CSE 124
IP Security and
Peer-to-peer Networking

March 3, 2016, UCSD

Prof. George Porter

Outline

1. VPNs
2. IP network security
3. Peer-to-peer networks
4. TritonTransfer-p2p

Part 1:
VPNs and IP Security

(con’t)

IPSec

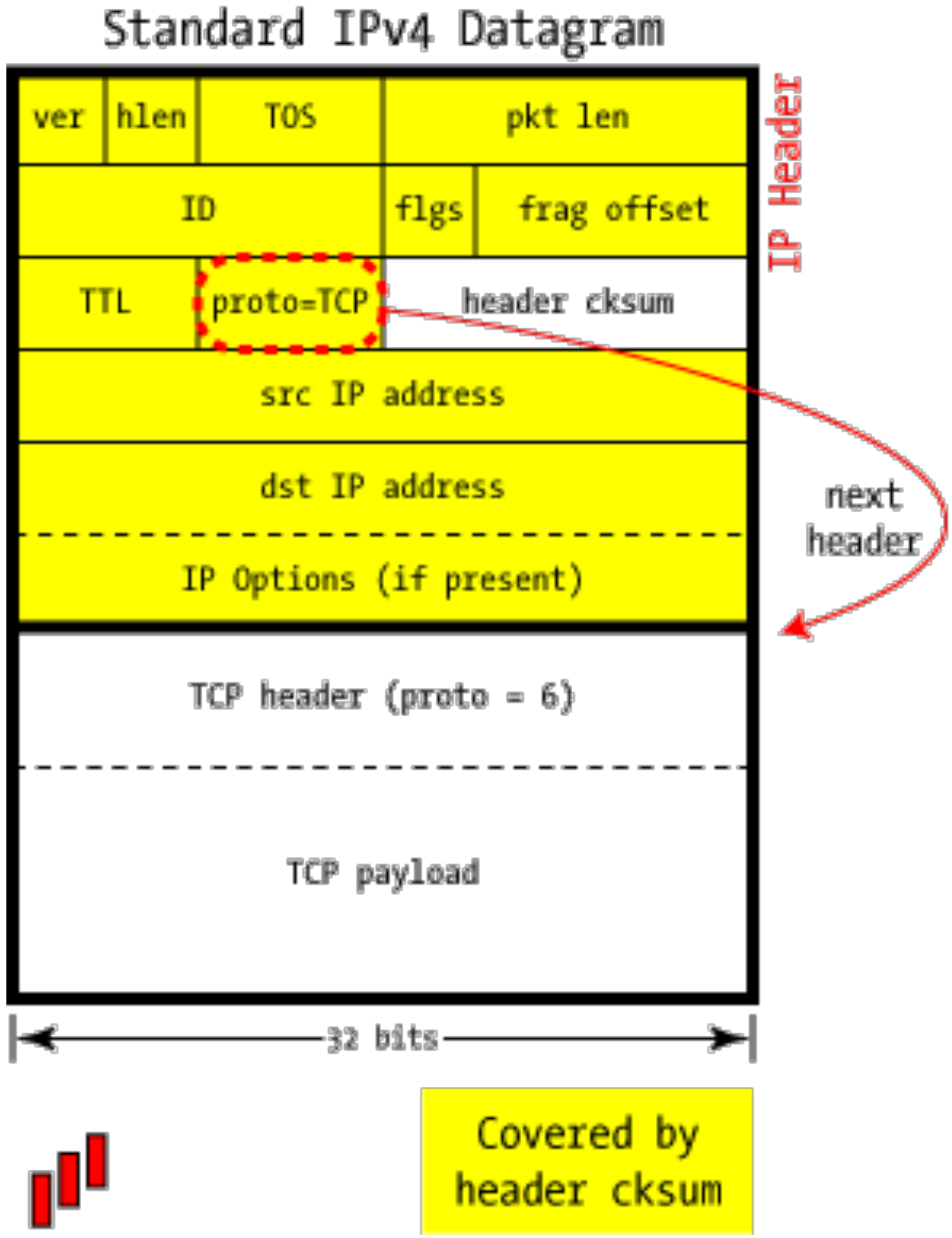
- Support for IPsec, as the architecture is called, is optional in IPv4 but mandatory in IPv6.
- IPsec is really a framework (as opposed to a single protocol or system) for providing all the security services discussed throughout this chapter.
- IPsec provides three degrees of freedom.
  - First, it is highly modular, allowing users (or more likely, system administrators) to select from a variety of cryptographic algorithms and specialized security protocols.
  - Second, IPsec allows users to select from a large menu of security properties, including access control, integrity, authentication, originality, and confidentiality.
  - Third, IPsec can be used to protect “narrow” streams (e.g., packets belonging to a particular TCP connection being sent between a pair of hosts) or “wide” streams (e.g., all packets flowing between a pair of routers).

Transport vs. tunnel mode

- Transport:
  - Host-to-host secure connection
  - Encrypted, authenticated, or both
- Tunnel
  - Host-to-network or network-to-network
  - Entire IP packet tunneled in secure IPSec "envelope" to recovered at destination

Security in IPSec

- AH: Authentication header
  - Access control, message integrity, authentication, and antireplay protection
- ESP: Encapsulating Security Payload
  - Like AH, but with encryption too
- SA: Security association
  - Selection of algorithms, crypto, hashes, etc
- SPI: Security Parameters Index (SPI)
  - Per-connection index into SA database
- ISAKMP: Internet Security Association and Key Management Protocol

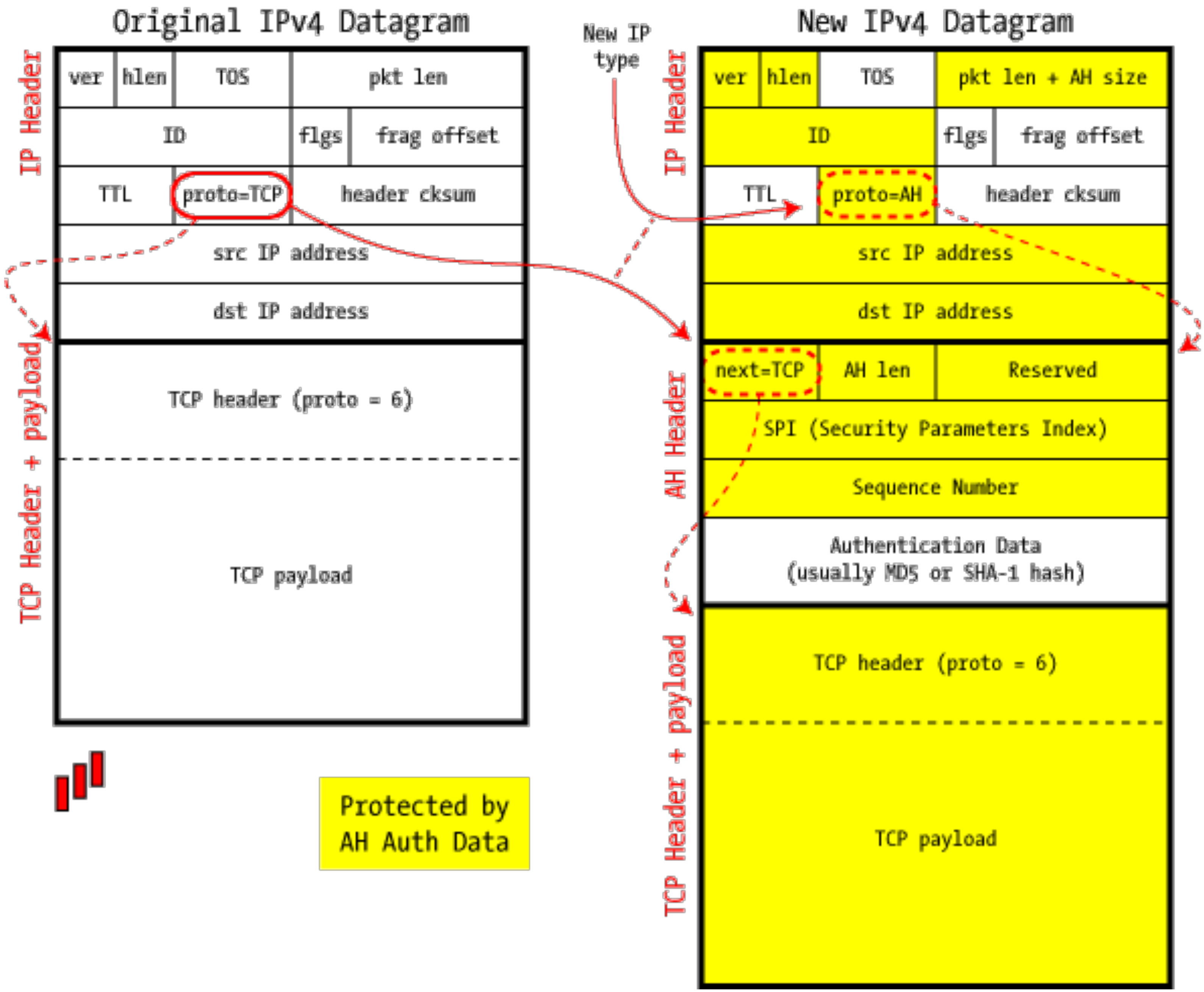


http://www.unixwiz.net/techtips/iguide-ipsec.html

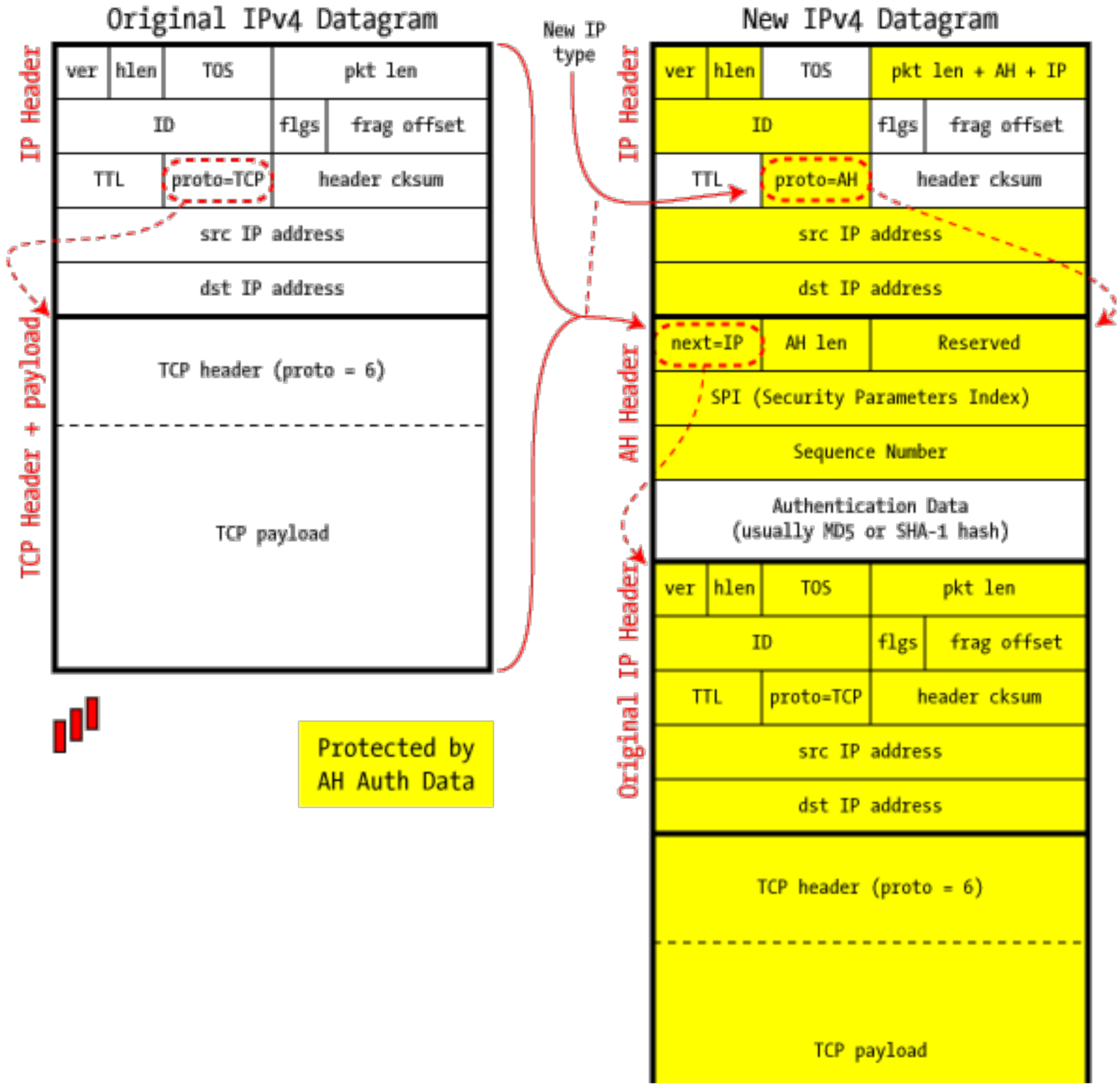
IP “next” protocols

Protocol code	Protocol Description
1	ICMP — Internet Control Message Protocol
2	IGMP — Internet Group Management Protocol
4	IP within IP (a kind of encapsulation)
6	TCP — Transmission Control Protocol
17	UDP — User Datagram Protocol
41	IPv6 — next-generation TCP/IP
47	GRE — Generic Router Encapsulation (used by PPTP)
50	IPsec: ESP — Encapsulating Security Payload
51	IPsec: AH — Authentication Header

IPSec in AH Transport Mode



IPSec in AH Tunnel Mode



Part 2:
Peer-to-peer networking

Overview and
unstructured p2p networks

Peer to peer (P2P) networks

- Applications of P2P
  - Storage, computation, network characterization
- Why are P2P systems gaining so much popularity?
- 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.

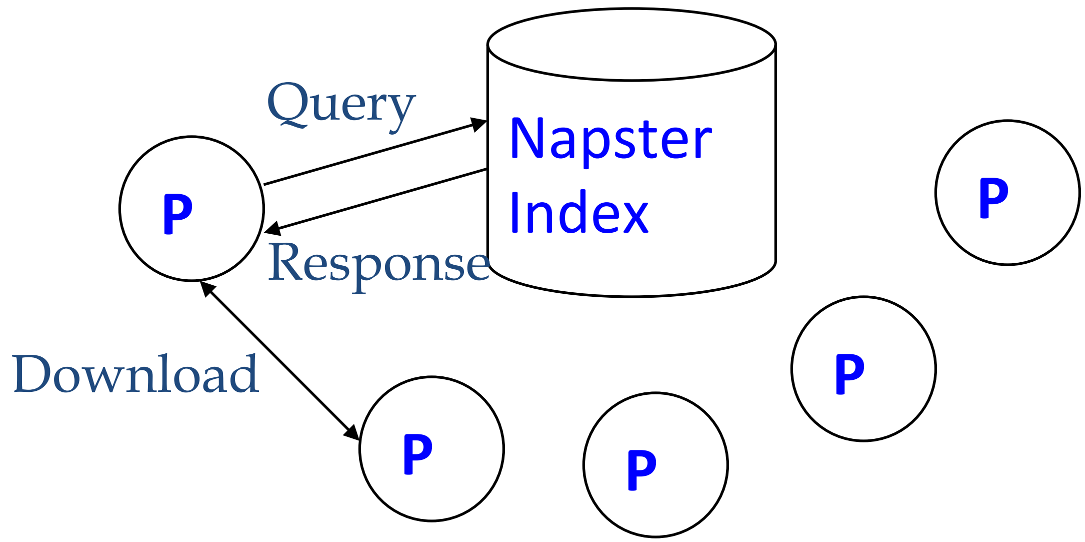
Peer-to-Peer Benefits

- 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

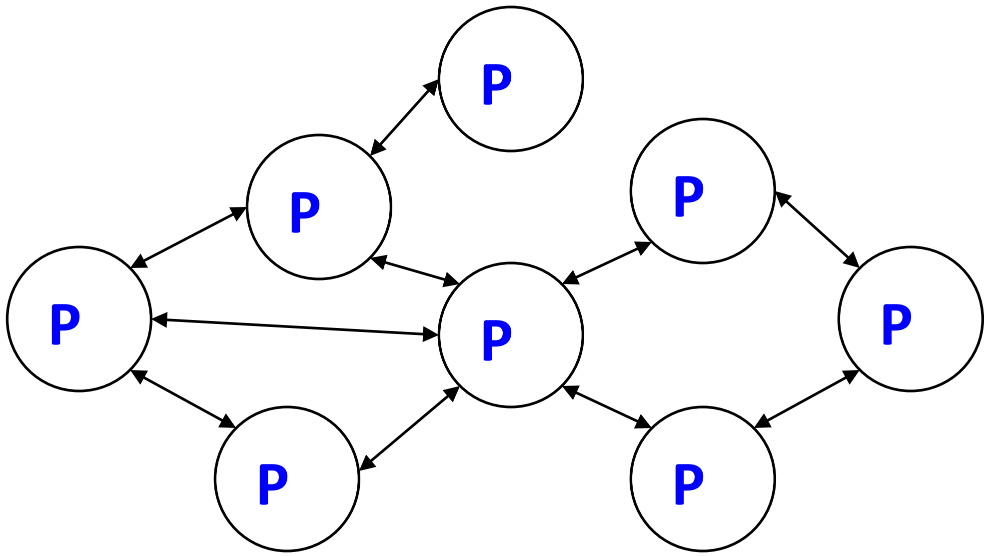
- Eternity Store
  - Research project at Berkeley: Oceanstore
- Farsite (Microsoft Research project)
  - xFS for client desktops
- Computation server
  - Seti@Home?
- Distributed index/distributed storage
  - Napster/Gnutella/Kazaa
  - Bit-torrent

Napster



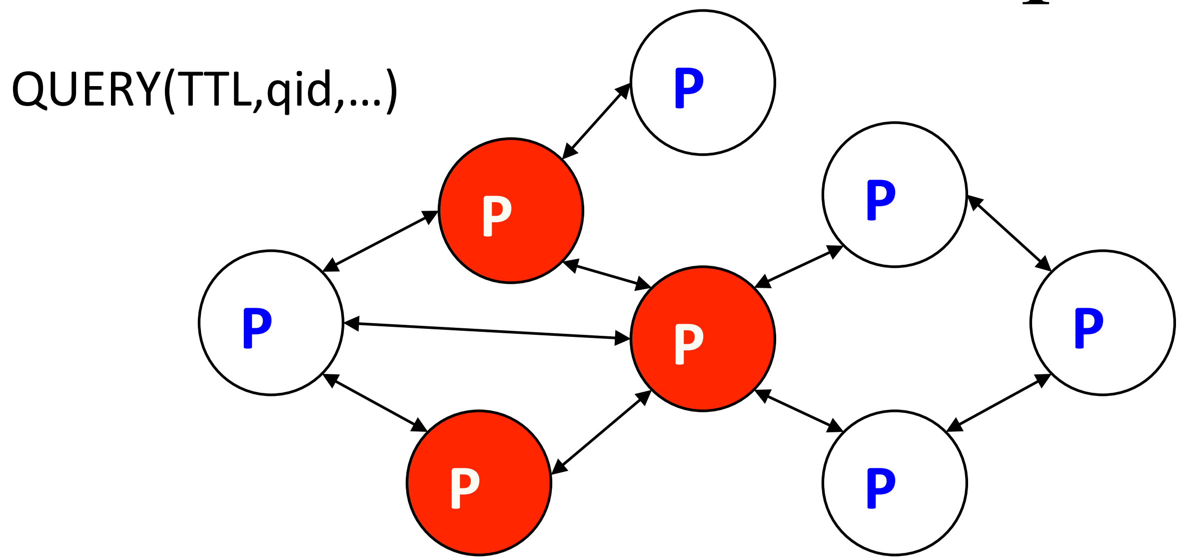
- Distributed storage, centralized index
- Which node to connect to?
  - Advertised connection speed, ping time from server

Gnutella



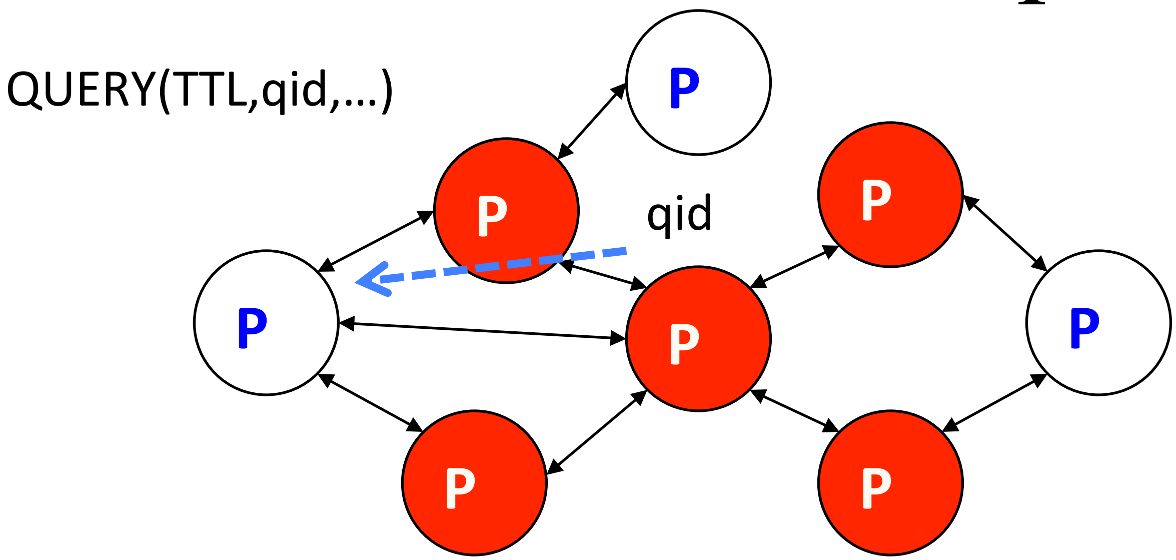
- Fully connected mesh
- Broadcast queries through the entire system
- Find just one member of the system and connect to it

Gnutella Requests



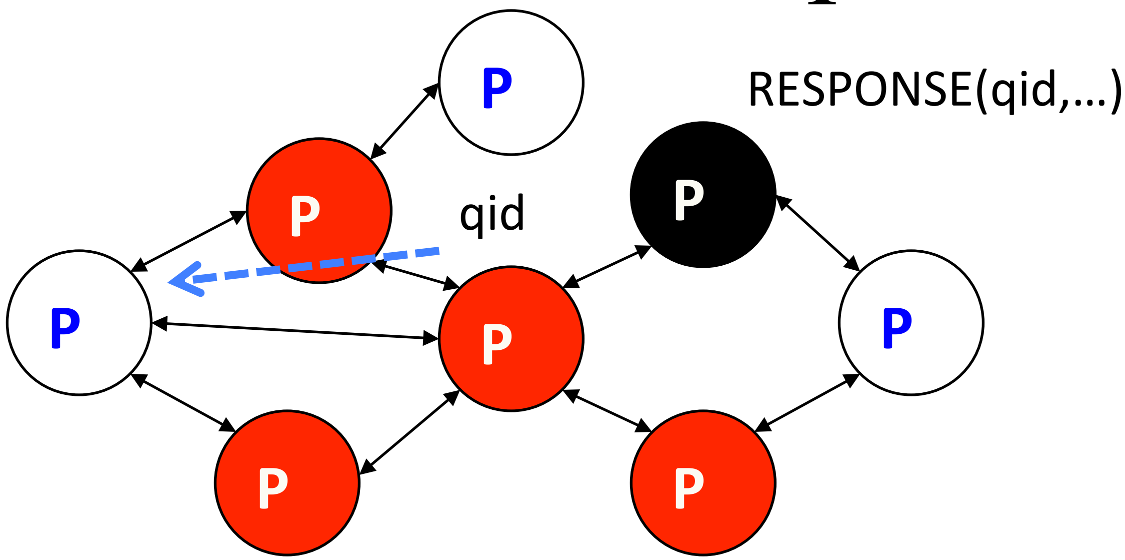
- Client sends QUERY message to neighbors
  - Limited by TTL field
- Each message has a query ID (qid)
  - To improve upon TTL
  - So that responses can be send back to the source

Gnutella Requests



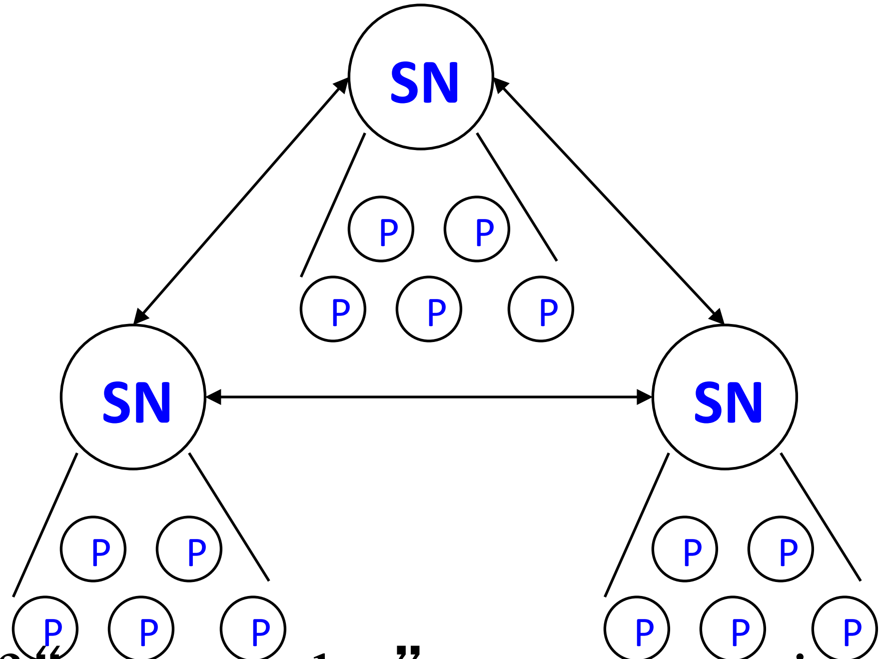
- Client sends QUERY message to neighbors
  - Limited by TTL field
- Each message has a query ID (qid)
  - To improve upon TTL
  - So that responses can be send back to the source

Gnutella Responses



- Client sends QUERY message to neighbors
  - Limited by TTL field
- Each message has a query ID (qid)
  - To improve upon TTL
  - So that responses can be send back to the source

Kazaa



- 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?

Part 3:
Peer-to-peer networking

Structured p2p networks and Chord

Chord

- Goal is to build fully distributed indexing scheme
- 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
- Note: P&D book, ch. 9.4 uses “Pastry”, which is functionally equivalent to Chord for our purposes

Chord Properties

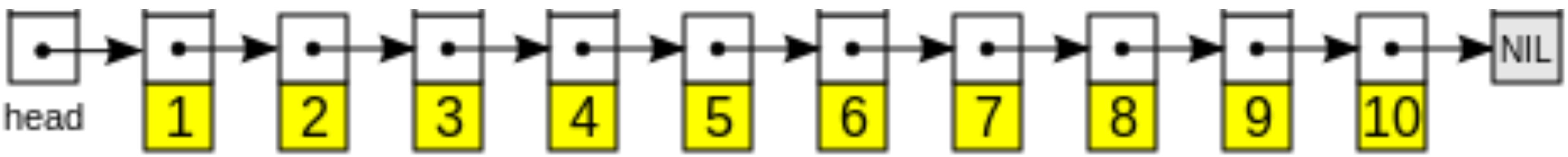
- Load balancing
- Decentralization
- Scalability
- Availability
- Flexible Naming
- Key idea:
  - Hash each object, use hash value to lookup that object
  - Just like a HashTable, but distributed to different nodes
  - Hash buckets → server IP addresses

How to choose the hash function?

- Hash(x): return x mod 101
  - What if more (or less) than 101 nodes?
- We could change the hash function on node entry/departure:
  - Hash(x): return x mod 102
  - But what happens to the data already in the system?

Chord intuition

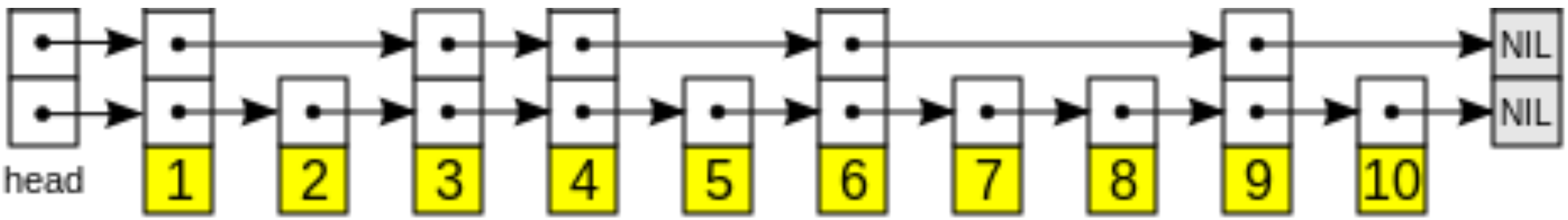
- Skip Lists (Pugh, 1989)
- Consider a linked list:



- Lookup time: O(n)

Chord intuition

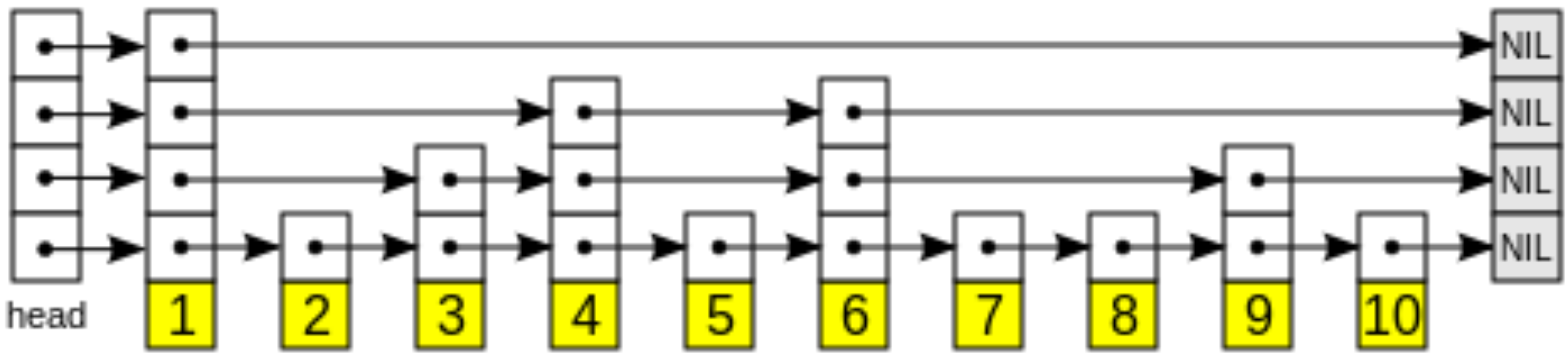
- Skip Lists (Pugh, 1989)
- Consider a linked list:



- Add 2nd row of pointers spaced further apart
  - Still O(n), but more efficient
  - Use 2nd row to get as close as possible without going over
  - Then last row to get to the desired element

Chord intuition

- Skip Lists (Pugh, 1989)
- Consider a linked list:

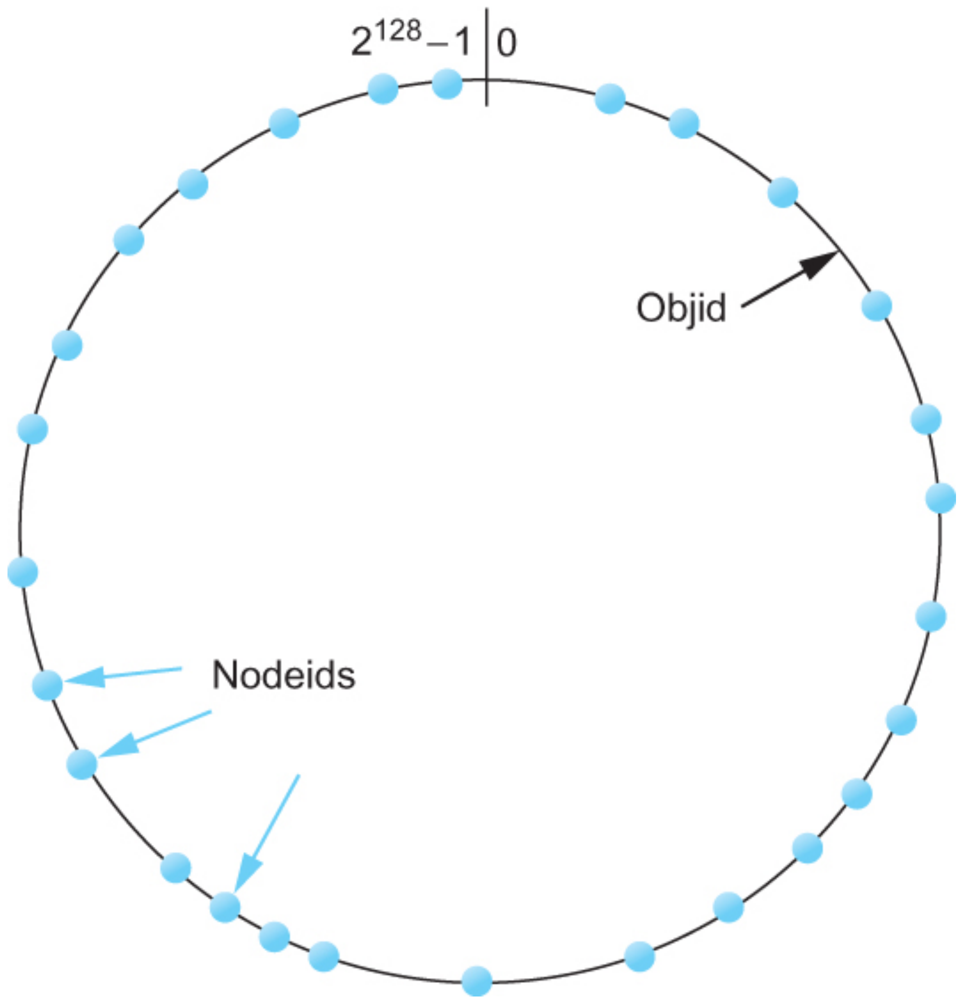


- Add log(N) rows
  - Get as close as possible on top row, then drop down a row, then drop down another row, until the bottom row
  - O(log N) lookup time

Chord: Consistent Hashing

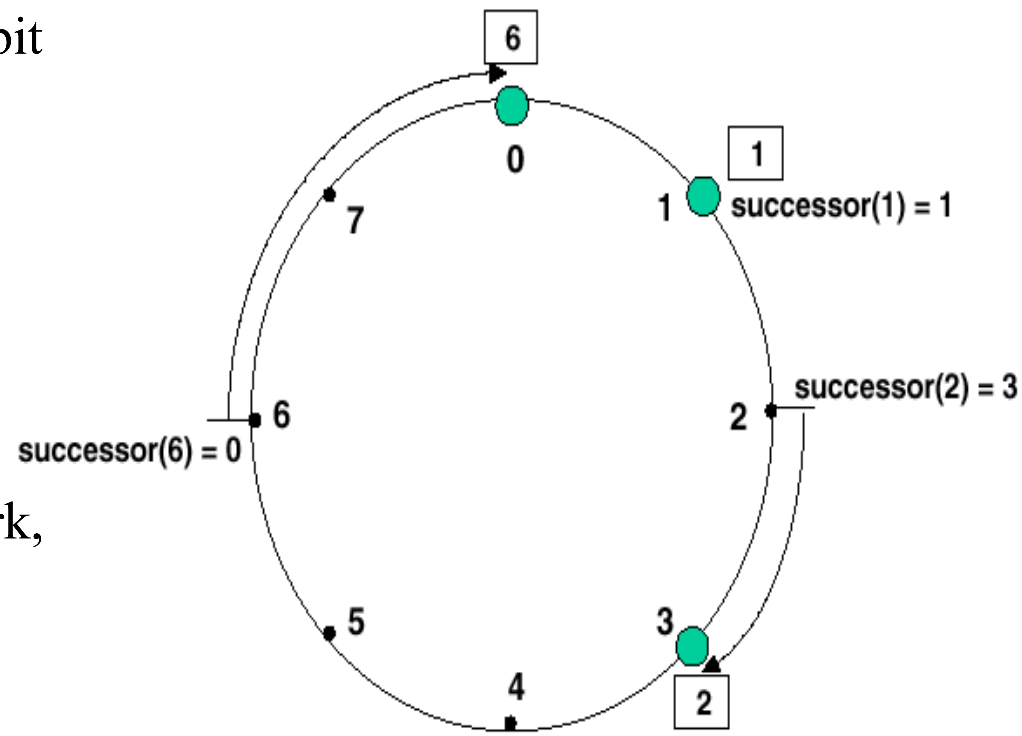
- Hash objects to very large space (e.g. 2^{128})
- Hash servers to same space (2^{128})
- Objects are stored on servers "near" them in the key space
- Given a set of *n* nodes, a consistent hash function will map keys (e.g., filenames) uniformly across the nodes
- Nice feature of consistent hashing for node addition:
  – Only *1/n* keys must be reassigned to new nodes
- Original proposals 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)

Consistent Hashing



Chord's Identifier Circle

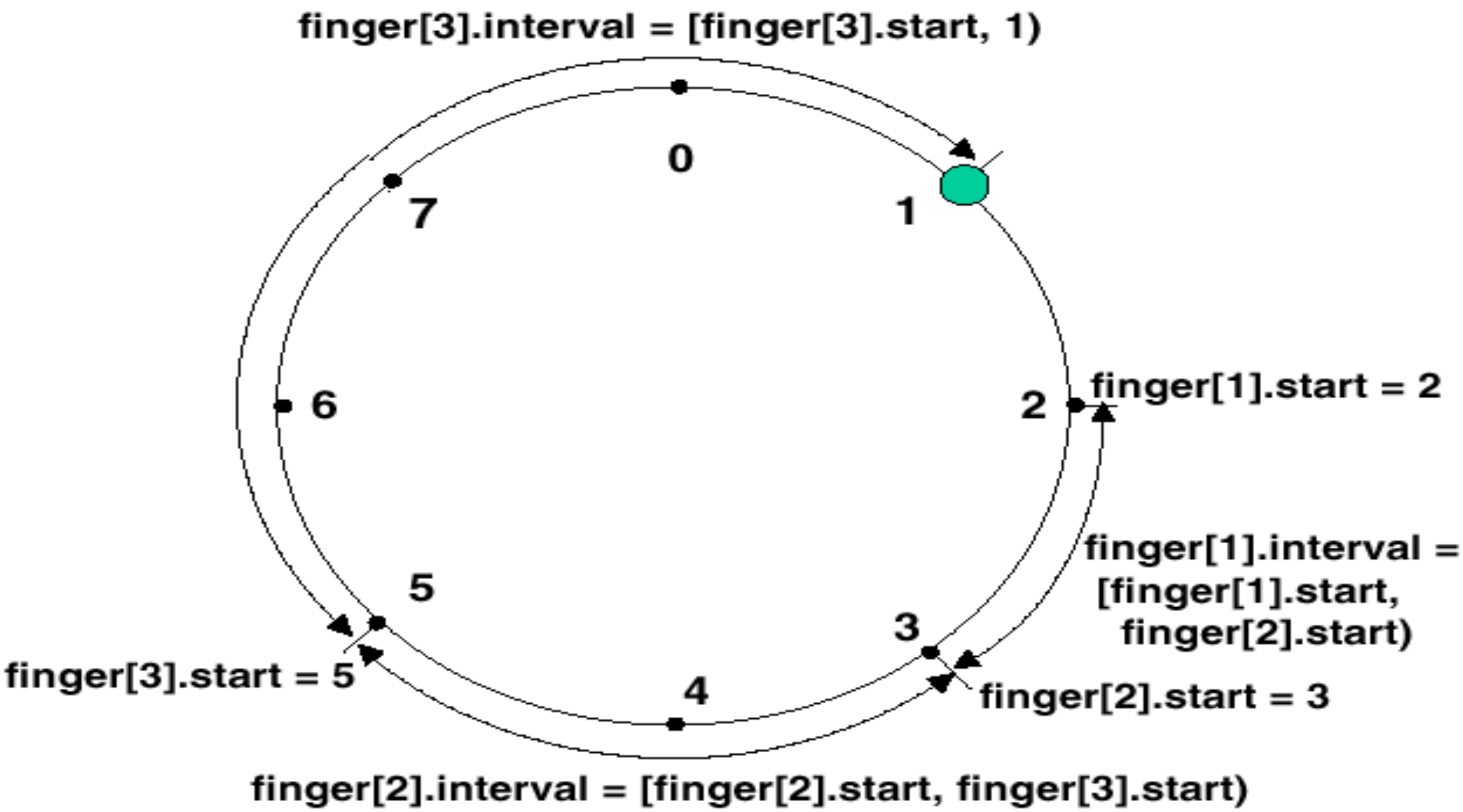
- Nodes and keys hashed to *m*-bit identifier
  - Assume keys > nodes
- Assign key *k* to first node whose identifier is equal to or larger than *k,* called *successor(k)*
- When node *n* joins the network, certain keys assigned to successor(n), now become mapped to n
  - When node *n* leaves the network,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



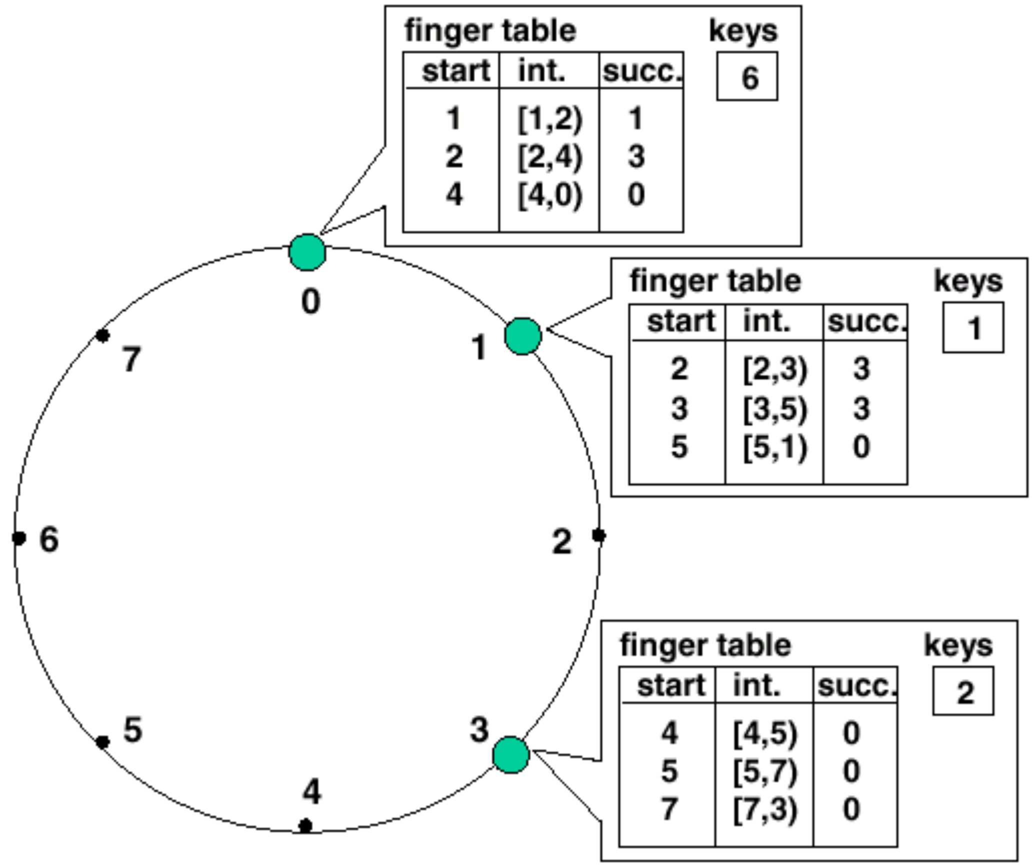
Scalable Key Location

- 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 successor(*n)*

Finger Table



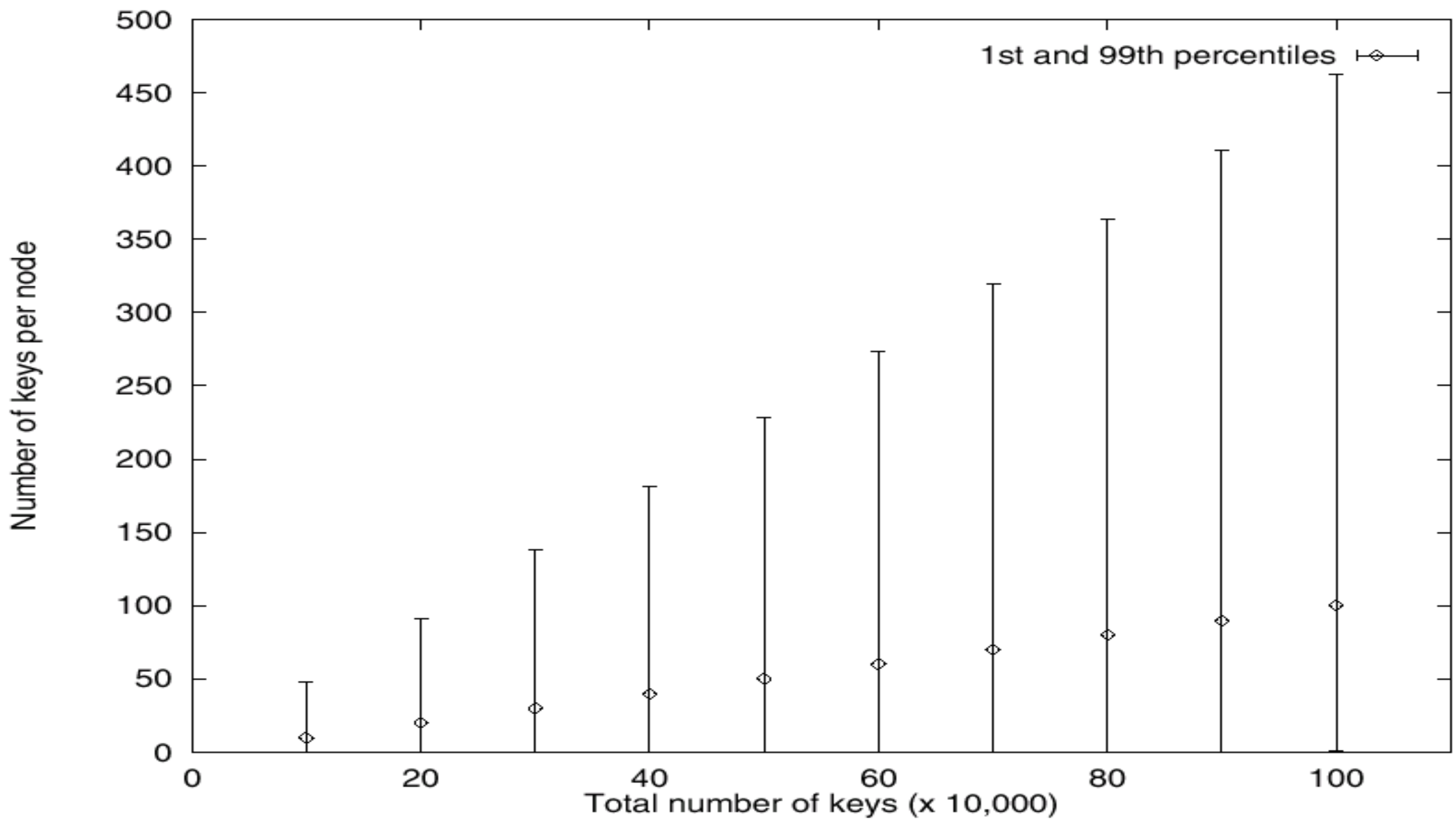
Finger Table



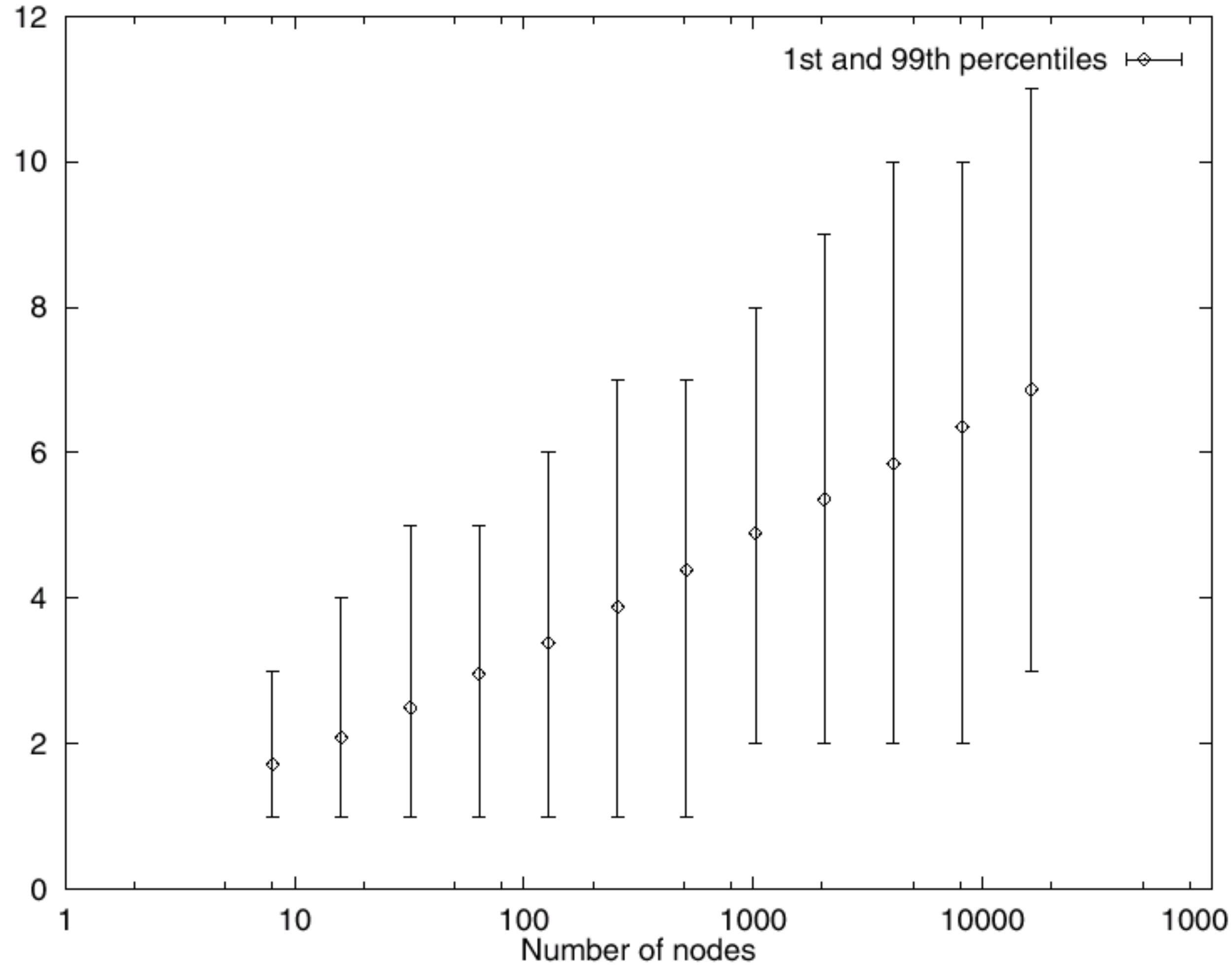
Iterative versus Recursive Lookups

- 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?

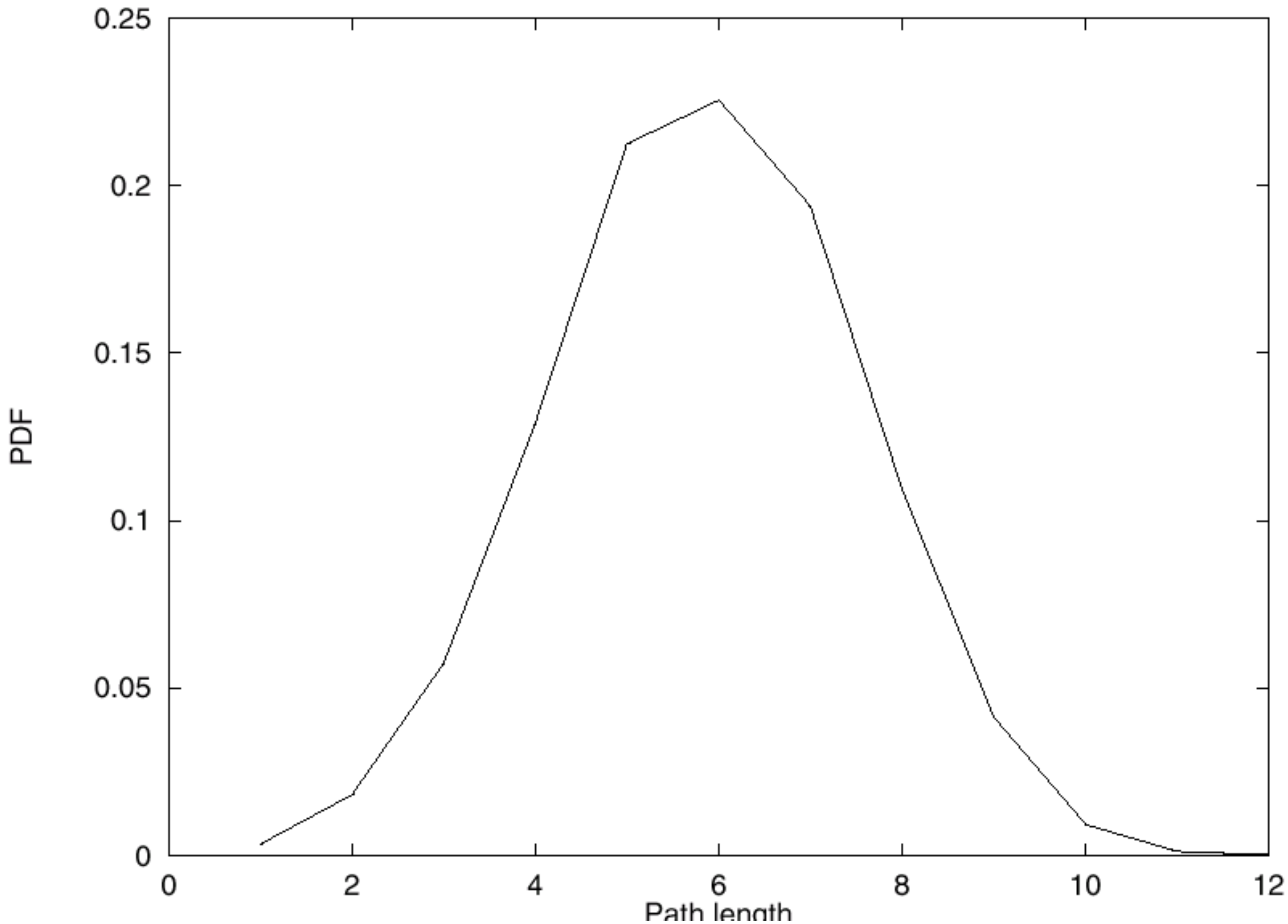
Effectiveness of Load Balancing



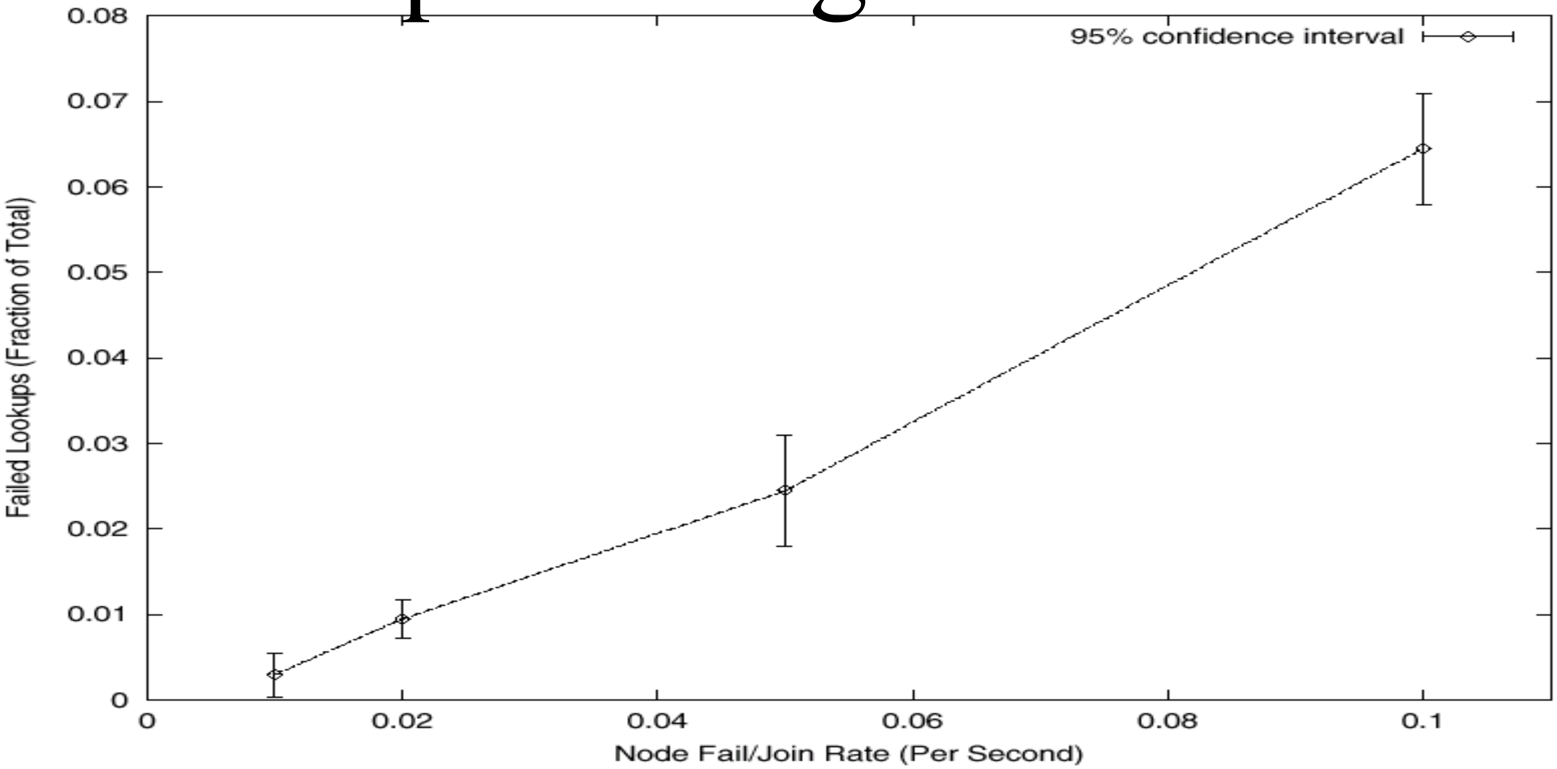
Path Length of Lookup



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?

Part 4:
TritonTransfer-p2p

Key ideas

- Separating the *metadata* server from the *block* server(s)
  - Modeled on Apache Hadoop/HDFS
- Surviving failures via replication
  - 2 replicas mean any server can be killed without loss of data

Implementation challenges

- How does the client find out about the block servers?

- Should the client upload blocks first, then create the file? Or create the file then upload the blocks?
  - Hint: depends on who chooses the location of the blocks—the client of the metadata server?