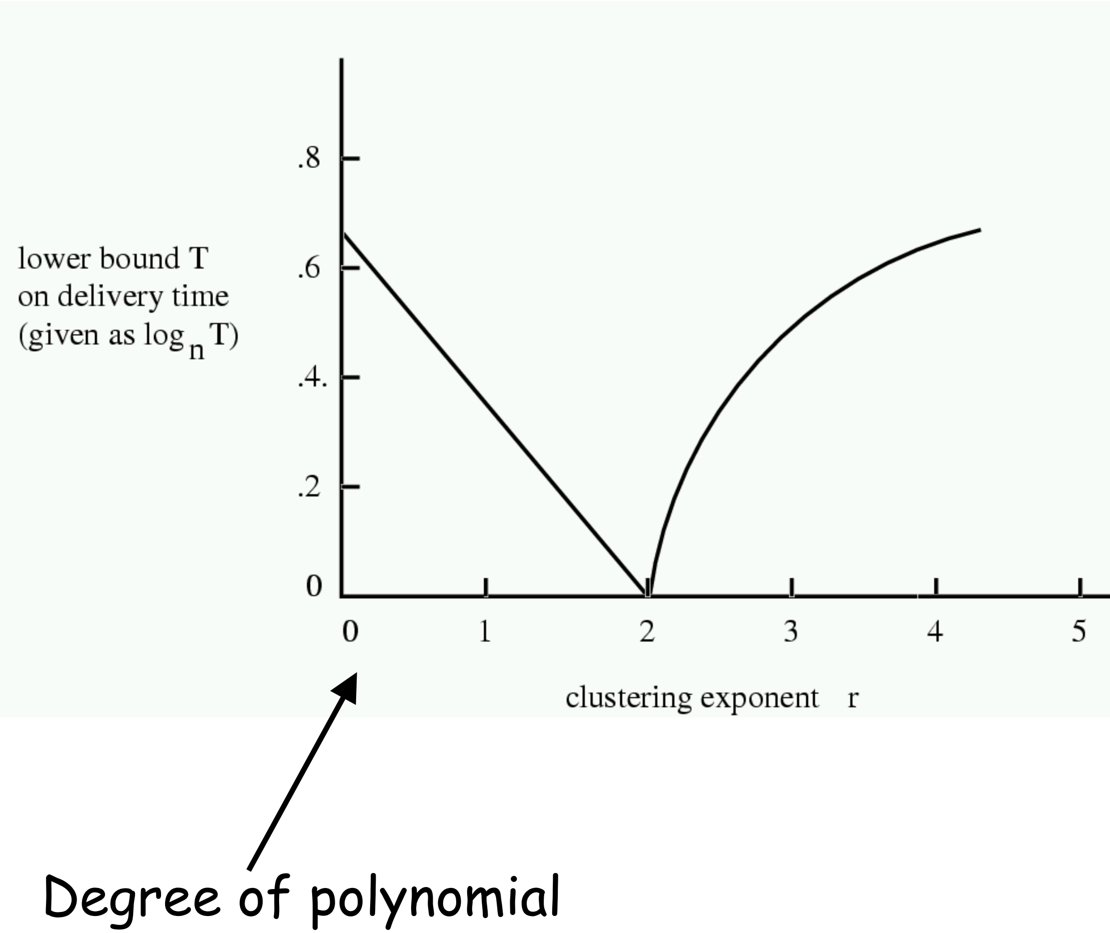
- For all nodes u,v, edge (u,v) is added with fixed probability p
- Performance in P2P Context: In some sense, these graphs are too random
  - long distance between pairs of nodes likely
  - difficult to build a good distributed algorithm that can find a short route between arbitrary pair of nodes

Small World Model

- Nodes have positions (e.g., on a 2D graph)
- Let d(u,v) be the distance between nodes u & v
- Constants p, q, r chosen:
  - each node u connects to all other nodes v where $d(u,v) < p$
  - each node connects to q additional (far away) nodes drawn from distribution where edge (u,v) is selected with probability proportional to $d(u,v)^{-r}$
  - Each node knows all neighbors within distance p and also knows q far neighbors
  - Search method: choose the neighbor that is closest (in distance) to the desired destination

Optimal Small-World Config

- Proven in [Kle00] that only for r=2 can a distributed algorithm reach the destination in expected time $O(\log^2 n)$
- For other r, time is polynomial in n

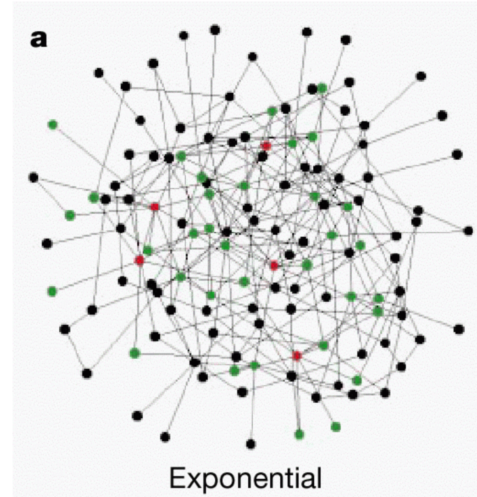


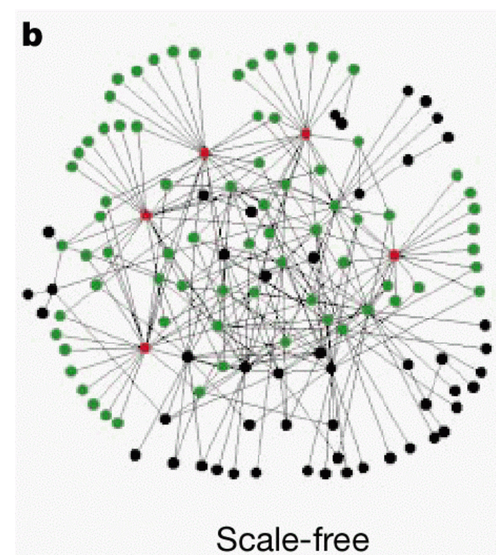
Small-World Freenet

- Freenet Architecture
  - each object has unique identifier/key (similar to DHTs)
  - search method is unstructured
- Small-World Modification using Kleinberg's result: each node maintains a set of neighbors according to Small-World criterion
- Search algorithm: always get as close to destination as possible, reverse path if node has no neighbors that are closest to destination
- Result: search time/messaging is $O(\log^2 n)$ with nodes having $O(\log^2 n)$ neighbors.

Scale-Free Graphs

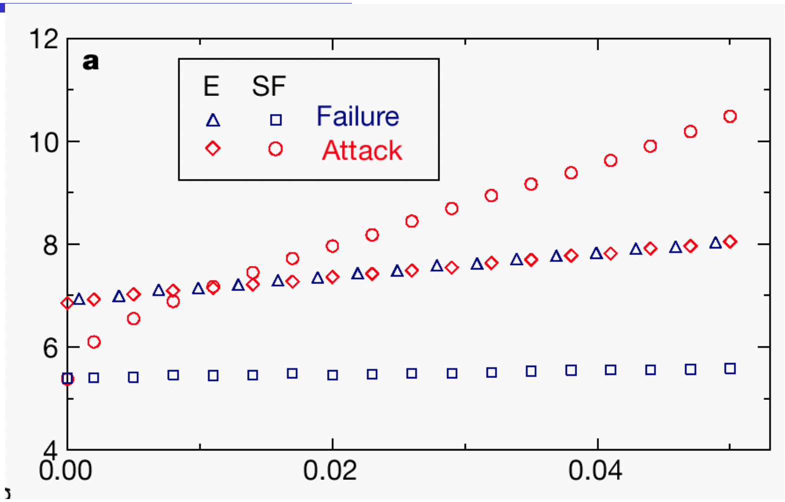
- Erdos-Renyi and Small-World graphs are exponential: the degree of nodes in the network decays exponentially
- Scale-free graph: node connects to node with current degree k with probability proportional to k
  - nodes with many neighbors more likely to get more neighbors
- Scale-free graphs' degree decays according to a power law: Pr(node has k neighbors) = $k^{-\alpha}$

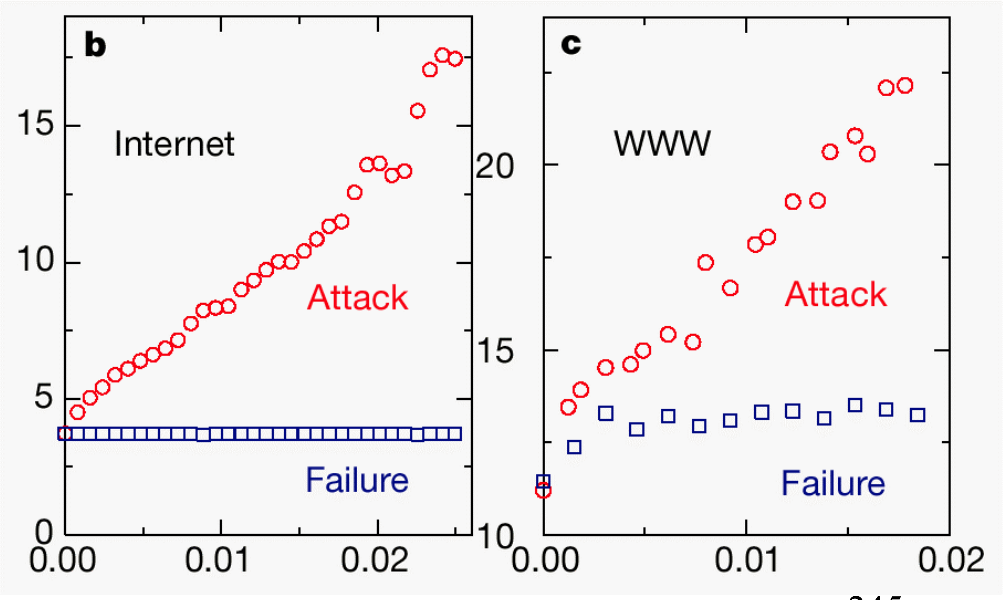




Are Scale-Free Networks Better?

- Average diameter lower in Scale-Free than in Exponential graphs
- What if nodes are removed?
  - at random: scale free keeps lower diameter
  - by knowledgable attacker: (nodes of highest degree removed first): scale-free diameter grows quickly
- Same results apply using sampled Internet and WWW graphs (that happen to be scale-free)





7. Measurement of Existing P2P Systems

- Systems observed
  - Gnutella
  - Kazaa
  - Overnet (DHT-based)
- Measurements described
  - fraction of time hosts are available (availability)
  - popularity distribution of files requested
  - # of files shared by host

Results from 3 studies

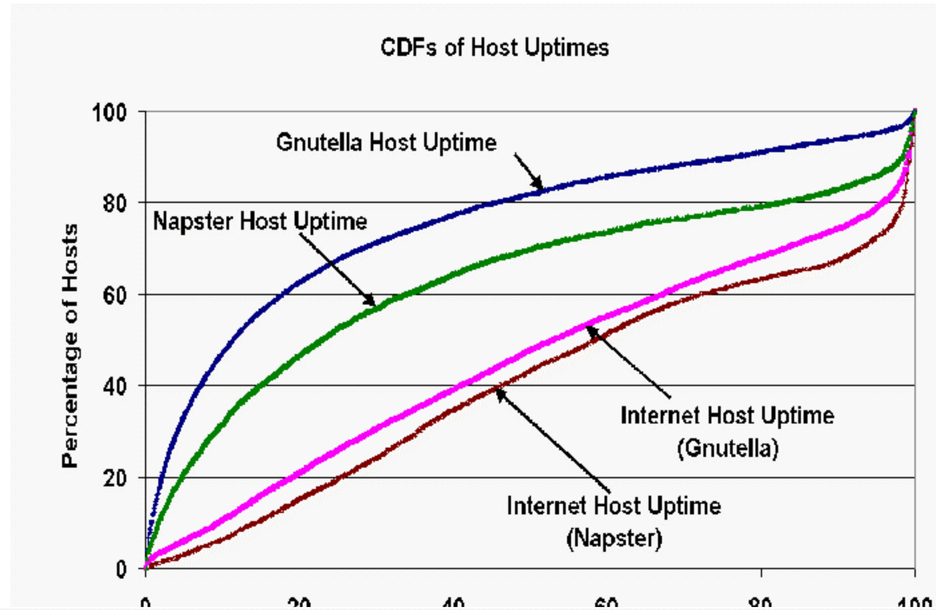
- [Sar02]
  - Sampled Gnutella and Napster clients for 8 and 4 day period
  - measured availability, bandwidths, propagation delays, file sizes, file popularities
- [Chu02]
  - Sampled Gnutella and Napster clients for monthlong period
  - measured availability, file sizes and popularities
- [Bha03]
  - Sampled Overnet clients for a week-long period
  - Measured availability, error due to use of IP address as identifier

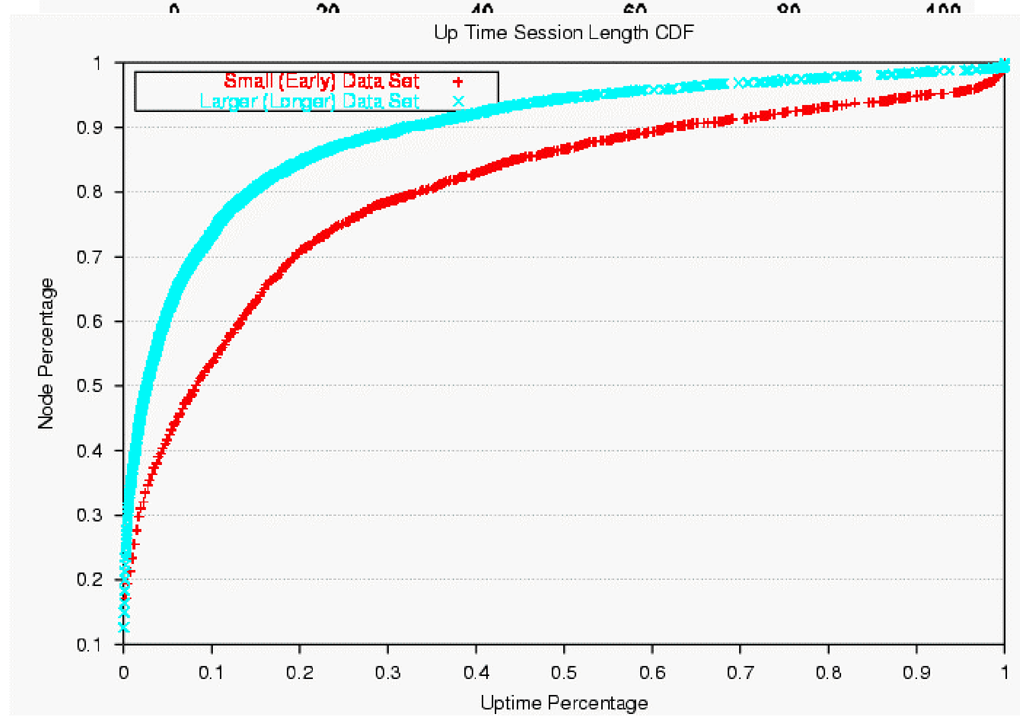
Methods used

- Identifying clients
  - Napster: ask central server for clients that provide popular names of files
  - Gnutella: send pings to well-known (bootstrap) peers and obtain their peer lists
  - Overnet: search for random IDs
- Probing:
  - Bandwidth/latency: tools that take advantage of TCP's reliability and congestion control mechanism
  - Availability/Files offered, etc: pinging host (by whatever means is necessary for the particular protocol, usually by mechanism provided in protocol)

Availability

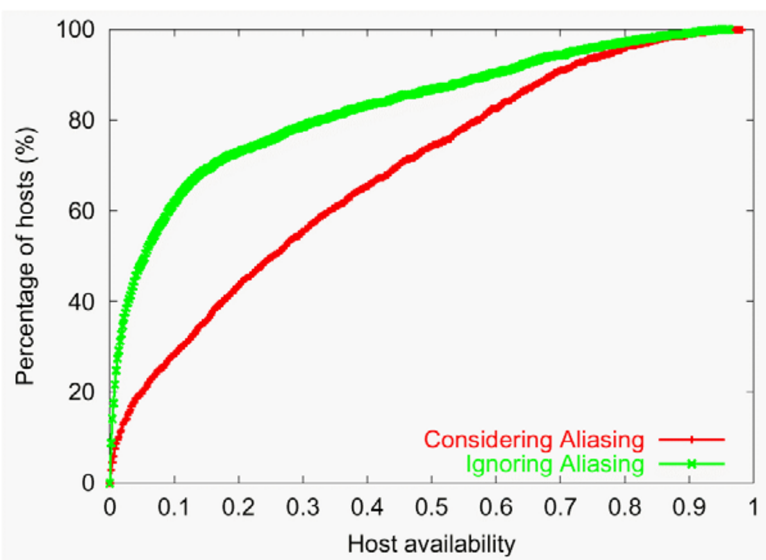
- [Sar02] results: application uptime CDF is concave
- [Chu02]: short studies overestimate uptime percentage
  - Implication: clients' use of P2P tool is performed in bursty fashion over long timescales

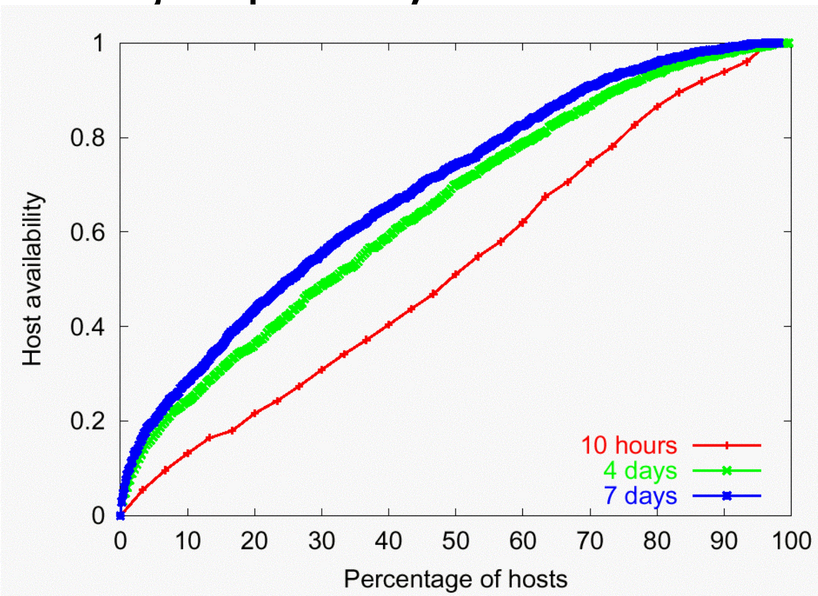




Availability con'td

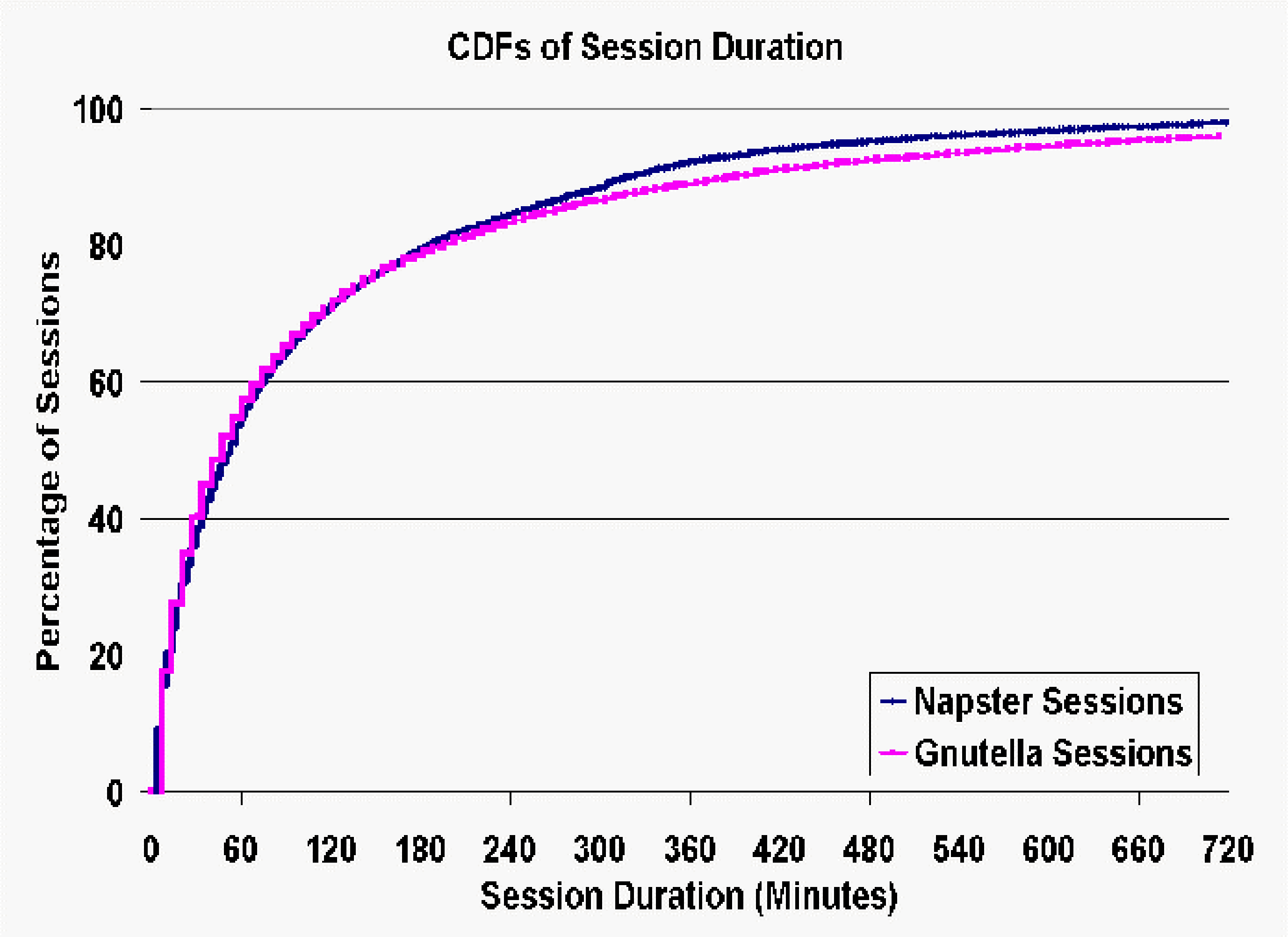
- [Bha03]: using IP address to identify P2P client can be inaccurate
  - nodes behind NAT box share IP address
  - address can change when using DHCP
  - [Chu02] results about availability as function of period similar even when clients are not "grouped" by IP address





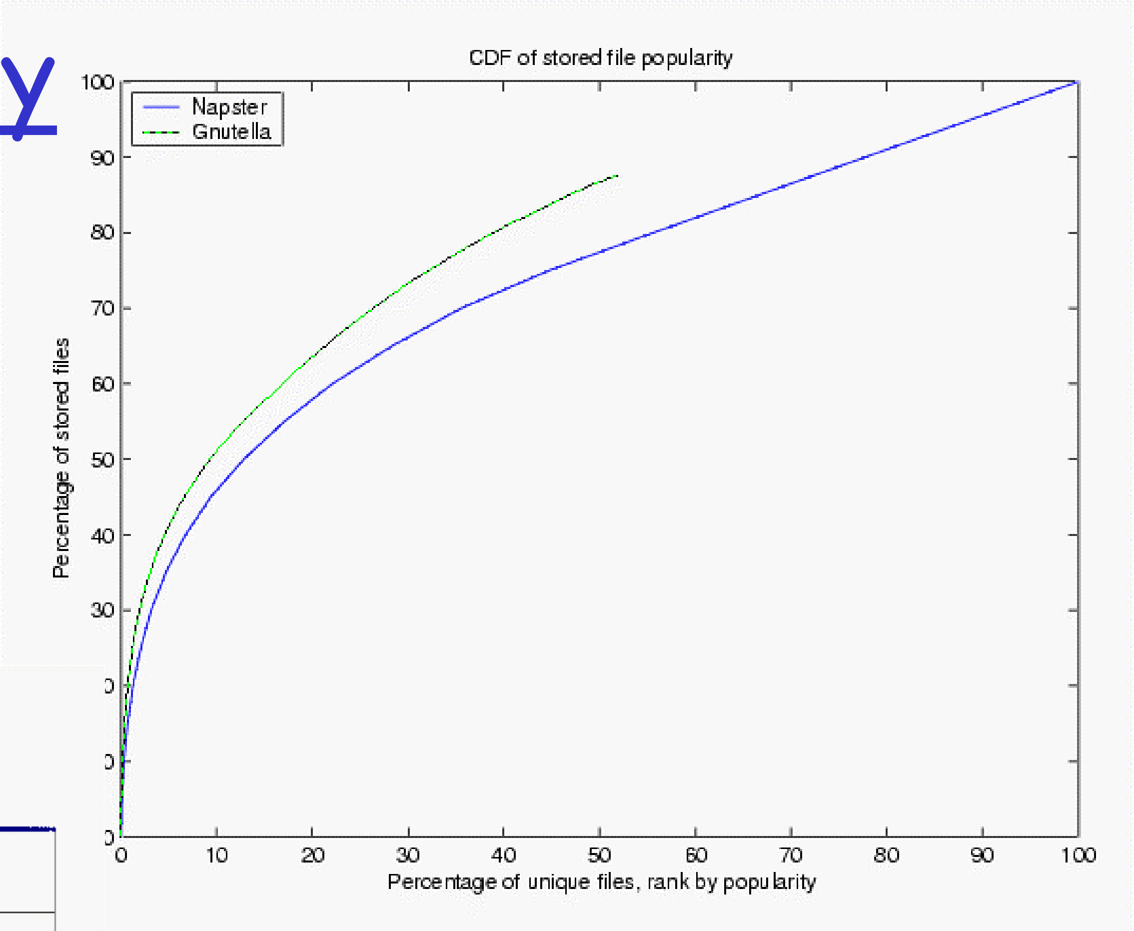
Session Duration

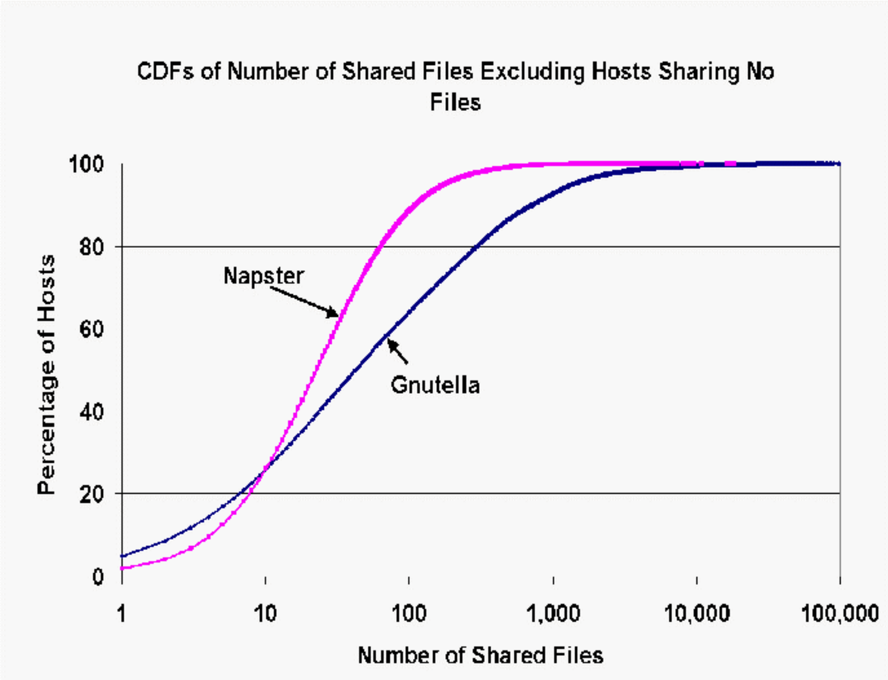
- [Sar02]



File popularity

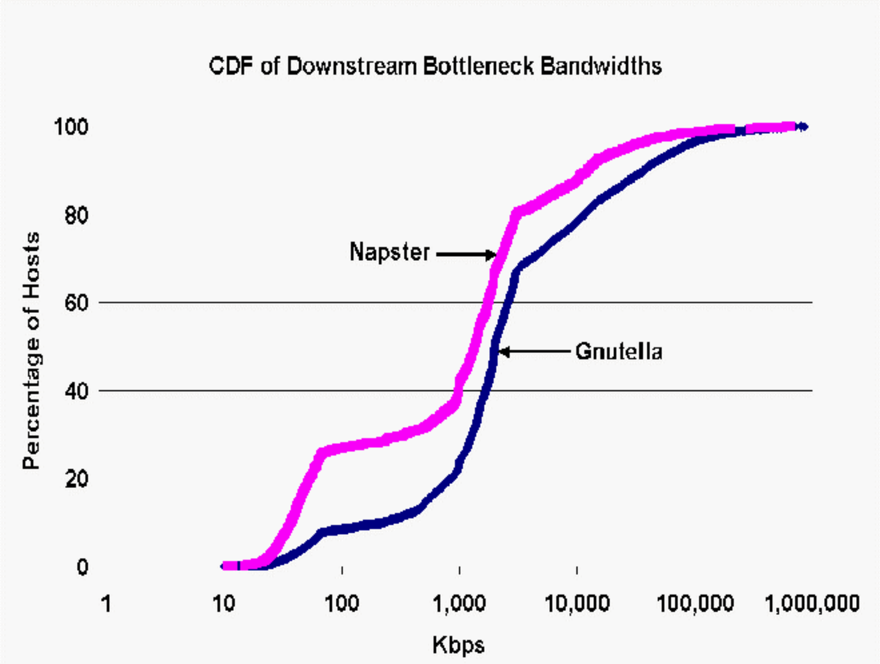
- Popular files are more popular in Gnutella than in Napster

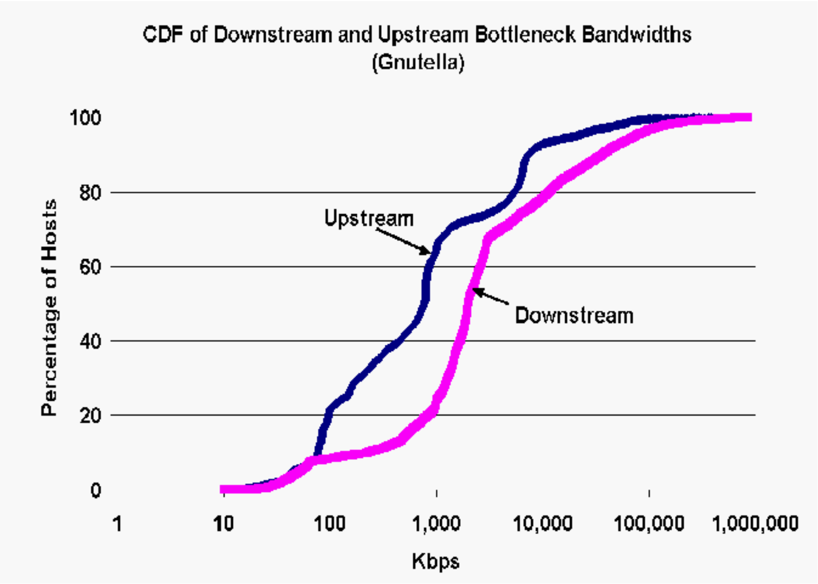




- Gnutella clients more likely to share more files

Bottleneck Bandwidths of Clients





9. Future Research

- Specific
  - **Locality** in DHT-based systems: how to "guarantee" copies of objects in the local area
- General
  - **Using DHTs**: To hash or not to hash (are DHTs a good thing)?
  - **Trust**: Building a "trusted" system from autonomous, untrusted / semi-trusted collections
  - **Dynamicity**: Building systems that operate in environments where nodes join/leave/fail at high rates

Selected References

- A. Oram (ed), *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, O'Reilly & Associates, 2001
- F. von Lohmann, "P2P File Sharing and Copyright Law: A Primer for Developers," *IPTPS 2003*
- David P. Anderson and John Kubiatowicz, The Worldwide Computer, *Scientific American*, March 2002
- Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications", *Proceedings of ACM SIGCOMM'01, San Diego, CA, August 2001.*
- Bujor Silaghi, Bobby Bhattacharjee, Pete Keleher, "Query Routing in the TerraDir Distributed Directory", *Proceedings of SPIE ITCOM, Boston, MA, July 2002.*
- Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Shenker, "A Scalable Content-Addressable Network", *Proceedings of ACM SIGCOMM'01, San Diego, CA, August 2001.*
- OceanStore: An Architecture for Global-Scale Persistent Storage , John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao. Appears in *Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000),* November 2000
- W. J. Bolosky, J. R. Douceur, D. Ely, M. Theimer; Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs, Proceedings of the international conference on Measurement and modeling of computer systems, 2000, pp. 34-43
- *J.* Kleinberg, The Small-World Phenomenon: An Algorithmic Perspective*, Proc. 32nd ACM Symposium on Theory of Computing,* Portland, OR, May, 2000
- R. Albert, H. Joeong, A. Barabasi, Error and Attack Tolerance of Complex Networks, *Nature*, vol. 46, July 2000.
- H. Zhang, A. Goel, R. Govindan, Using the Small-World Model to Improve Freenet Performance, *Proceedings of IEEE Infocom*, New York, NY, June 2002.
- J. Chu, K. Labonte, B. Levine, Availability and Locality Measurements of Peer-to-Peer File Systems, *Proceedings of SPIE ITCOM*, Boston, MA, July 2002.
- R. Bhagwan, S. Savage, G. Voelker, Understanding Availability, in *Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, Berkeley, CA, Feb 2003.
- S. Saroiu, P. Gummadi, S. Gribble, A Measurement Study of Peer-to-Peer File Sharing Systems, in *Proceedings of Multimedia Computing and Networking 2002 (MMCN'02)*, San Jose, CA, January 2002.

- Antony Rowstron and Peter Druschel, "Pastry: Scalable, Decentralized, Object Location and Routing for Large-scale Peer-to-peer Systems", *Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms (Middelware)'02*
- Ben Y. Zhao, John Kubiatowicz, Anthony Joseph, "Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing", *Technical Report, UC Berkeley*
- A. Rowstron and P. Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility", 18th ACM SOSP'01, Lake Louise, Alberta, Canada, October 2001.
- *S. Iyer, A. Rowstron and P. Druschel, "SQUIRREL: A decentralized, peer-to-peer web cache", appeared in Principles of Distributed Computing (PODC 2002), Monterey, CA*
- Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica, *Wide-area cooperative storage with CFS*, ACM SOSP 2001, Banff, October 2001
- Ion Stoica, Daniel Adkins, Shelley Zhaung, Scott Shenker, and Sonesh Surana, Internet Indirection Infrastructure, in *Proceedings of ACM SIGCOMM'02*, Pittsburgh, PA, August 2002, pp. 73-86
- L. Garces-Erce, E. Biersack, P. Felber, K.W. Ross, G. Urvoy-Keller, Hierarchical Peer-to-Peer Systems, 2003, http://cis.poly.edu/~ross/publications.html
- Kangasharju, K.W. Ross, D. Turner, Adaptive Content Management in Structured P2P Communities, 2002, http://cis.poly.edu/~ross/publications.html
- K.W. Ross, E. Biersack, P. Felber, L. Garces-Erce, G. Urvoy-Keller, TOPLUS: Topology Centric Lookup Service, 2002, http://cis.poly.edu/~ross/publications.html
- P. Felber, E. Biersack, L. Garces-Erce, K.W. Ross, G. Urvoy-Keller, Data Indexing and Querying in P2P DHT Networks, http://cis.poly.edu/~ross/publications.html
- K.W. Ross, Hash-Routing for Collections of Shared Web Caches, *IEEE Network Magazine,* Nov-Dec 1997
- A. Keromytis, V. Misra, D. Rubenstein, SOS: Secure Overlay Services, in *Proceedings of ACM SIGCOMM'02*, Pittsburgh, PA, August 2002
- M. Reed, P. P. Syverson, D. Goldschlag, Anonymous Connections and Onion Routing, *IEEE Journal on Selected Areas of Communications*, Volume 16, No. 4, 1998.
- V. Scarlata, B. Levine, C. Shields, Responder Anonymity and Anonymous Peer-to-Peer File Sharing, in *Proc. IEEE Intl. Conference on Network Protocols (ICNP),* Riverside, CA, November 2001.
- E. Sit, R. Morris, Security Considerations for Peer-to-Peer Distributed Hash Tables, in *Proc. 1st International Workshop on Peer-to-Peer Systems (IPTPS),* Cambridge, MA, March 2002.
- J. Saia, A. Fiat, S. Gribble, A. Karlin, S. Sariou, Dynamically Fault-Tolerant Content Addressable Networks, in *Proc. 1st International Workshop on Peer-to-Peer Systems (IPTPS),* Cambridge, MA, March 2002.
- M. Castro, P. Druschel, A. Ganesh, A. Rowstron, D. Wallach, Secure Routing for Structured Peer-to-Peer Overlay Netwirks, In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI'02),* Boston, MA, December
- 2002.
- Edith Cohen and Scott Shenker, "Replication Strategies in Unstructured Peer-to-Peer Networks", in *Proceedings of ACM SIGCOMM'02*, Pittsburgh, PA, August 2002
- Dan Rubenstein and Sambit Sahu, "An Analysis of a Simple P2P Protocol for Flash Crowd Document Retrieval", Columbia University Technical Report