Peer-peer and Application-level Networking

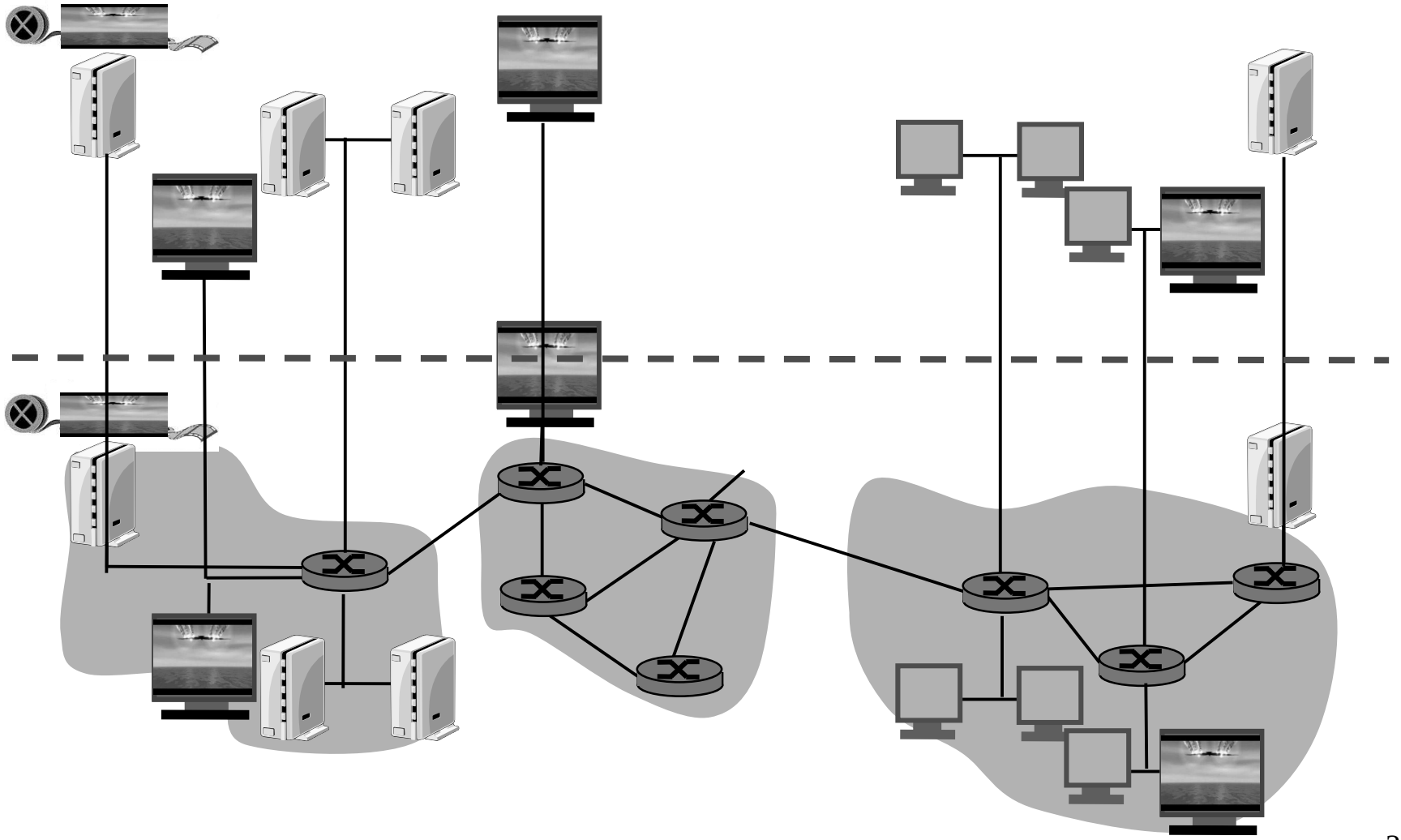
Presented by Richard Gold

Based ***strongly*** on material by

Jim Kurose,Brian Levine,Don Towsley, and the class of 2001 for the Umass Comp Sci 791N course..

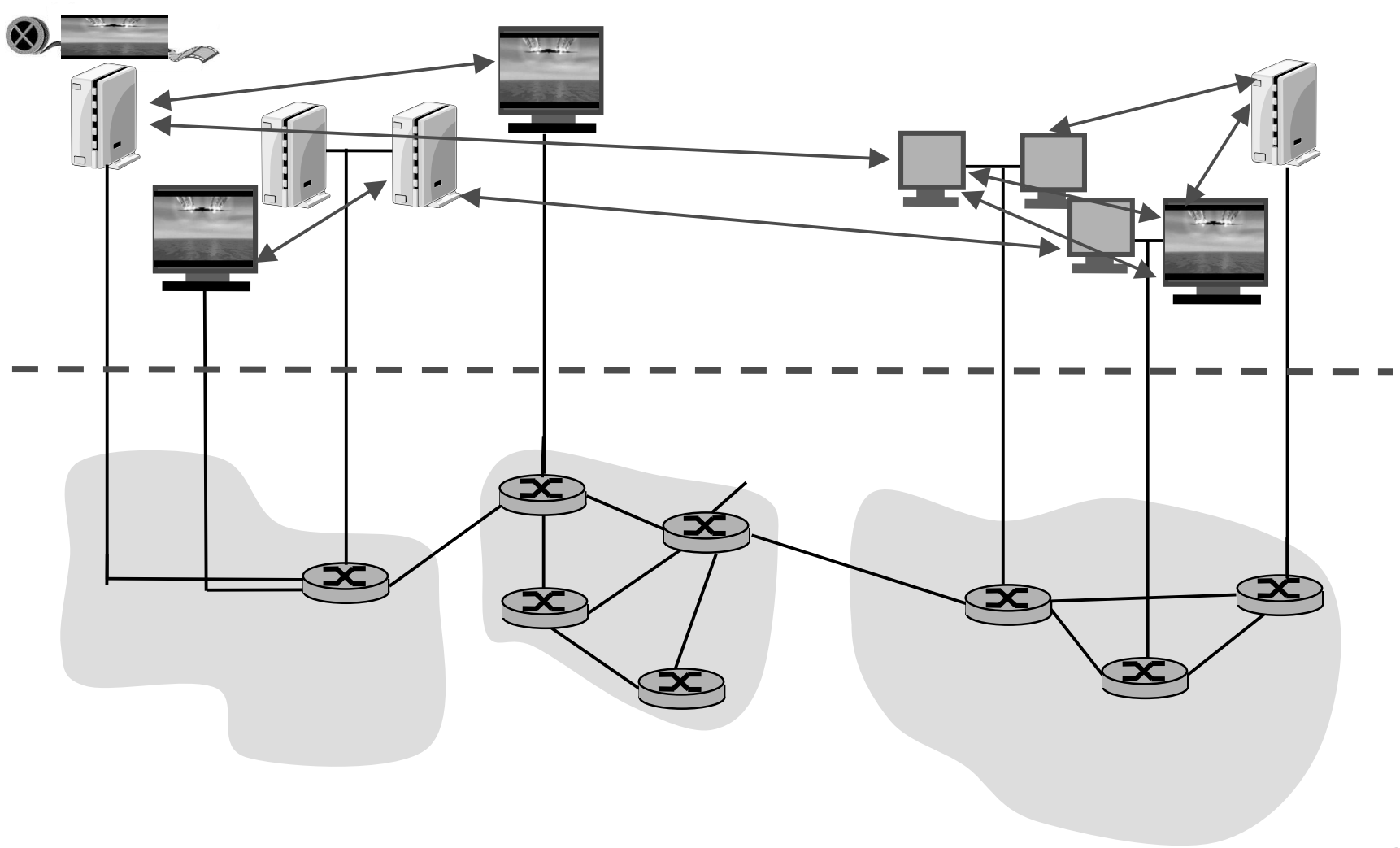
Originally presented by Jon Crowcroft

Peer-peer networking



Peer-peer networking

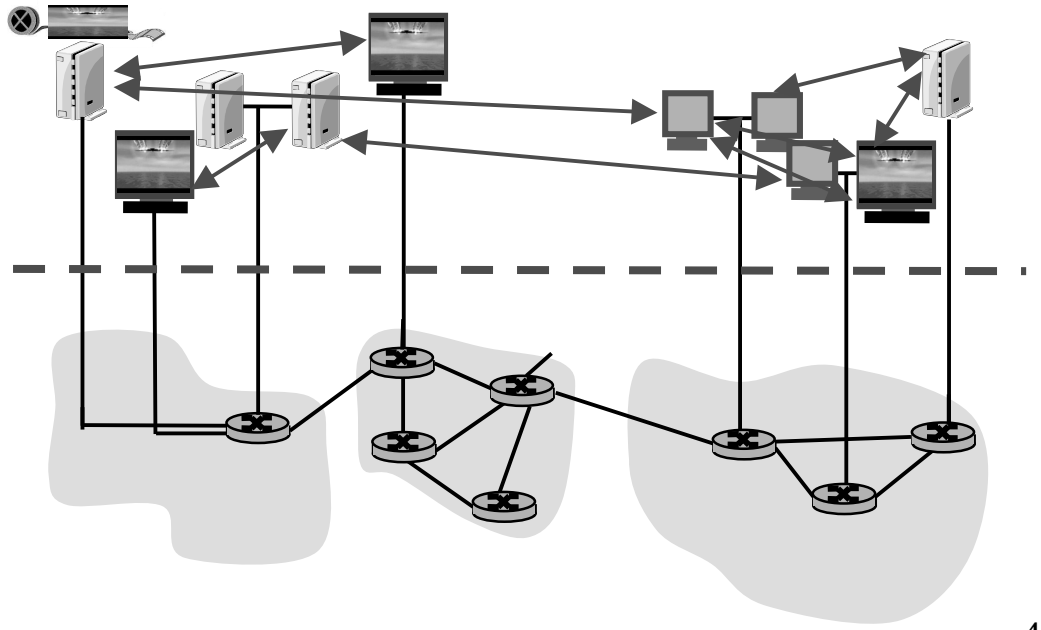
Focus at the application level



Peer-peer networking

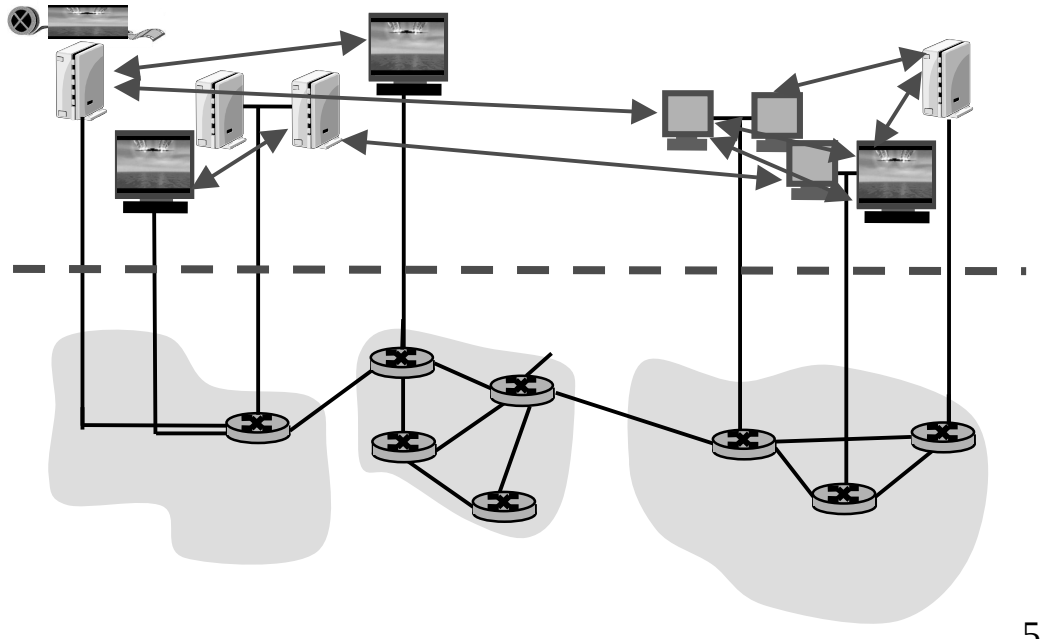
Peer-peer applications

- Napster, Gnutella, Freenet: file sharing
- ad hoc networks
- multicast overlays (e.g., video distribution)



Peer-peer networking

- Q: What are the new technical challenges?
- Q: What new services/applications enabled?
- Q: Is it just "networking at the application-level"?
  - "There is nothing new under the Sun" (William Shakespeare)



Client Server v. Peer to Peer

- RPC/RMI
- Synchronous
- Assymmetric
- Emphasis on language integration and binding models (stub *IDL/XDR* compilers etc)
- Kerberos style security – access control, crypto

- Messages
- Asynchronous
- Symmetric
- Emphasis on service location, content addressing, application layer routing.
- Anonymity, high availability, integrity.
- Harder to get right☺

Peer to peer systems actually old

- IP routers are peer to peer.
- Routers discover topology, and maintain it
- Routers are neither client nor server
- Routers continually chatter to each other
- Routers are fault tolerant, inherently
- Routers are autonomous

Peer to peer systems

- Have no distinguished role
- So no single point of bottleneck or failure.
- However, this means they need distributed algorithms for
  - Service discovery (name, address, route, metric, etc)
  - Neighbour status tracking
  - Application layer routing (based possibly on content, interest, etc)
  - Resilience, handing link and node failures
  - Etc etc etc

Ad hoc networks and peer2peer

- Wireless ad hoc networks have many similarities to peer to peer systems
- No *a priori knowledge*
- No given infrastructure
- Have to construct it from "thin air"!
- Note for later – wireless☺

Overlays and peer 2 peer systems

- P2p technology is often used to create overlays which offer services that could be offered in the IP level
- Useful **deployment** strategy
- Often economically a way around other barriers to deployment
- IP Itself was an overlay (on telephone core infrastructure)
- Evolutionary path!!!

Rest of lecture oriented from case studies from literature

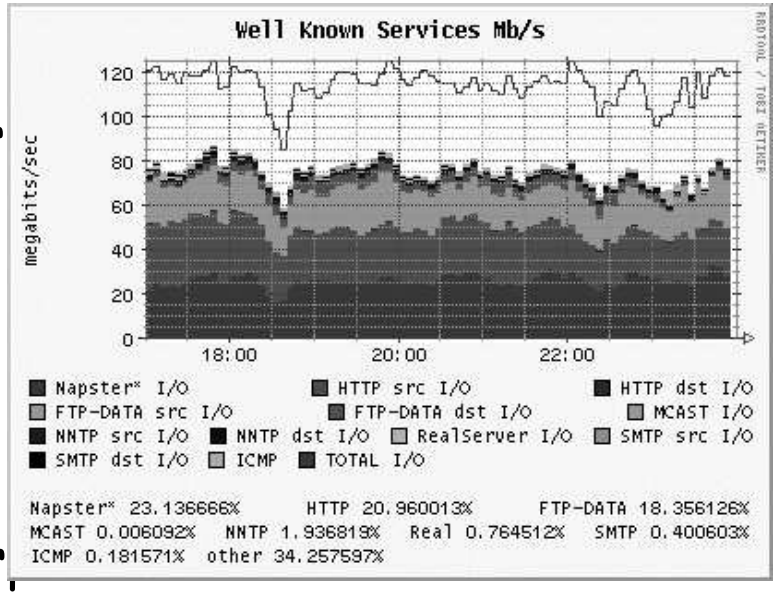
- Piracy^H^H^H^H^H^content sharing ☺
- Napster
- Gnutella
- Chord
- etc

1. NAPSTER

- The most (in)famous
- Not the first (c.f. probably Eternity, from Ross Anderson in Cambridge)
- But instructive for what it gets right, and
- Also wrong…
- Also has a political message…and economic and legal…etc etc etc

Napster

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



Napster: how does it work

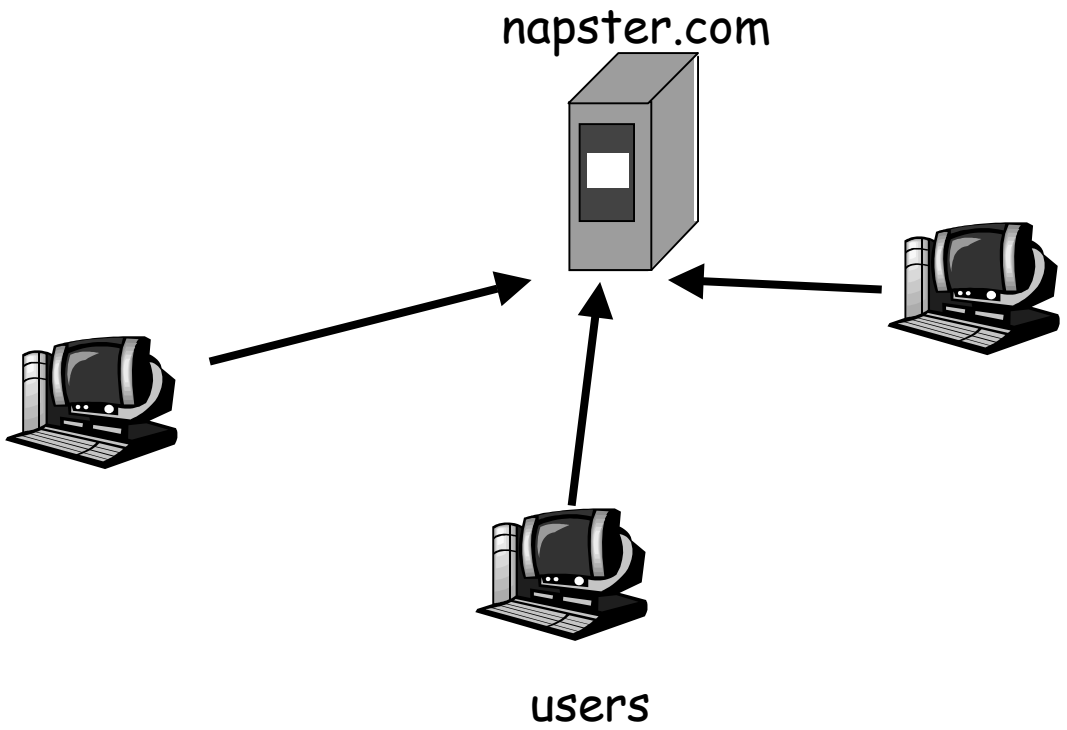
Application-level, client-server protocol over point-to-point TCP

Four steps:

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

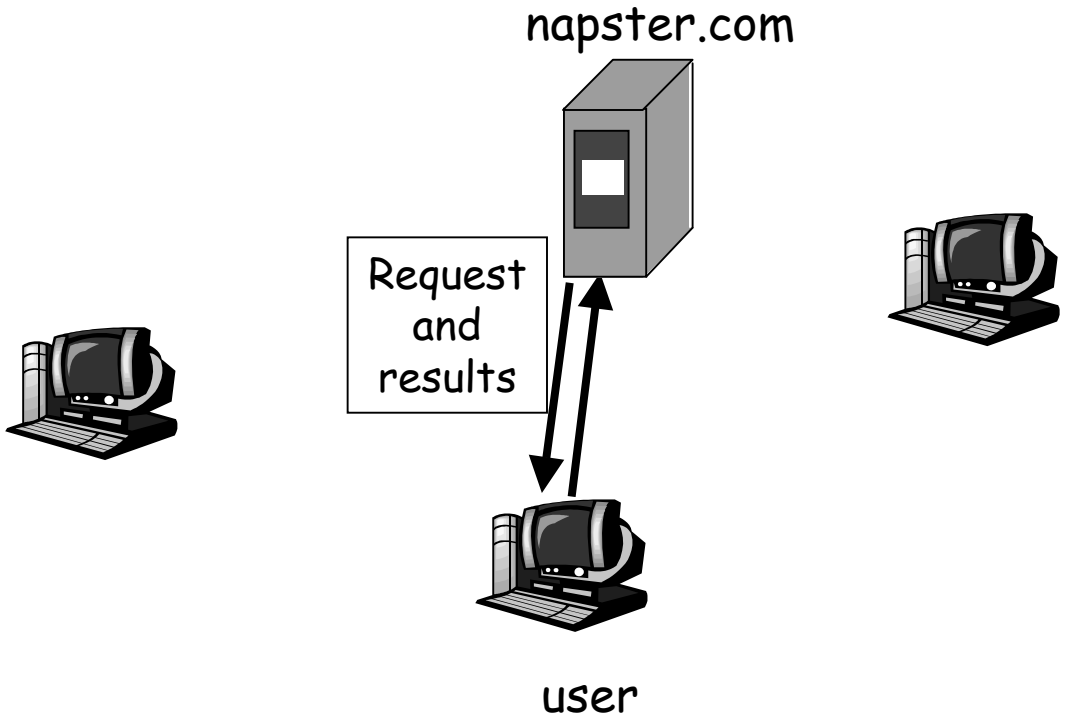
Napster

1. File list is uploaded



Napster

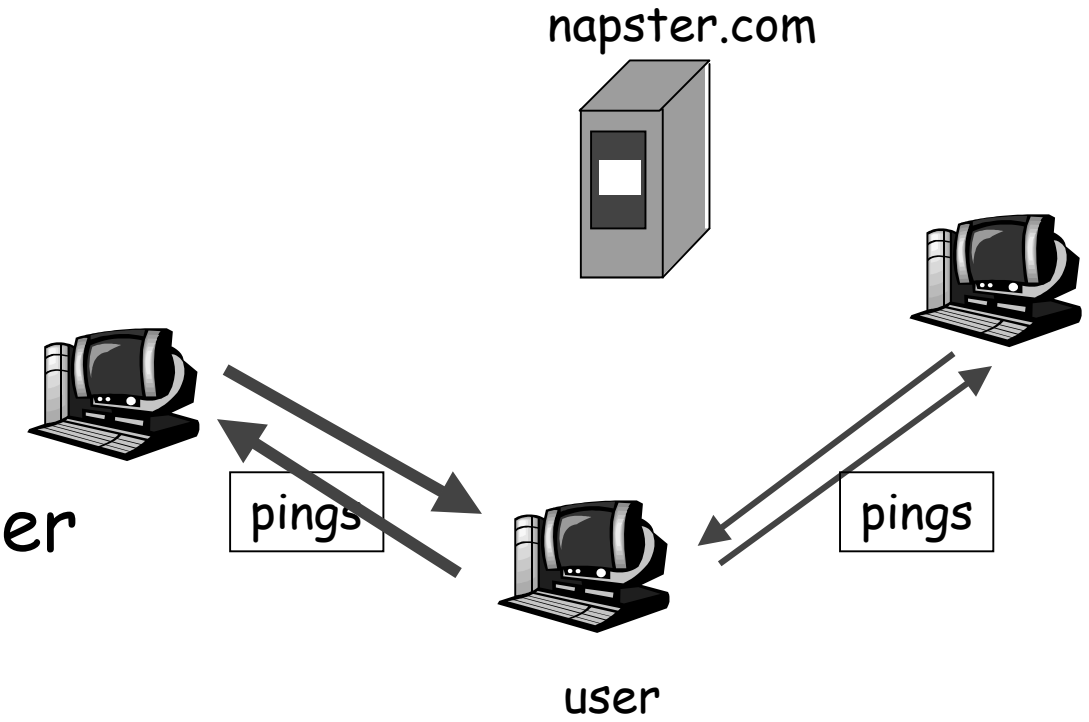
2. User requests search at server.



Napster

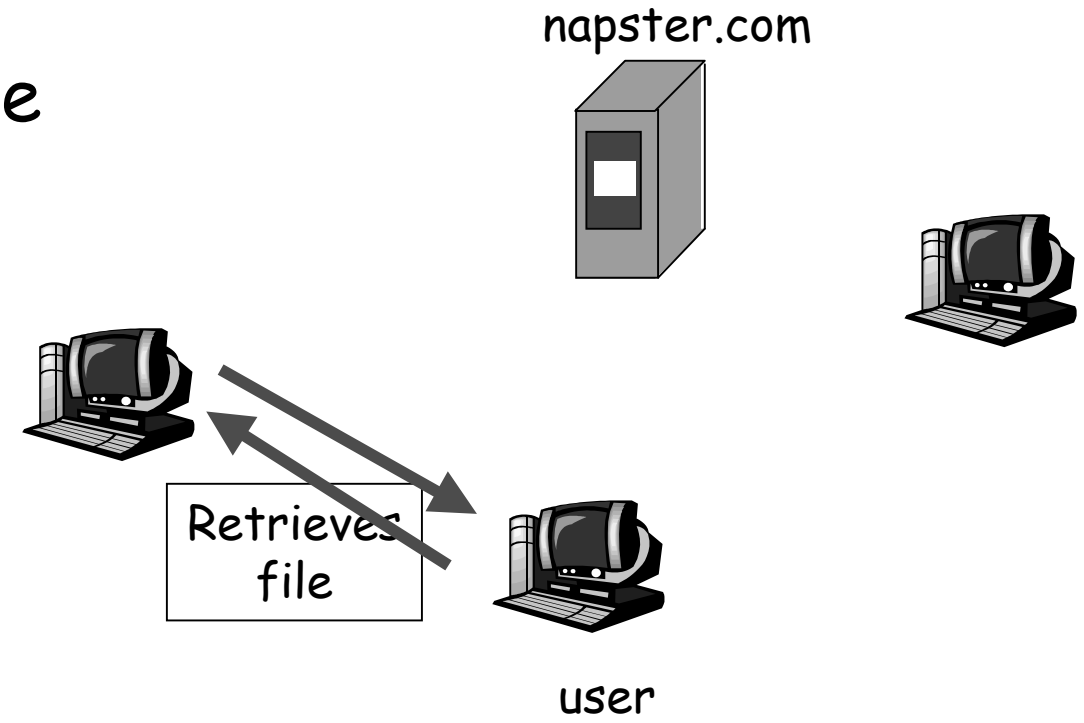
3. User pings hosts that apparently have data.

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



Napster

4. User retrieves file



Napster: architecture notes

- centralized server:
  - single logical point of failure
  - can load balance among servers using DNS rotation
  - potential for congestion
  - Napster "in control" (freedom is an illusion)
- no security:
  - passwords in plain text
  - no authentication
  - no anonymity

2 Gnutella

- Napster fixed
- Open Source
- Distributed
- Still very political…

Gnutella

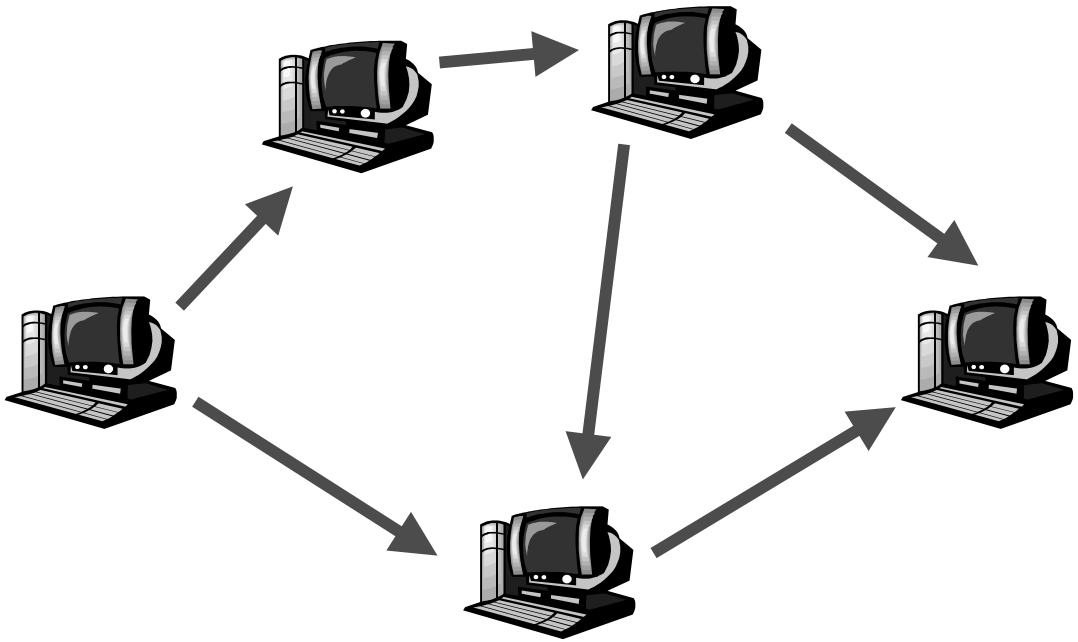
- peer-to-peer networking: applications connect to peer applications
- focus: decentralized method of searching for files
- each application instance serves to:
  - store selected files
  - route queries (file searches) from and to its neighboring peers
  - respond to queries (serve file) if file stored locally
- Gnutella history:
  - 3/14/00: release by AOL, almost immediately withdrawn
  - too late: 23K users on Gnutella at 8 am this AM
  - many iterations to fix poor initial design (poor design turned many people off)

Gnutella: how it works

Searching by flooding:

- If you don't have the file you want, query 7 of your partners.
- If they don't have it, they contact 7 of their partners, for a maximum hop count of 10.
- Requests are flooded, but there is no tree structure.
- No looping but packets may be received twice.
- What we care about:
  - 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?

Flooding in Gnutella: loop prevention



Seen already list: "A"

Gnutella: initial problems and fixes

- Freeloading: WWW sites offering search/retrieval from Gnutella network without providing file sharing or query routing.
  - Block file-serving to browser-based non-file-sharing users
- Prematurely terminated downloads:
  - long download times over modems
  - modem users run gnutella peer only briefly (Napster problem also!) or any users becomes overloaded
  - fix: peer can reply "I have it, but I am busy. Try again later"
  - late 2000: only 10% of downloads succeed
  - 2001: more than 25% downloads successful (is this success or failure?)

www.limewire.com/index.jsp/net_improvements

Gnutella: initial problems and fixes (more)

- 2000: avg size of reachable network ony 400-800 hosts. Why so smalll?
  - modem users: not enough bandwidth to provide search routing capabilities: routing black holes
- Fix: create peer hierarchy based on capabilities
  - previously: all peers identical, most modem blackholes
  - connection preferencing:
    - favors routing to well-connected peers
    - favors reply to clients that themselves serve large number of files: prevent freeloading
  - Limewire gateway functions as Napster-like central server on behalf of other peers (for searching purposes)

Anonymous?

- Not anymore than it's scalable.
- The person you are getting the file from knows who you are. That's not anonymous.

- Other protocols exist where the owner of the files doesn't know the requester.

- Peer-to-peer anonymity exists.
- See Eternity and Freenet! For the terminally enthusiastic (or paranoid!)

Gnutella Discussion:

- Architectural lessons learned?
- Do Gnutella's goals seem familiar? Does it work better than say squid or summary cache? Or multicast with carousel?
- anonymity and security?
- Other?
- Good source for technical info/open questions:

  http://www.limewire.com/index.jsp/tech_papers

Lecture 3: Distributed Hash Tables

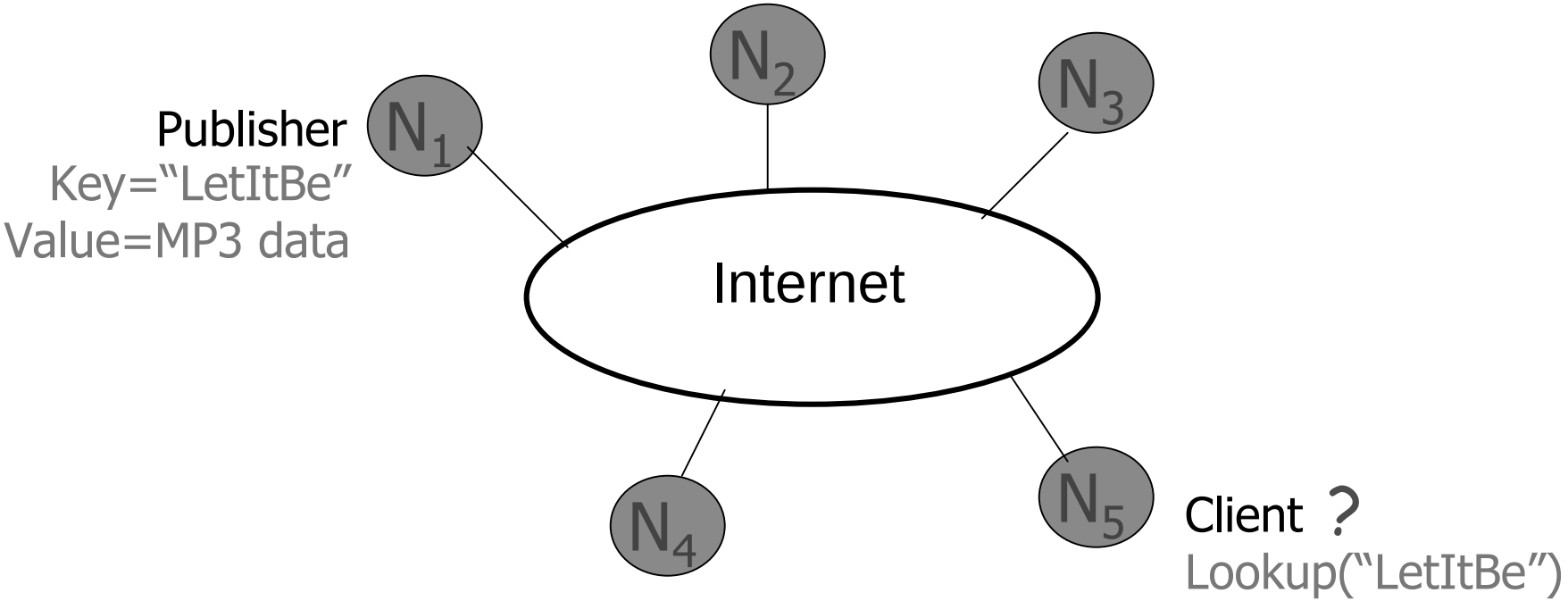
- Can we go from content to location in one go?
- Can we still retain locality?
- Can we keep any anonimity
- Look at Chord
- Tapestry, CAN, Pastry are similar projects
- Notice how networking people like silly names☺

Outline for Chord

- Motivation and background
- Consistency caching
- Chord
- Performance evaluation
- Conclusion and discussion

Motivation

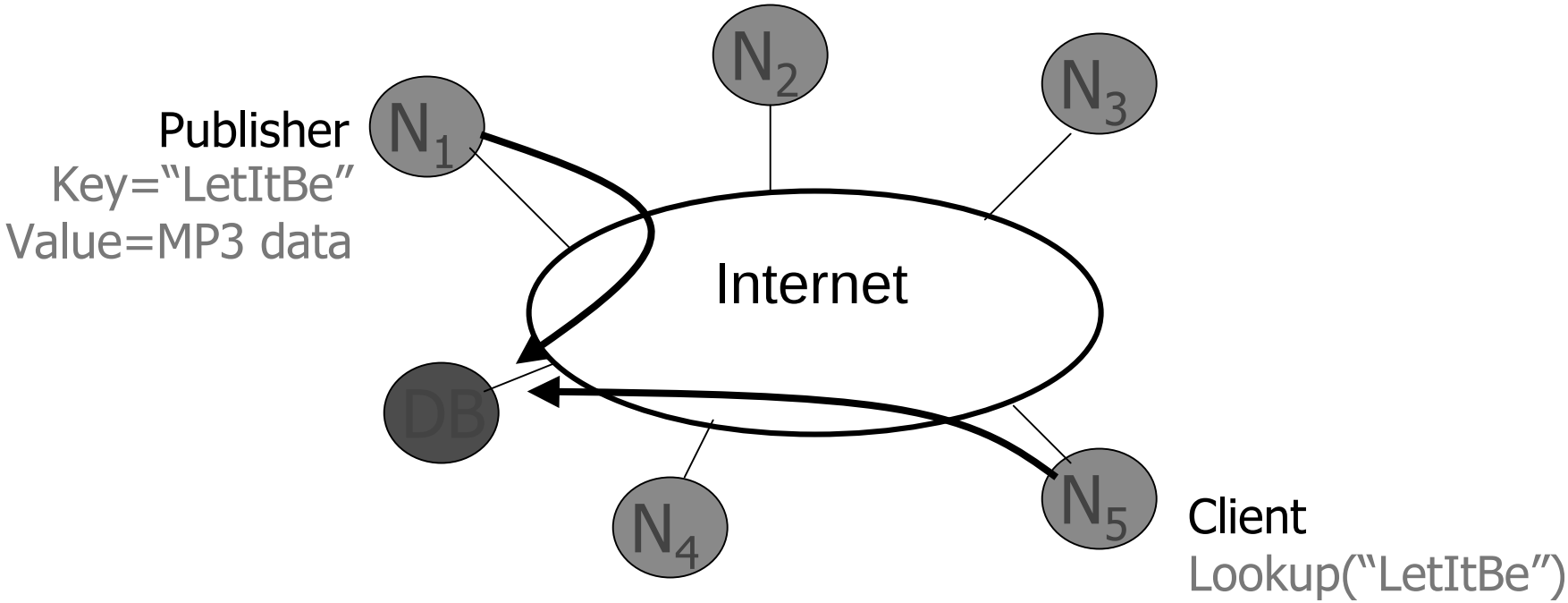
How to find data in a distributed file sharing system?



- Lookup is the key problem

Centralized Solution

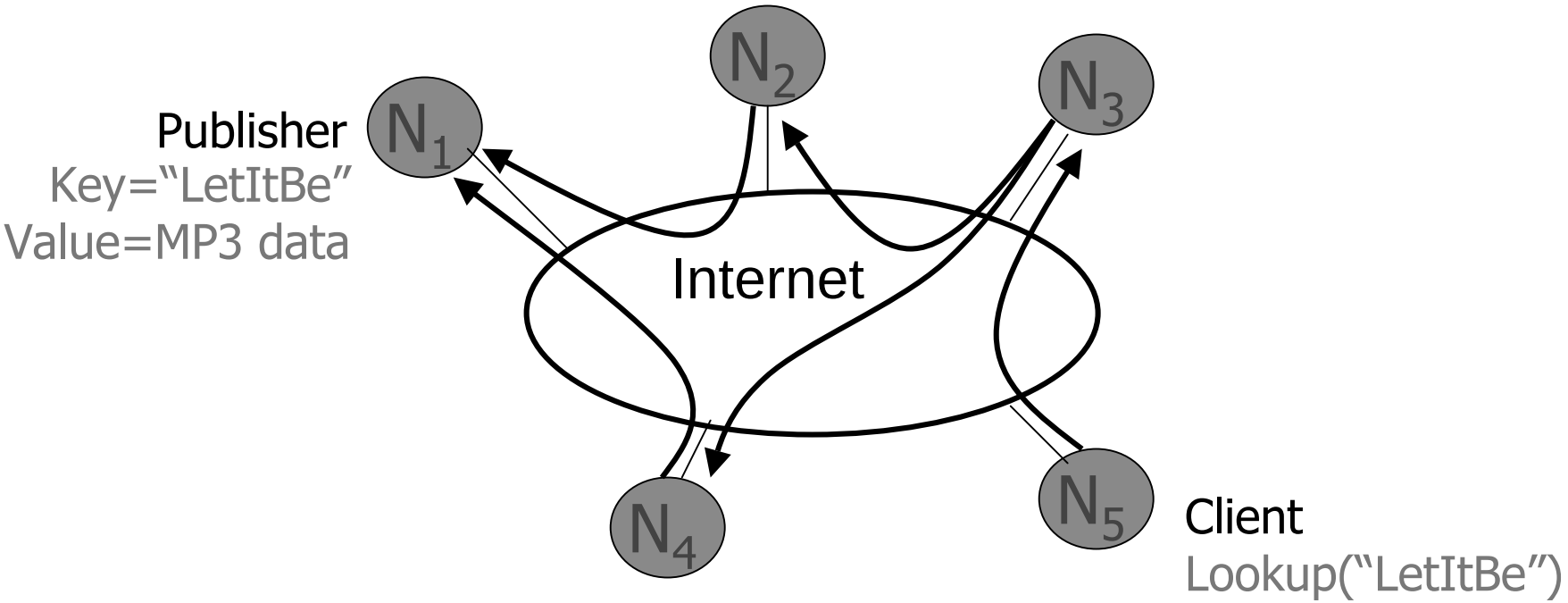
- Central server (Napster)



- Requires O(M) state
- Single point of failure

Distributed Solution (1)

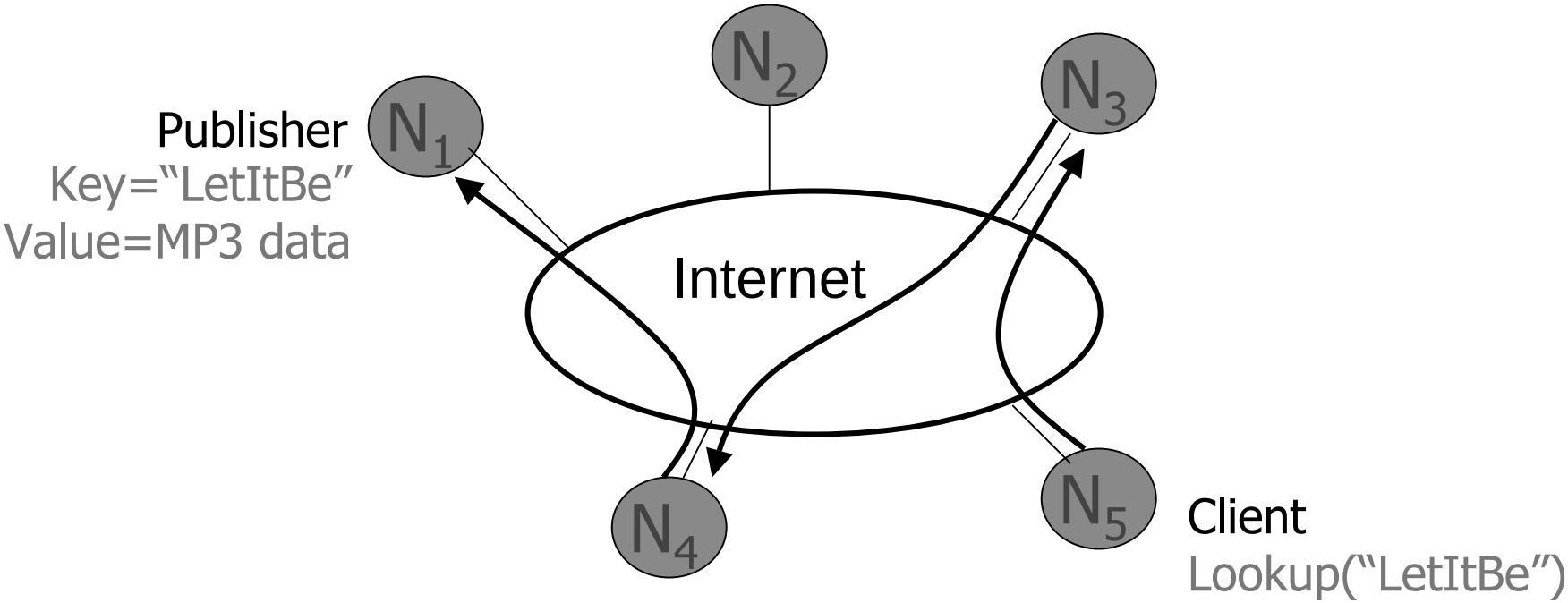
- Flooding (Gnutella, Morpheus, etc.)



- Worst case O(N) messages per lookup

Distributed Solution (2)

- Routed messages (Freenet, Tapestry, Chord, CAN, etc.)



- Only exact matches

Routing Challenges

- Define a useful key nearness metric
- Keep the hop count small
- Keep the routing tables "right size"
- Stay robust despite rapid changes in membership

Authors claim:

- Chord: emphasizes efficiency and simplicity

Chord Overview

- Provides peer-to-peer hash lookup service:
  - Lookup(key) → IP address
  - Chord does not store the data
- How does Chord locate a node?
- How does Chord maintain routing tables?
- How does Chord cope with changes in membership?

Chord properties

- Efficient: O(Log N) messages per lookup
  - N is the total number of servers
- Scalable: O(Log N) state per node
- Robust: survives massive changes in membership

- Proofs are in paper / tech report
  - Assuming no malicious participants

Chord IDs

- *m* bit identifier space for both keys and nodes
- Key identifier = SHA-1(key)

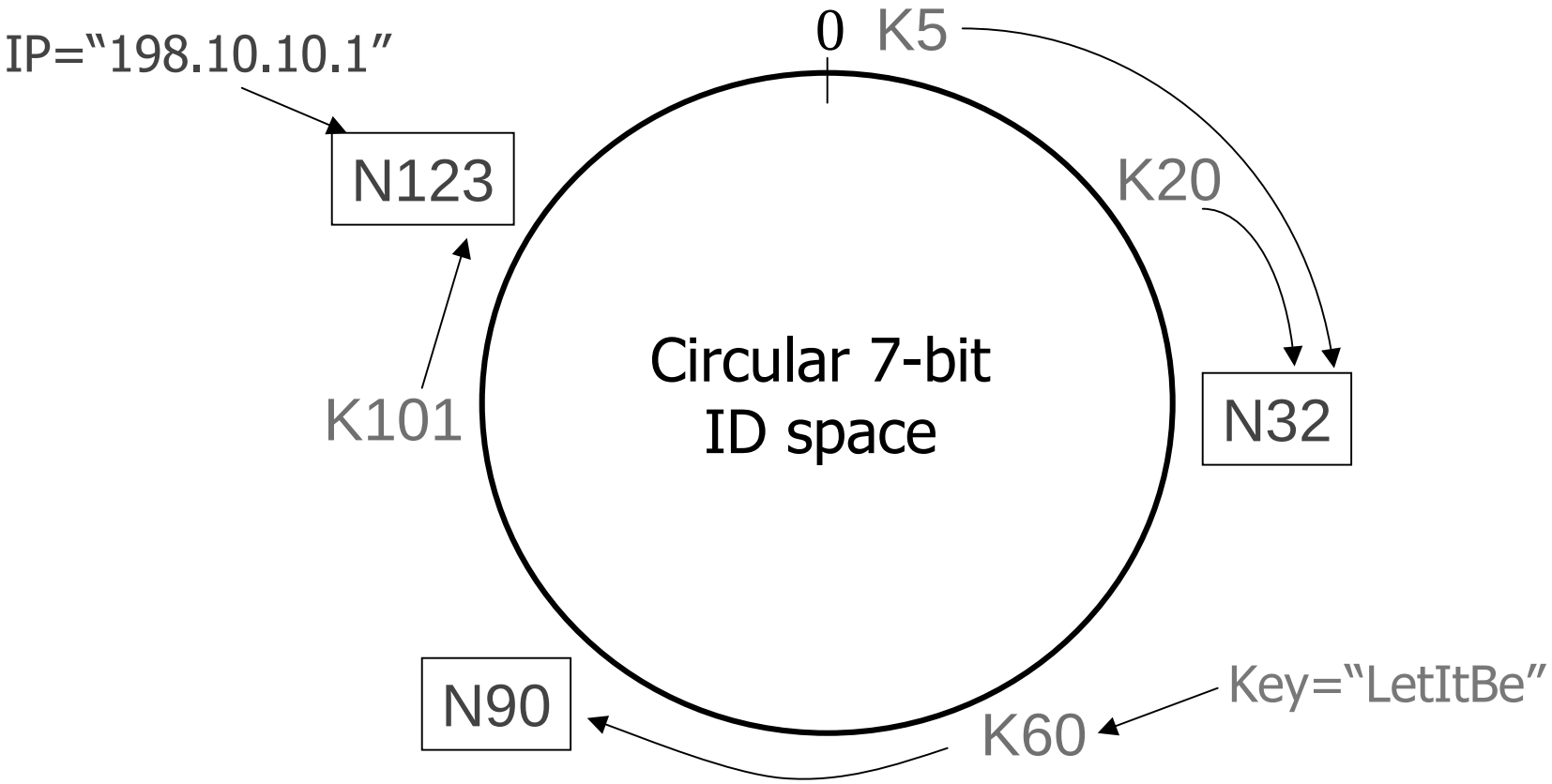
Key="LetItBe" $\xrightarrow{\text{SHA-1}}$ ID=60

- Node identifier = SHA-1(IP address)

IP="198.10.10.1" $\xrightarrow{\text{SHA-1}}$ ID=123

- Both are uniformly distributed
- How to map key IDs to node IDs?

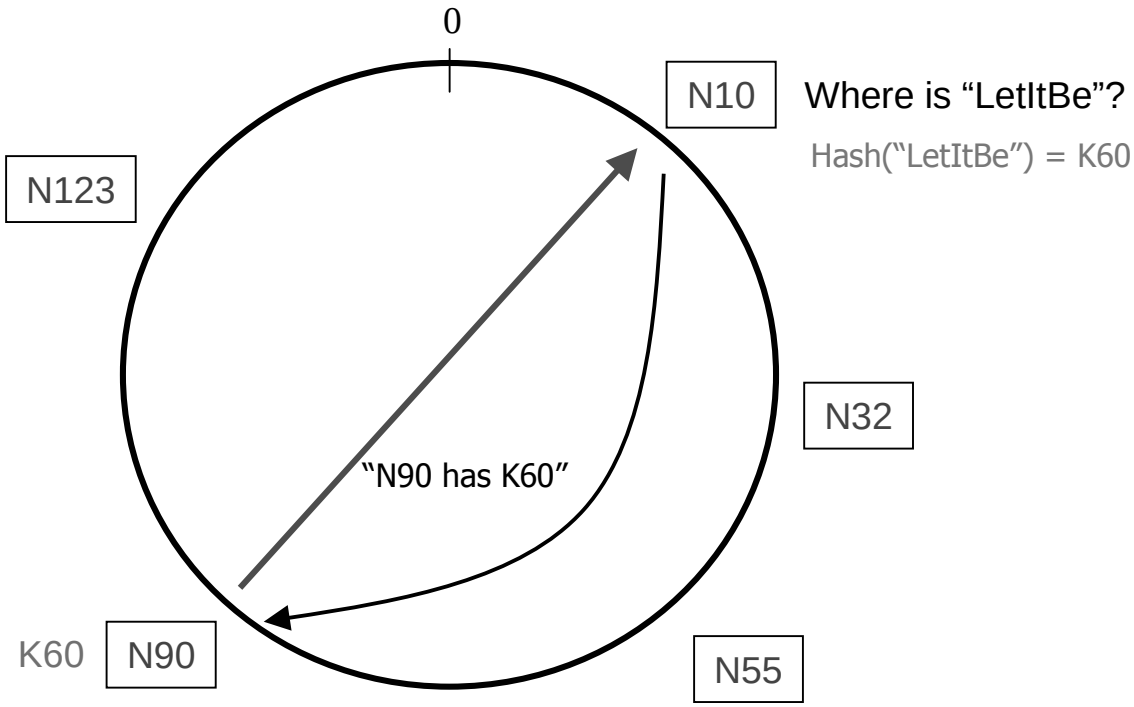
Consistent Hashing [Karger 97]



- A key is stored at its successor: node with next higher ID

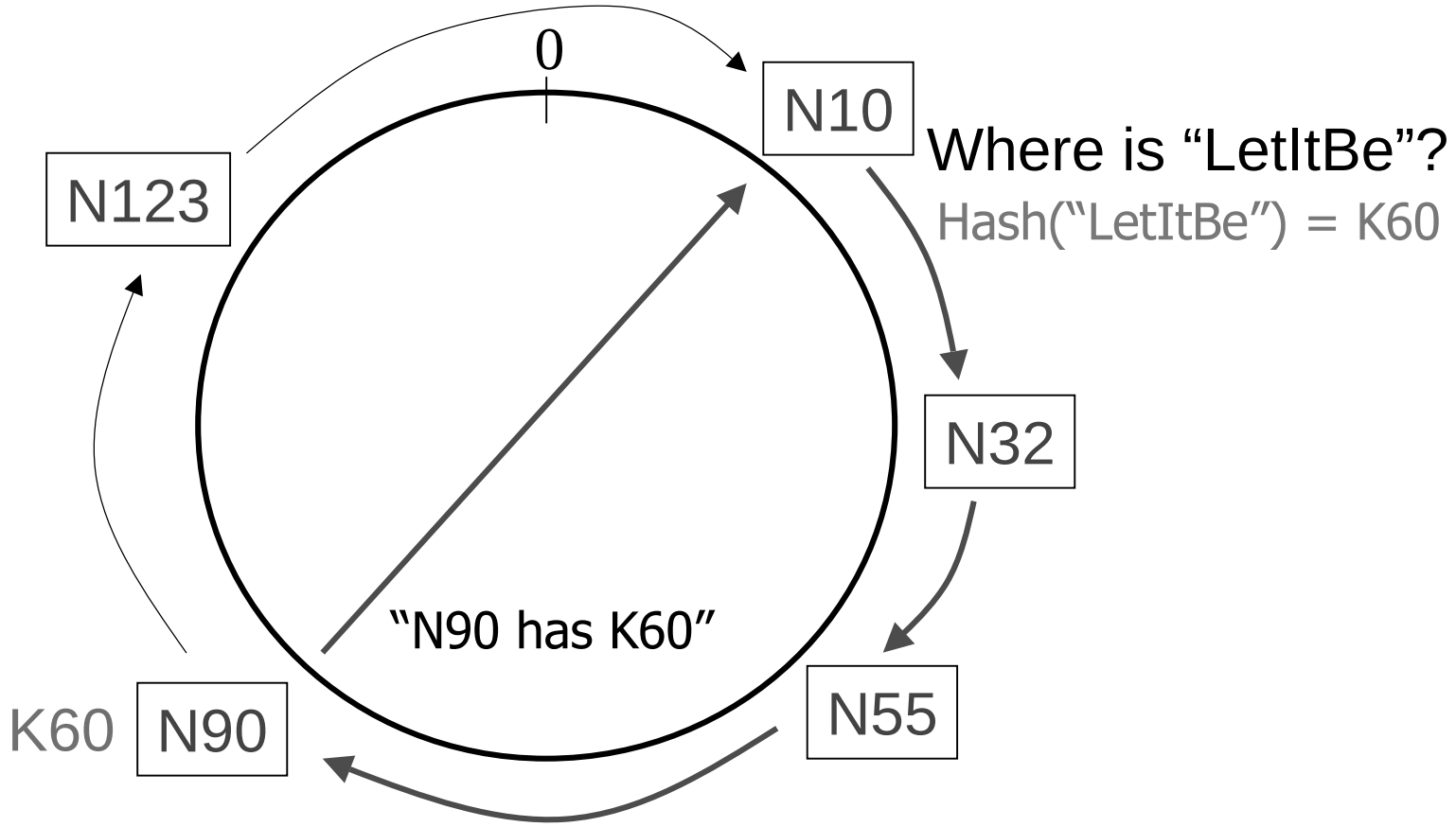
Consistent Hashing

- Every node knows of every other node
  - requires global information
- Routing tables are large O(N)
- Lookups are fast O(1)



Chord: Basic Lookup

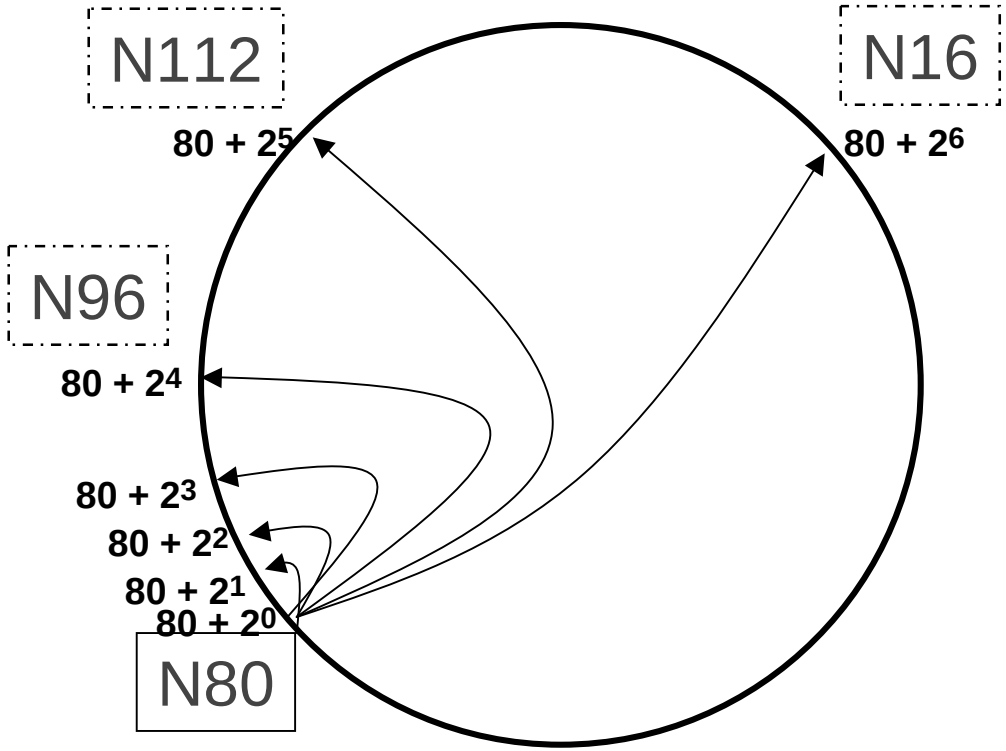
- Every node knows its successor in the ring



- requires O(N) time

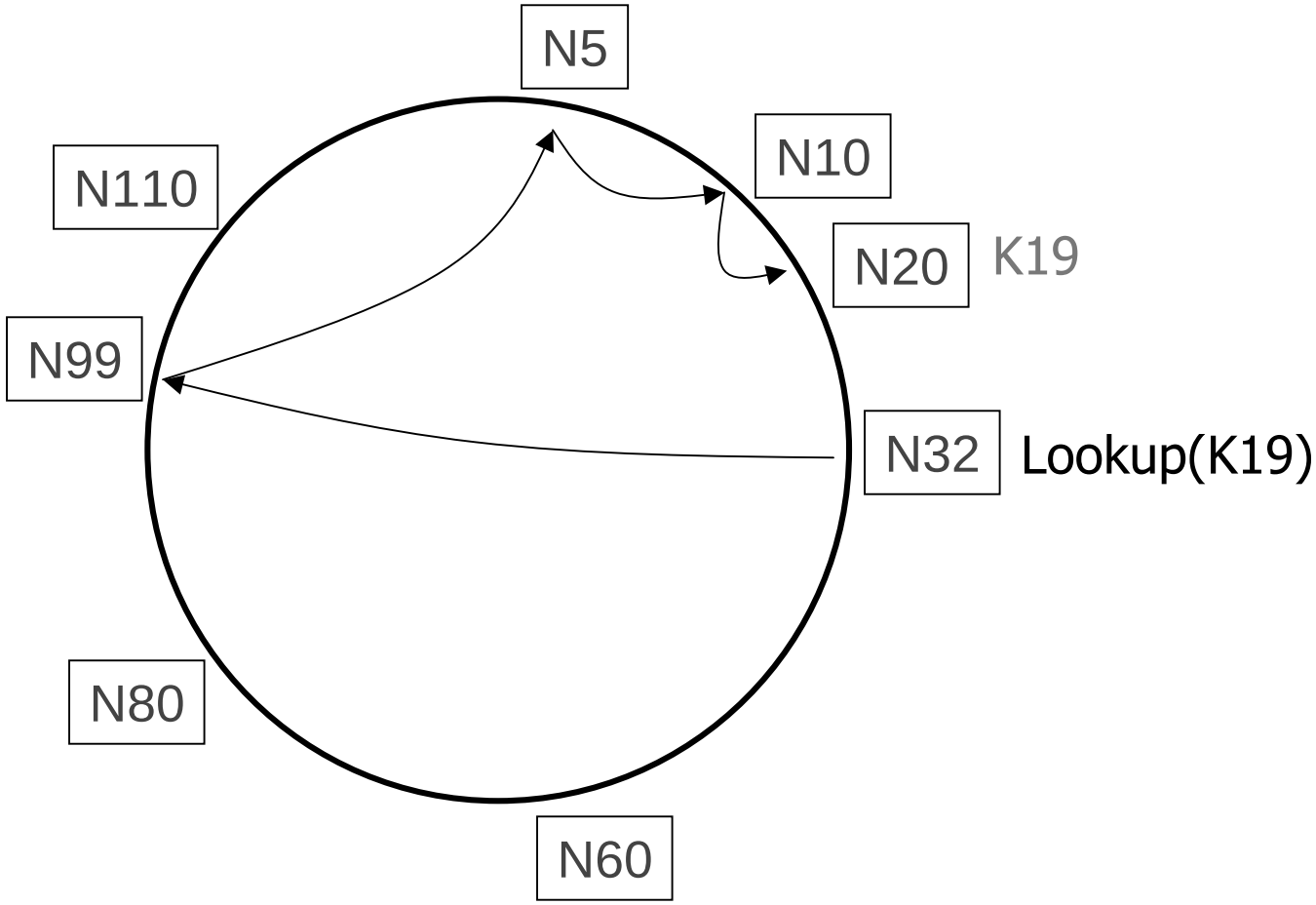
"Finger Tables"

- Every node knows *m* other nodes in the ring
- Increase distance exponentially



Lookups are Faster

- Lookups take O(Log N) hops



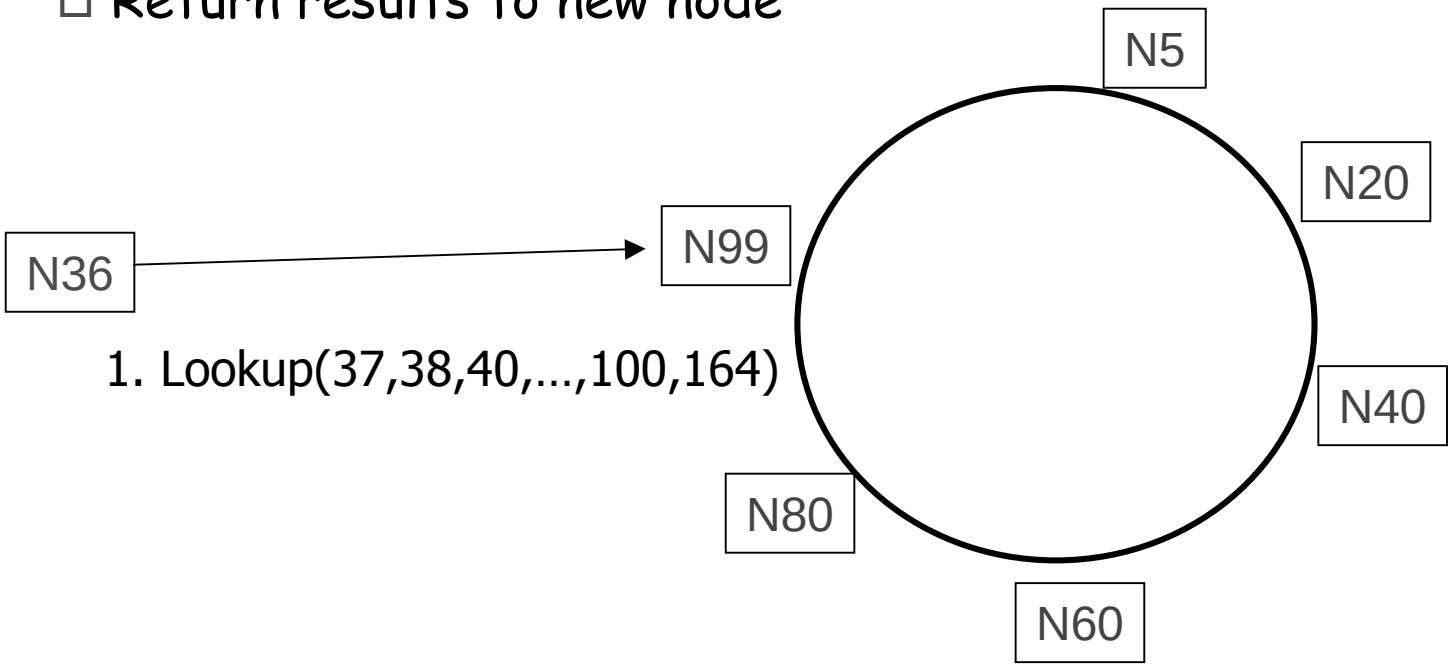
Joining the Ring

- Three step process:
  - Initialize all fingers of new node
  - Update fingers of existing nodes
  - Transfer keys from successor to new node

- Less aggressive mechanism (lazy finger update):
  - Initialize only the finger to successor node
  - Periodically verify immediate successor, predecessor
  - Periodically refresh finger table entries

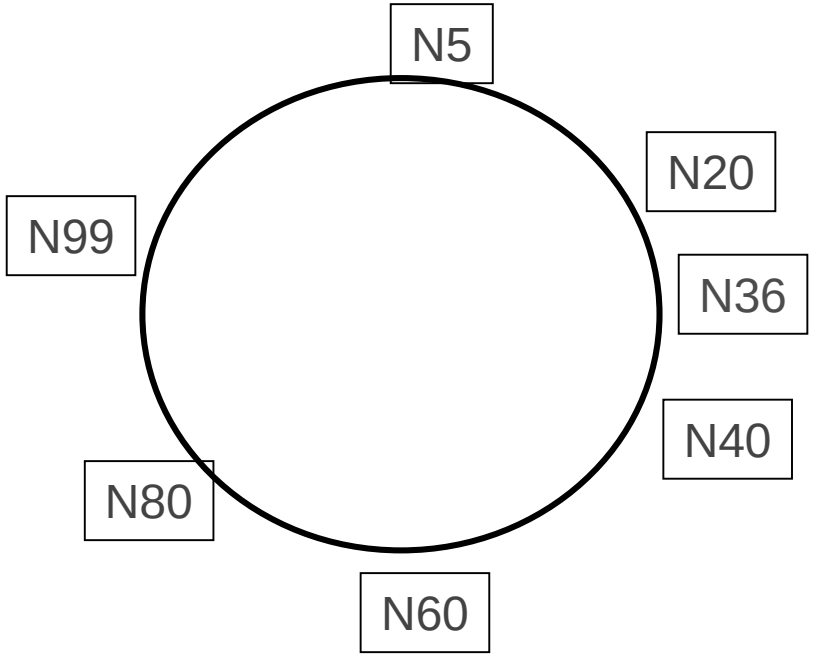
Joining the Ring - Step 1

- Initialize the new node finger table
  - Locate any node *p* in the ring
  - Ask node *p* to lookup fingers of new node N36
  - Return results to new node



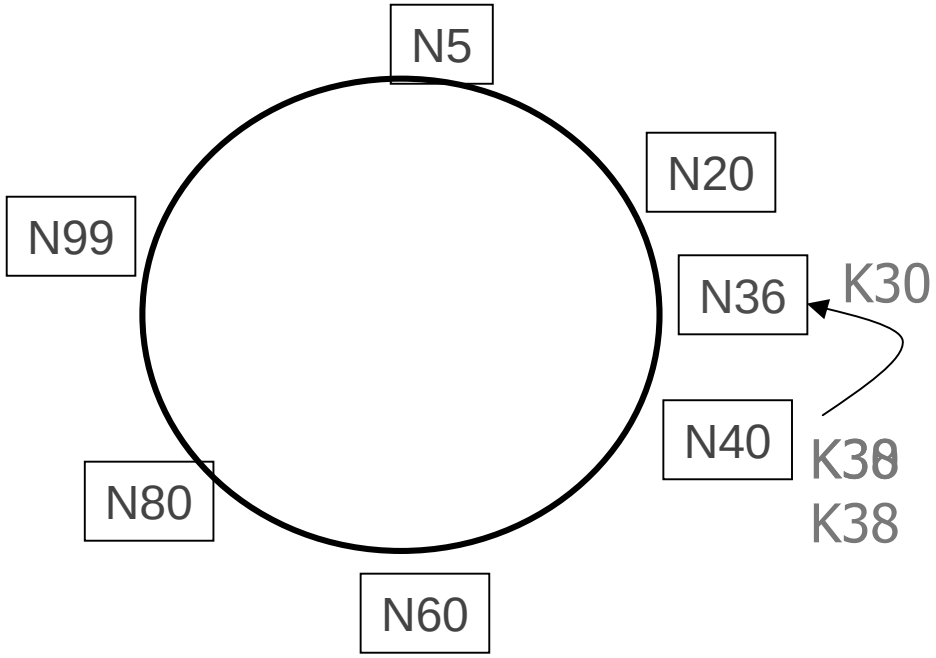
Joining the Ring - Step 2

- Updating fingers of existing nodes
  - new node calls update function on existing nodes
  - existing nodes can recursively update fingers of other nodes



Joining the Ring - Step 3

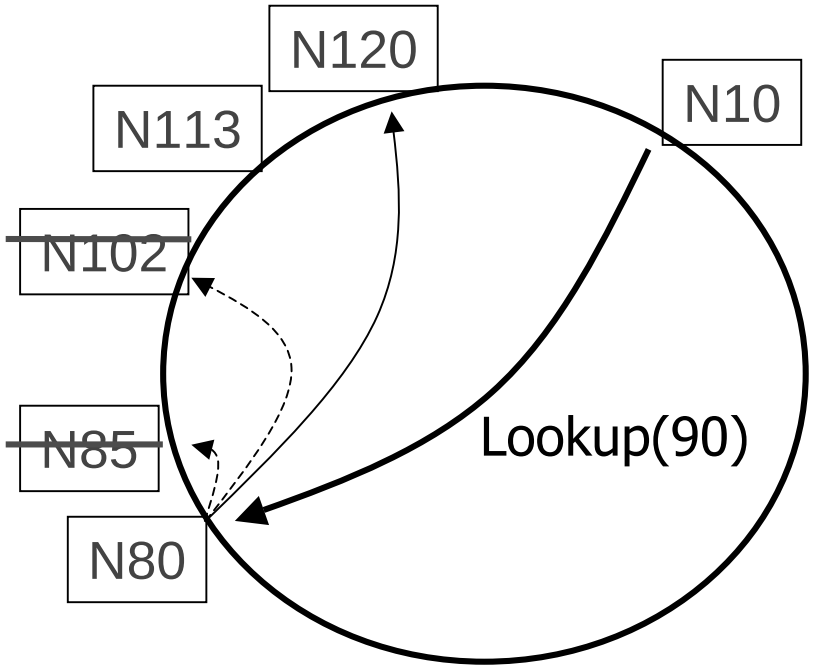
- Transfer keys from successor node to new node
  - only keys in the range are transferred



Copy keys 21..36
from N40 to N36

Handing Failures

- Failure of nodes might cause incorrect lookup



- N80 doesn't know correct successor, so lookup fails
- Successor fingers are enough for correctness

Handling Failures

- Use successor list
  - Each node knows *r* immediate successors
  - After failure, will know first live successor
  - Correct successors guarantee correct lookups

- Guarantee is with some probability
  - Can choose *r* to make probability of lookup failure arbitrarily small

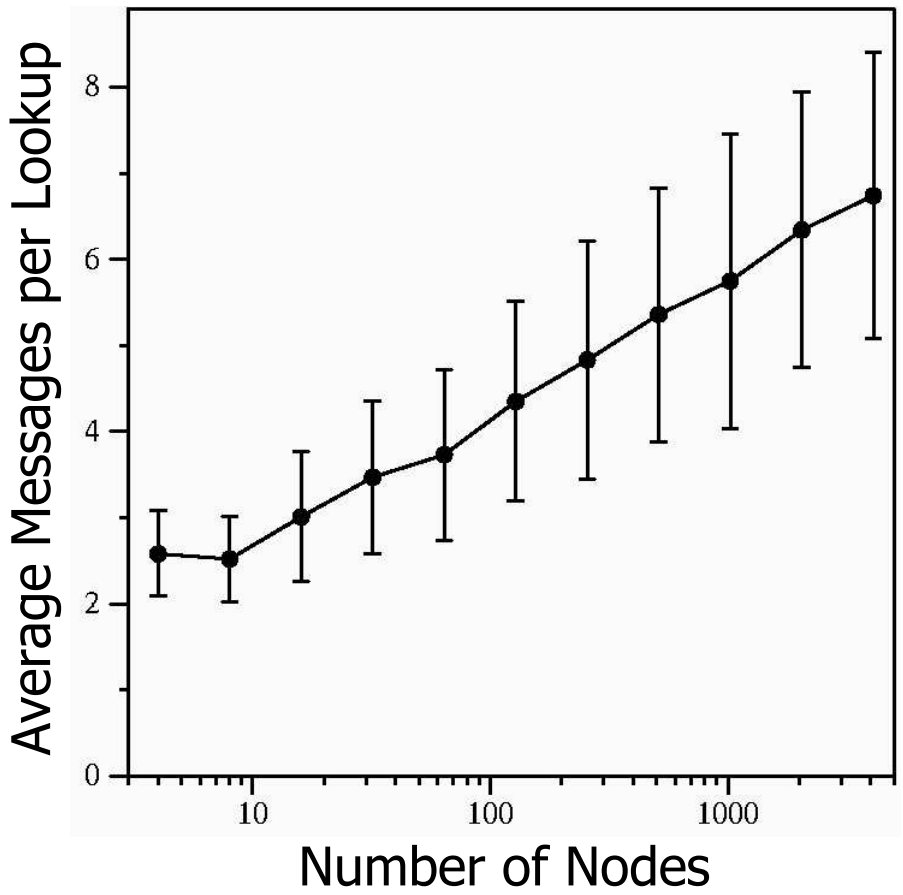
Evaluation Overview

- Quick lookup in large systems
- Low variation in lookup costs
- Robust despite massive failure

- Experiments confirm theoretical results

Cost of lookup

- Cost is O(Log N) as predicted by theory
- constant is 1/2



Current implementation

- Chord library: 3,000 lines of C++
- Deployed in small Internet testbed
- Includes:
  - Correct concurrent join/fail
  - Proximity-based routing for low delay (?)
  - Load control for heterogeneous nodes (?)
  - Resistance to spoofed node IDs (?)

Strengths

- Based on theoretical work (consistent hashing)
- Proven performance in many different aspects
  - "with high probability" proofs
- Robust (Is it?)

Weakness

- **NOT** that simple (compared to CAN)
- Member joining is complicated
  - aggressive mechanisms requires too many messages and updates
  - no analysis of convergence in lazy finger mechanism
- Key management mechanism mixed between layers
  - upper layer does insertion and handle node failures
  - Chord transfer keys when node joins (no leave mechanism!)
- Routing table grows with # of members in group
- Worst case lookup can be slow

Discussions

- Network proximity (consider latency?)
- Protocol security
  - Malicious data insertion
  - Malicious Chord table information
- Keyword search and indexing
- ...

Wrapup discussion questions:

- Is ad hoc networking a peer-peer application?
  - Yes (30-1)
- Why peer-peer over client-server?
  - A well-deigned p2p provides better "scaability"
- Why client-server of peer-peer
  - peer-peer is harder to make reliable
  - availability different from client-server (p2p is more often at least partially "up")
  - more trust is required
- If all music were free in the future (and organized), would we have peer-peer.
  - Is there another app: ad hoc networking, any copyrighted data, peer-peer sensor data gathering and retrieval, simulation
- Evolution #101 – what can we learn about systems?