

Peer to Peer Systems and Chord

Amin Vahdat

CSE 123b

June 8, 2006

Presented by Jeannie Albrecht

Peer to Peer Networks

P2P Redux

- Example P2P systems
 - Napster, Gnutella, Kazaa
- Applications of P2P
 - Storage, computation, network characterization
 - Grid computing, gaming, file sharing
- Why are P2P systems gaining so much popularity?
 - Continuously running machines – cable modems, DSL, etc.
- The concept has been around for a long time
 - USENET, Internet routing (BGP)
- Is there any real need for P2P? Is the need technical?
- Business models for P2P
 - Payback for willingness to host applications

Peer-to-peer Defined

- Traditionally, network services were defined by the *client-server* model
 - Clients received from well-known services at well-known points in the network
- Peer-to-peer can be defined as “*anything, anywhere*”
 - Clients pull double duty as servers
 - All participants (peers) cooperate to deliver some service
 - “From each according to his abilities; to each according to his needs”
 - Functionality determined dynamically based on available processing power, network connectivity, content popularity, etc.

Peer-to-Peer Benefits

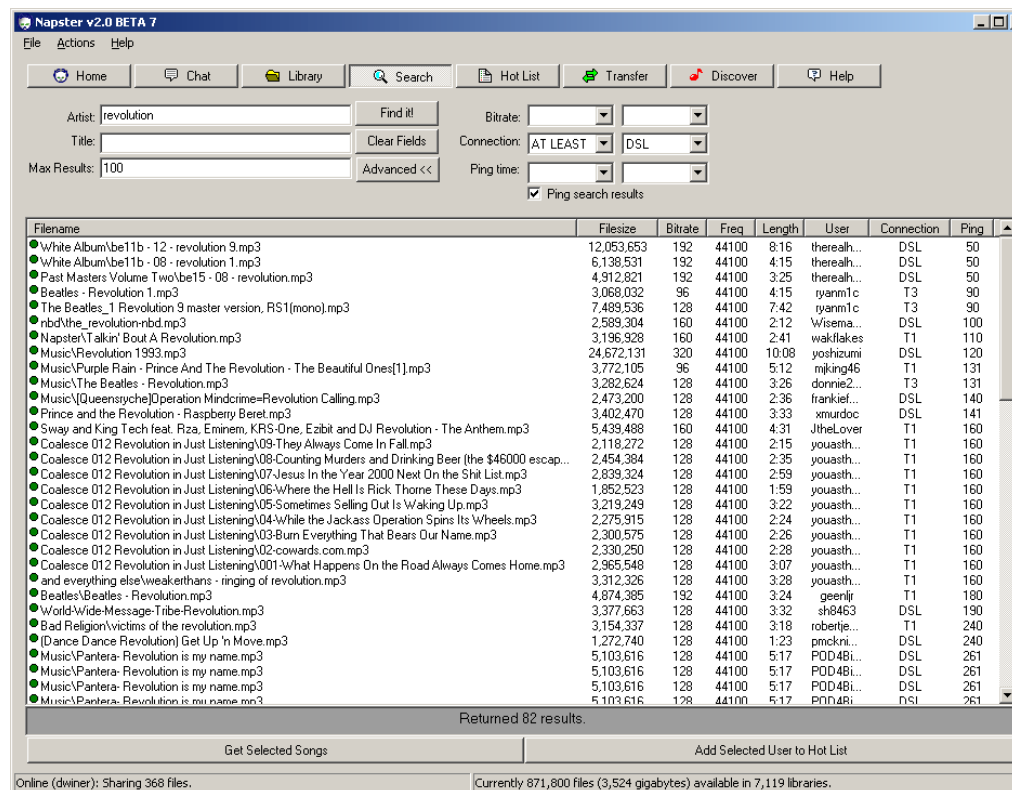
- Can spread functionality across millions of participants
 - At arbitrary point in the network
- Can replicate content across multiple participants
 - Potentially, dynamically adjust replication degree based on popularity of content
- Plan for failure as the common case
- Traditional network services fixed to a static set of locales in the network
 - Fixed available computation power and bandwidth
 - Have to plan for peaks, but difficult to predict

P2P Applications

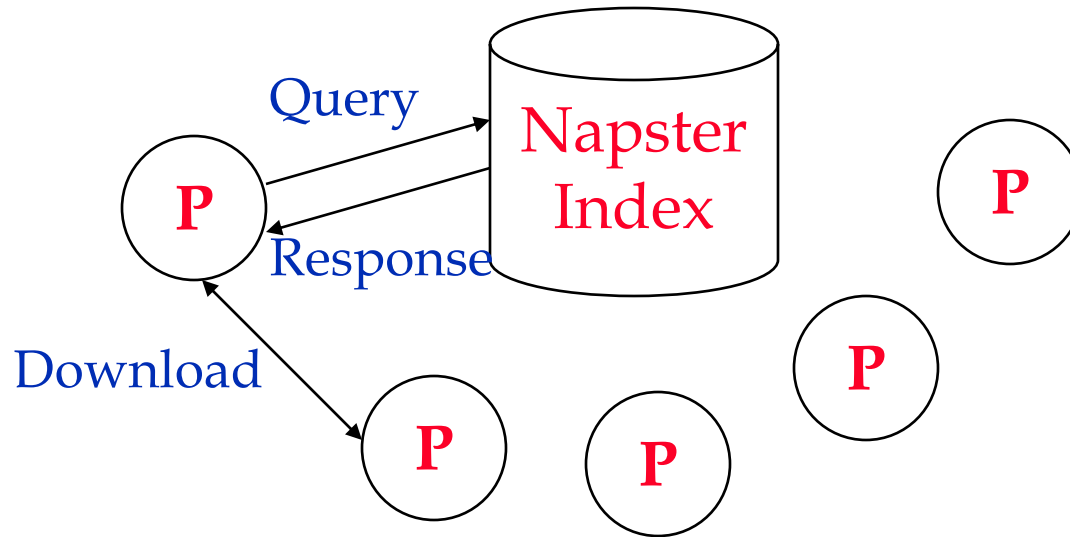
- Eternity Store
 - Research project at Berkeley: Oceanstore
- Farsite (Microsoft Research project)
 - xFS for client desktops
- Computation server
 - Seti@Home?
- Distributed index/distributed storage
 - Napster/Gnutella/Kazaa

Napster Facts

- First widely-used P2P music sharing service
- At least 26 million users at peak
- Original service was shut down by court order



Napster

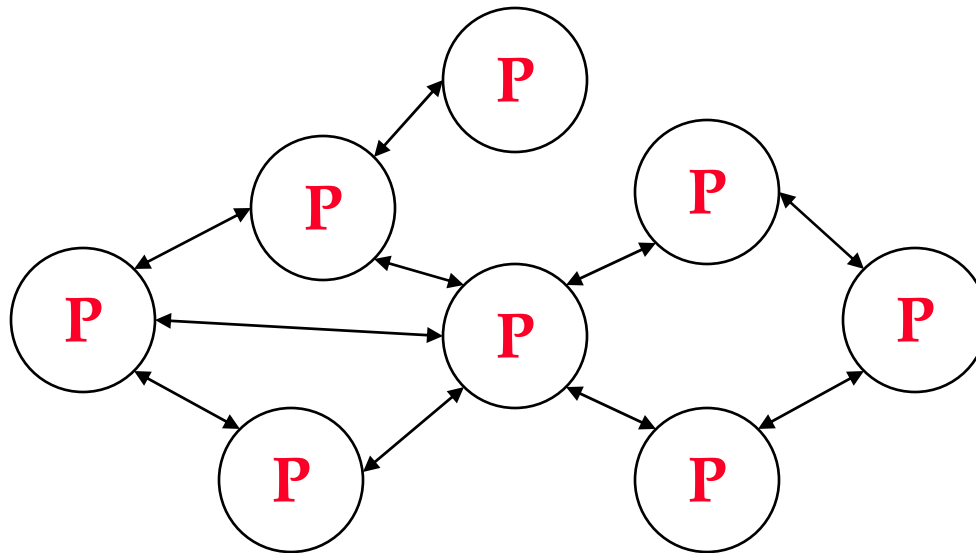


- Distributed storage, centralized index
- Which node to connect to?
 - Advertised connection speed, ping time from server
- Since it was centralized, there was single point of control

Gnutella Facts

- First released by AOL in March 2000
- AOL stopped release after 1 day due to legal concerns
- It was too late
 - Software was reverse engineered and Gnutella was born
- Gnutella is file-sharing network with many clients
 - LimeWire, FrostWire, Morpheus, Mutella, Gnucleus
- Completely decentralized, so unlike Napster it cannot be shut down
- One of most successful file sharing networks to date

Gnutella

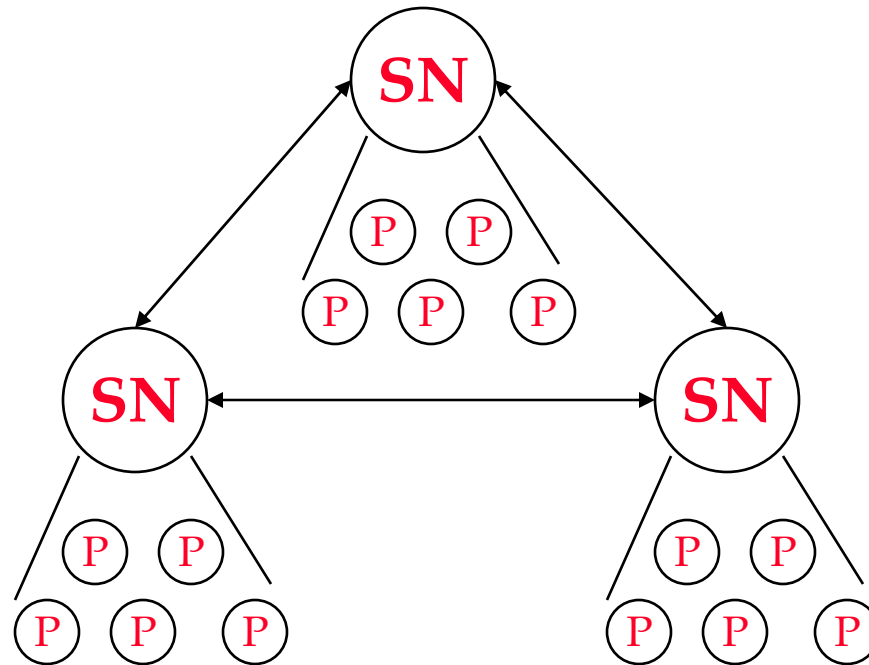


- Fully connected mesh
- Broadcast queries through the entire system
- Find just one member of the system and connect to it
- Unstable and unscalable network → ultrapeers and leaves

Kazaa Facts

- Kazaa is file sharing application that uses FastTrack protocol
- Started in March 2001
- Initial users used Morpheus
 - Morpheus later switched to Gnutella
- 3 million concurrent users
- Creators of Kazaa later created Skype

Kazaa and FastTrack



- Elect set of “supernodes” to act as regional indices
 - Important to select nodes with high bandwidth, available computation power, and storage
 - Searches for data go to supernode, which performs broadcast among all other supernodes

Anonymity, Security, Fault Tolerance

- How to ensure anonymity in lookups?
 - How would the system know who to return data to?
- What about anonymity in publishing
 - Prevent censorship
- One bad node can bring down entire peer to peer system?
- Incentive to freeload?
 - BitTorrent tries to prevent this by rewarding users for uploading
 - Users can only download if they also upload

Chord

Chord

- Goal is to build fully distributed indexing scheme
 - Given a key, map key to node
- No node has any more responsibility than any other node
- Distribute keys evenly among n nodes
 - For every request, route request to the node responsible for the key
- Every node acts as router
 - Cannot maintain state for every node in the system
 - Cannot broadcast to entire system for every lookup

Chord Properties

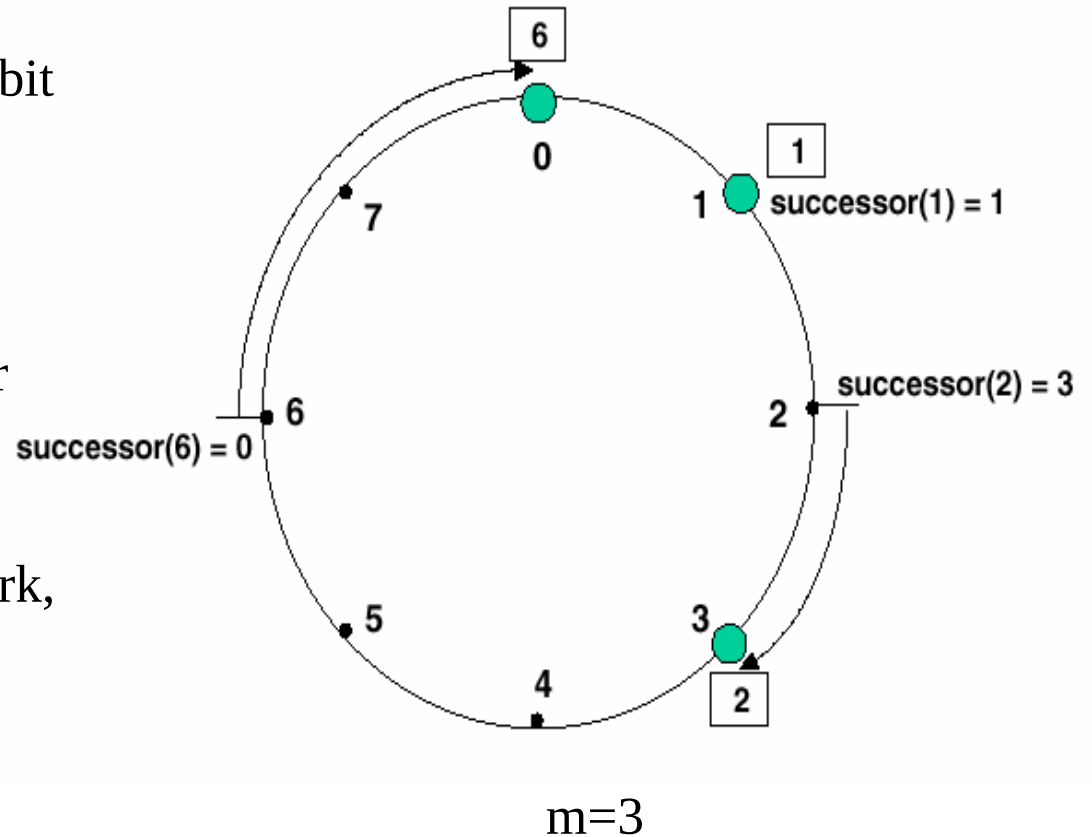
- Load balancing
 - Keys evenly distributed
- Decentralization
 - Fully distributed
 - No node has more responsibility than other nodes
- Scalability
 - Only route through $O(\lg n)$ hosts
 - For 1 billion hosts, only need 9 hops
- Availability
 - Node addition/removal handled efficiently
- Flexible Naming
 - No constraints on keys

Chord: Consistent Hashing

- Given a set of n nodes, a consistent hash function will map keys (e.g., filenames) uniformly across the nodes
- Nice feature of consistent hashing for node addition/removal:
 - Only $1/n$ keys must be reassigned to new nodes
- Original proposal required all nodes to know about most other nodes
 - Chord improves on this by requiring each node to know about $O(\lg n)$ other nodes (for good performance), $O(1)$ other nodes (for correctness—only need to know neighbors)

Chord's Identifier Circle

- Nodes and keys hashed to m -bit identifier
 - Assume keys $>$ nodes
- Assign key k to first node whose identifier is equal to or larger than k , called $successor(k)$
- When node n joins the network, certain keys assigned to $successor(n)$, now become mapped to n
 - When node n leaves the network, all of its keys get reassigned to its successor

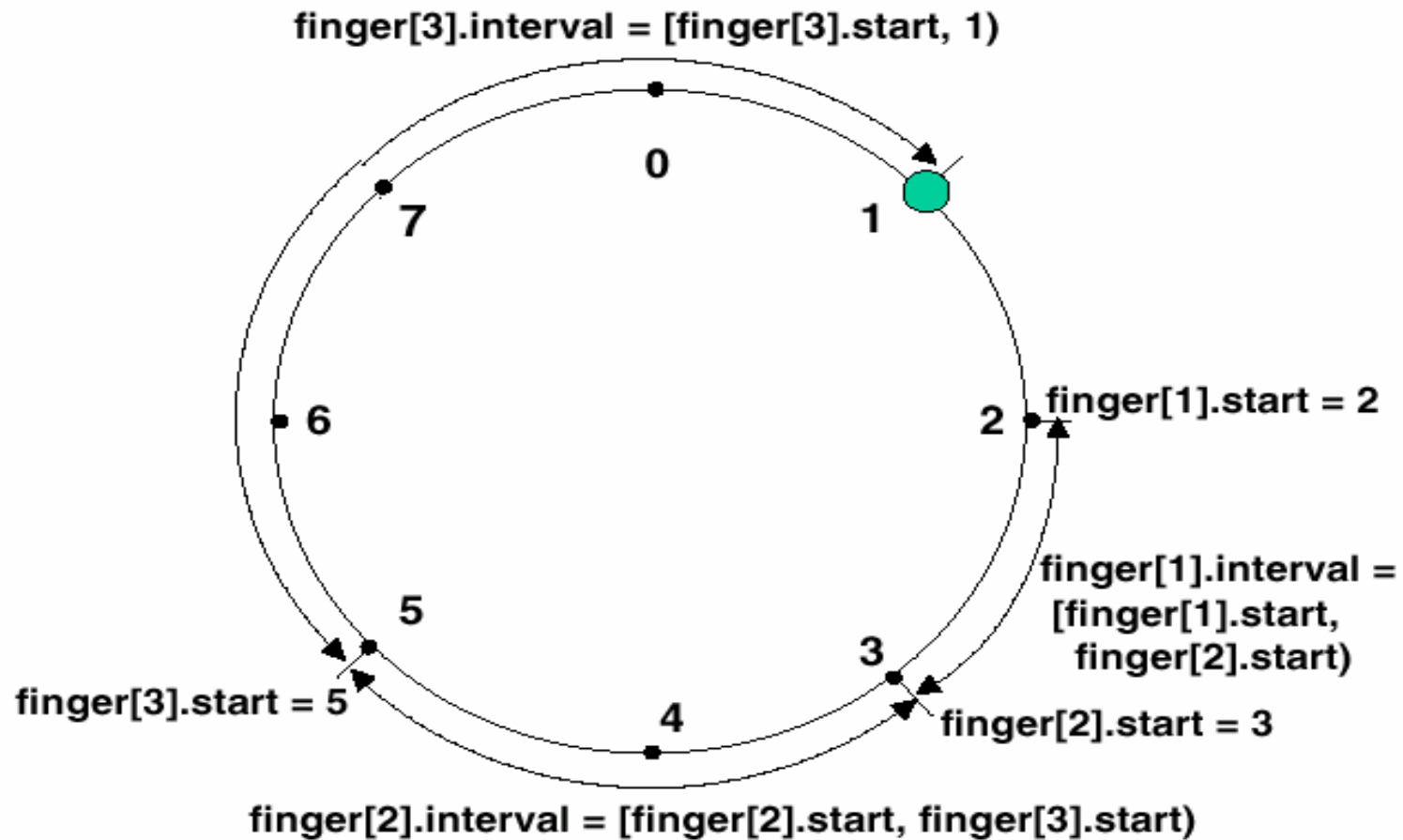


Three active nodes: 0, 1, 3
Node 1 responsible for key 1
Node 3 responsible for key 2
Node 0 responsible for key 6

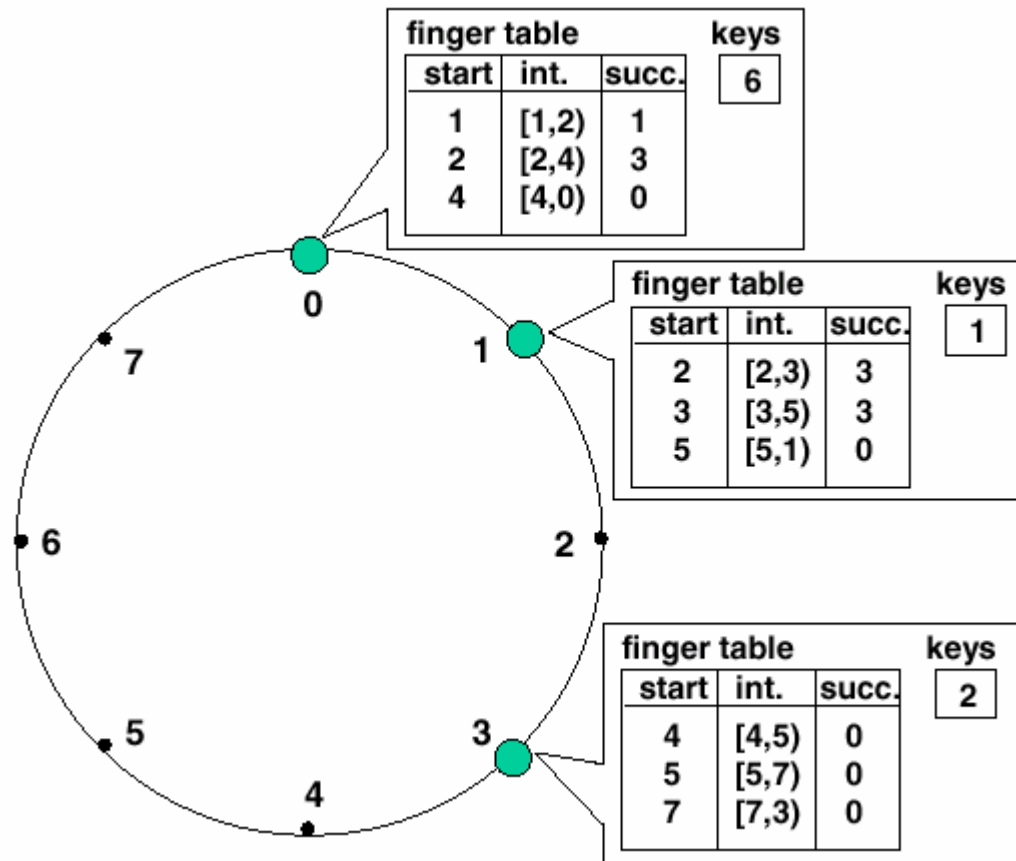
Scalable Key Location

- For correctness, each node need only maintain a pointer to its successor
 - Sufficient information to route requests to appropriate node
 - However, $O(n)$ hops to locate data → does not scale
- Each node maintains *finger table*
 - m entries in table, 1 for each bit in identifier
 - Entry i at node n contains ip addr/port for first node s , that succeeds n by at least 2^{i-1}
 - So first entry is the $\text{successor}(n)$

Finger Table



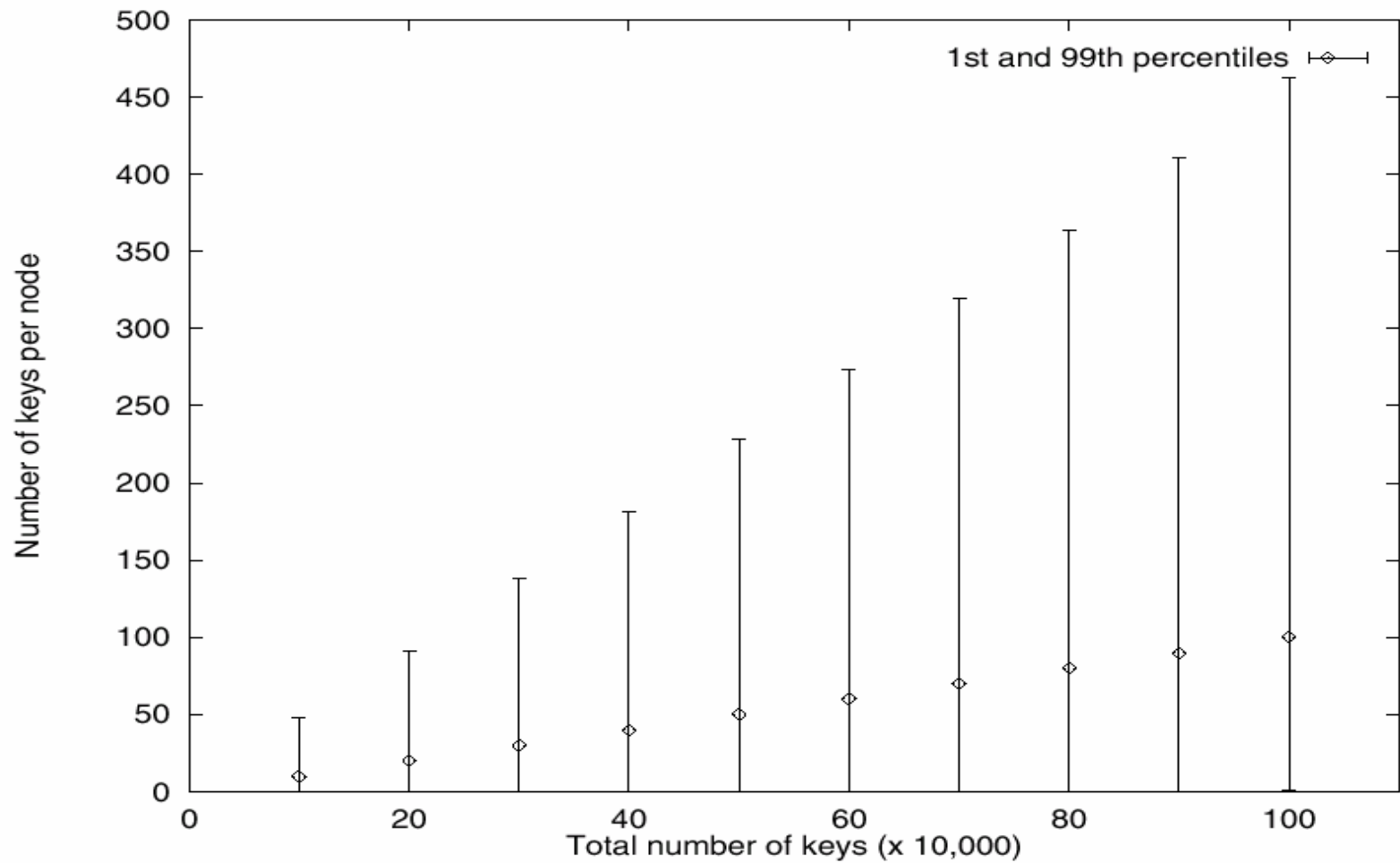
Finger Table



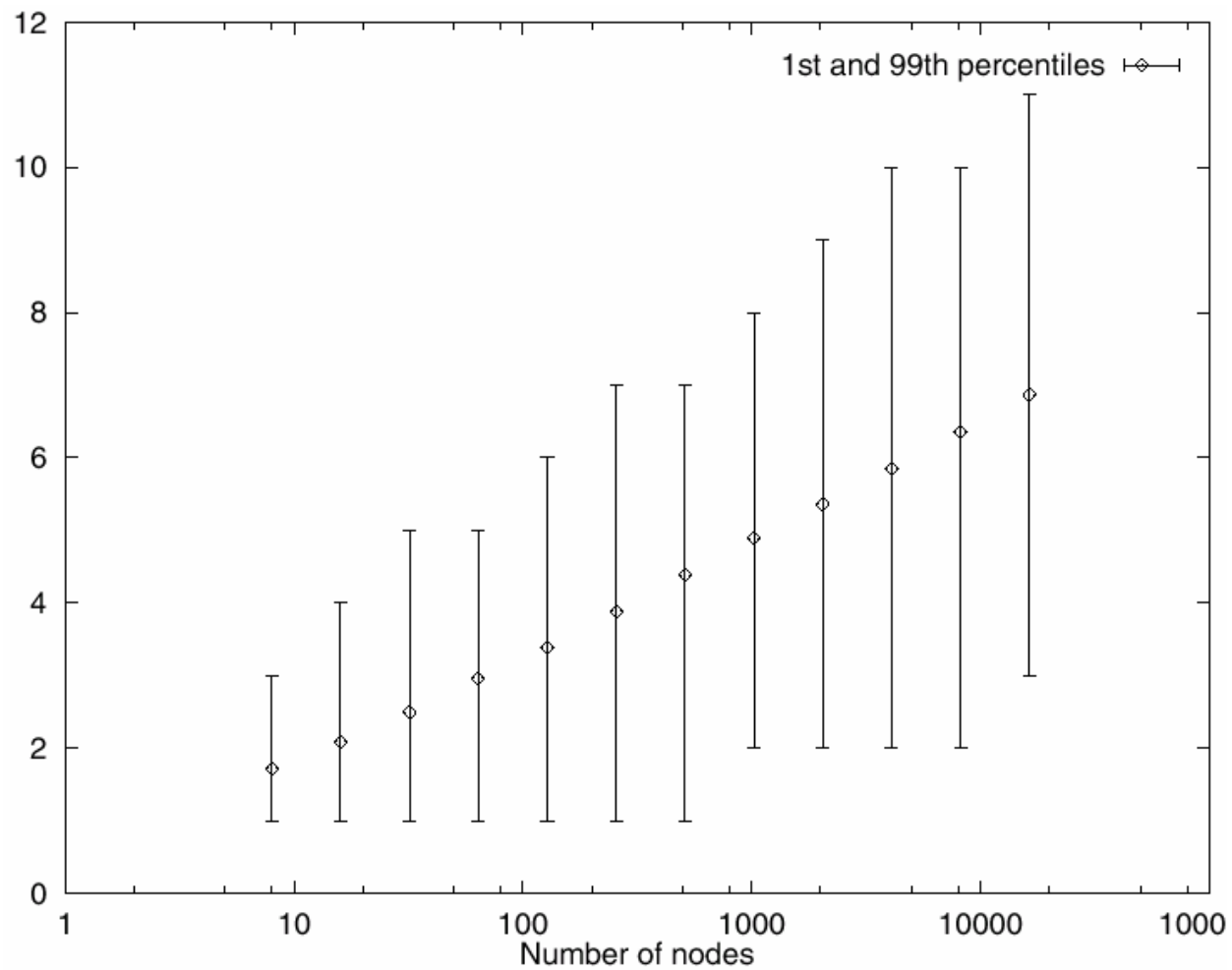
Iterative versus Recursive Lookups

- With iterative lookups, each node responsible for contacting intermediate hosts for successor information
- With recursive, intermediate nodes are responsible for obtaining answer and passing down the chain
- With analogy to DNS lookups
- What are the tradeoffs in iterative versus recursive lookup?
 - More hops in iterative lookup
 - But recursive puts more responsibility on remote hosts

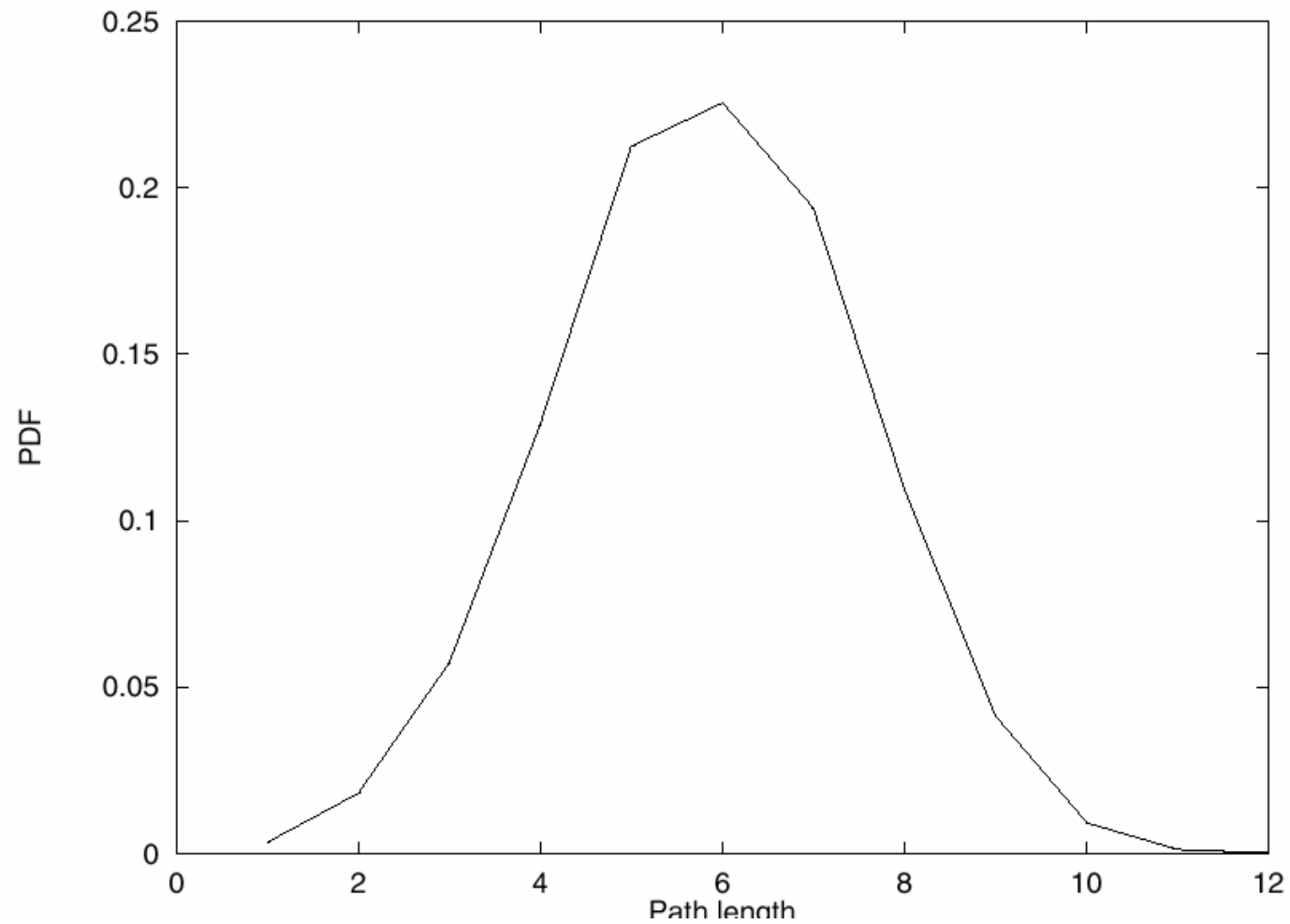
Effectiveness of Load Balancing



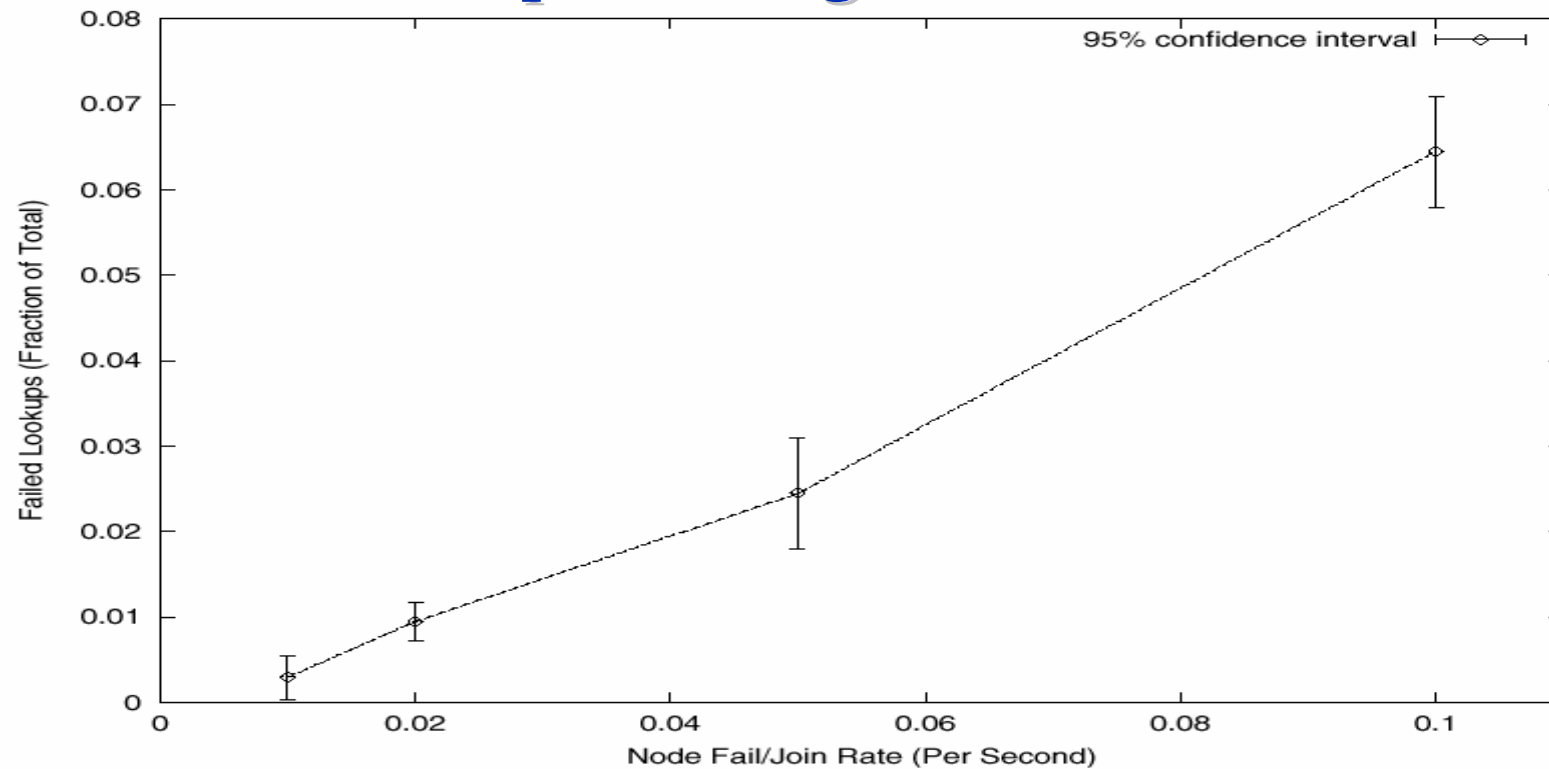
Path Length of Lookup



Distribution of Path Length (4096 nodes)



Lookups During Stabilization



- Stabilization function runs every 30 seconds
- 500 nodes total
- x axis varies from 1 mod every 100 sec to every 10 sec

Discussion

- Locality with respect to the underlying network?
 - From SD, first lookup goes to Australia, second to Europe, third to Asia
- Even $O(\lg n)$ steps too many for routing in large networks?
- Single popular key mapping to a single node?
- What about search?
- How does replication fit into the picture?