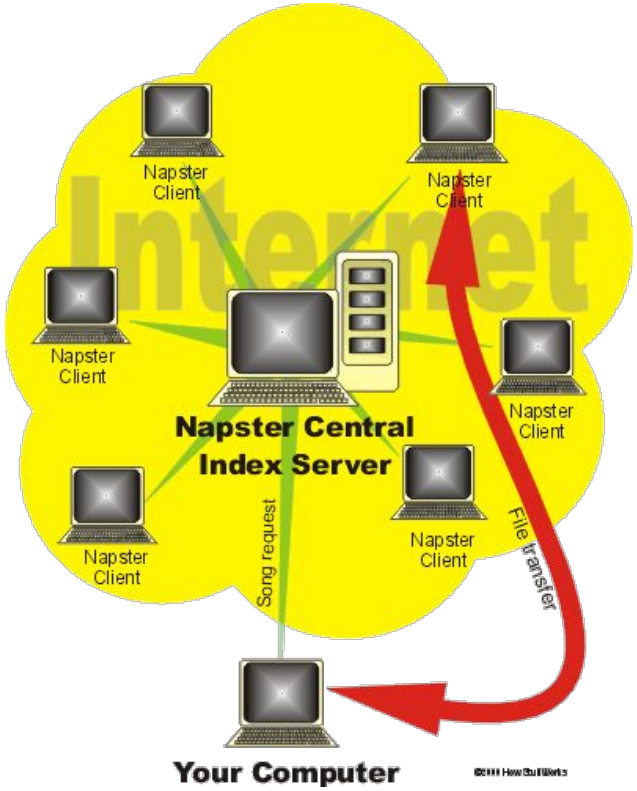
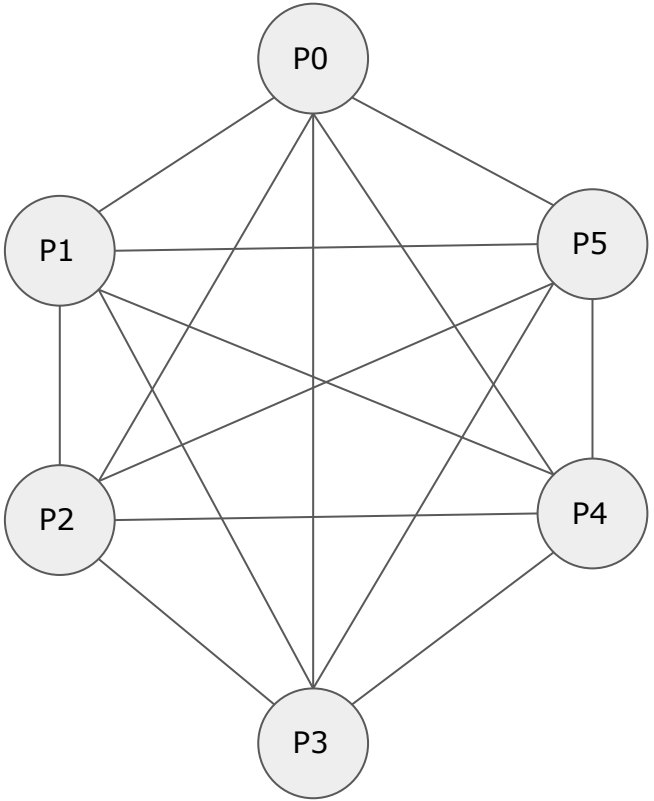
CSE 550 Autumn 2021
Large Scale Storage Systems

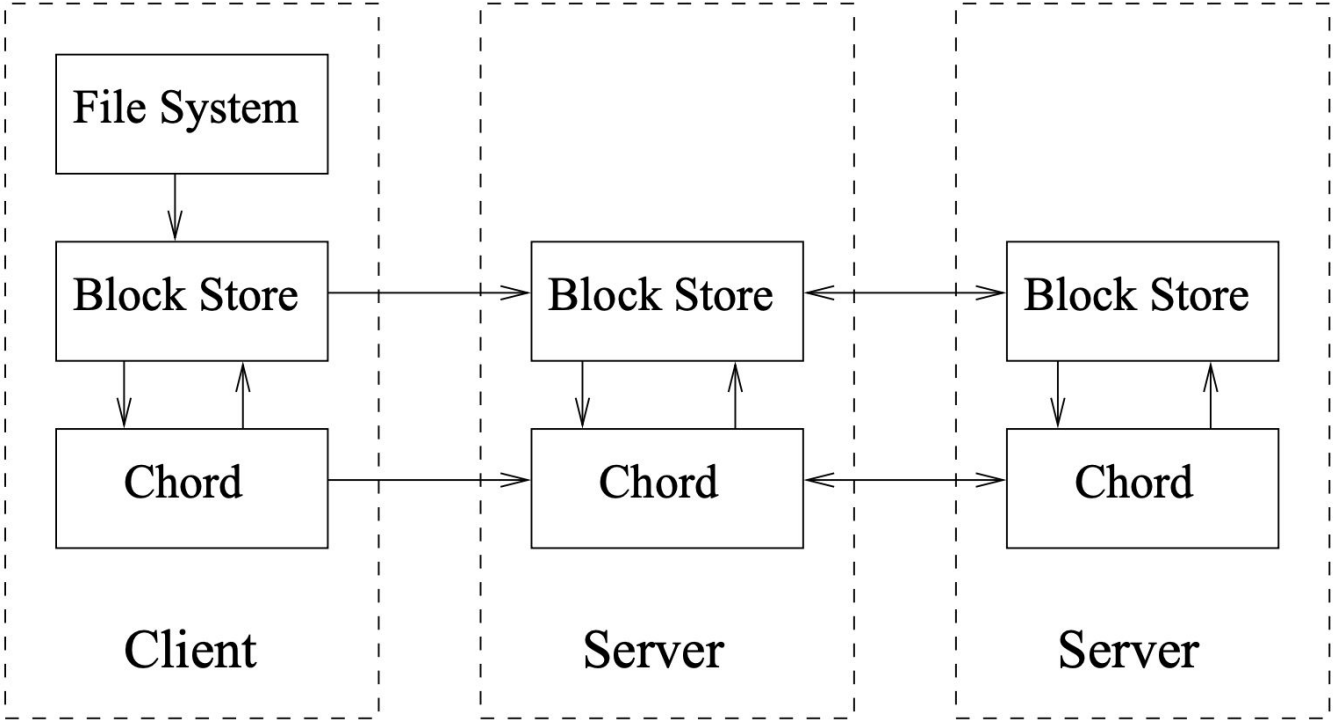
Presentor: Zihao Ye, Weixin Deng

Centralized vs Decentralized





Distributed Storage System



Distributed Hash Table

P2P(Peer-to-Peer) Lookup service

Interface

- put(key, value)
- get(key)

Example:

- BitTorrent
- CDN
- DNS

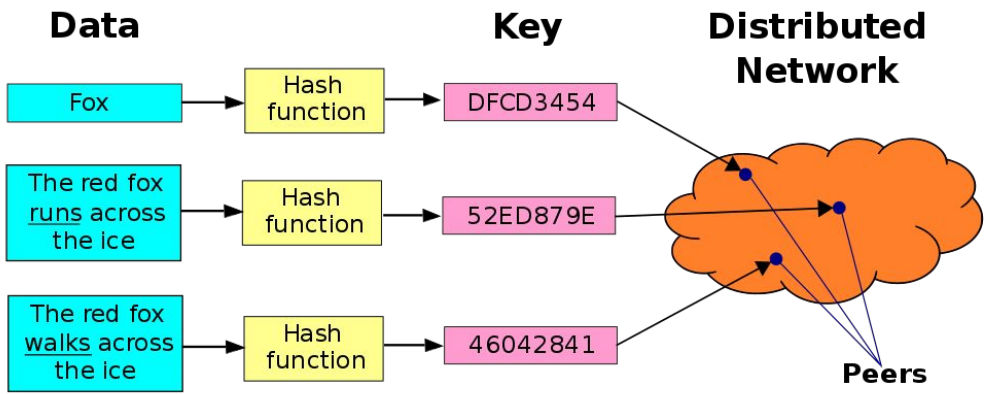
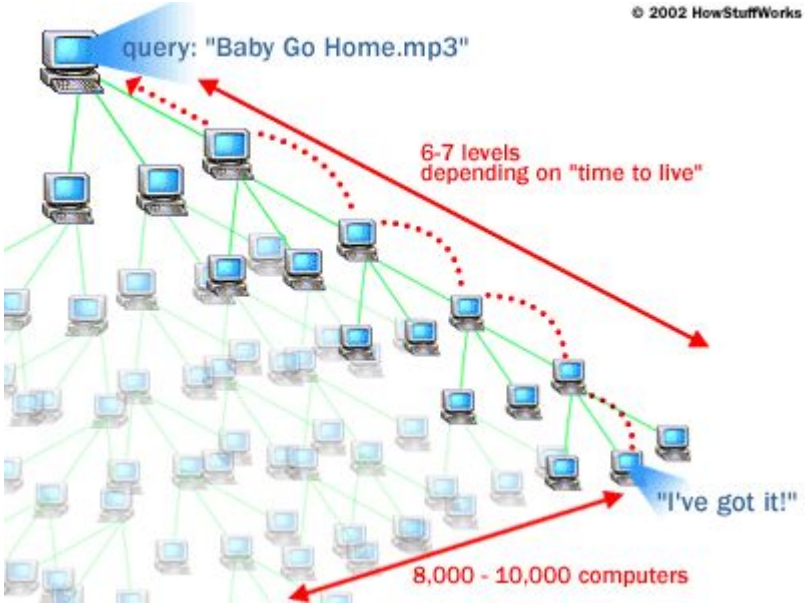


Image source: wiki

Solution?

- Gnutella (Flooding)
  - Broadcasting request to neighbors.

Incurs large amount of Internet traffic.



Chord: A Scalable P2P protocol

- Efficient & scalable
- Fault tolerant
- Easy to understand and extend

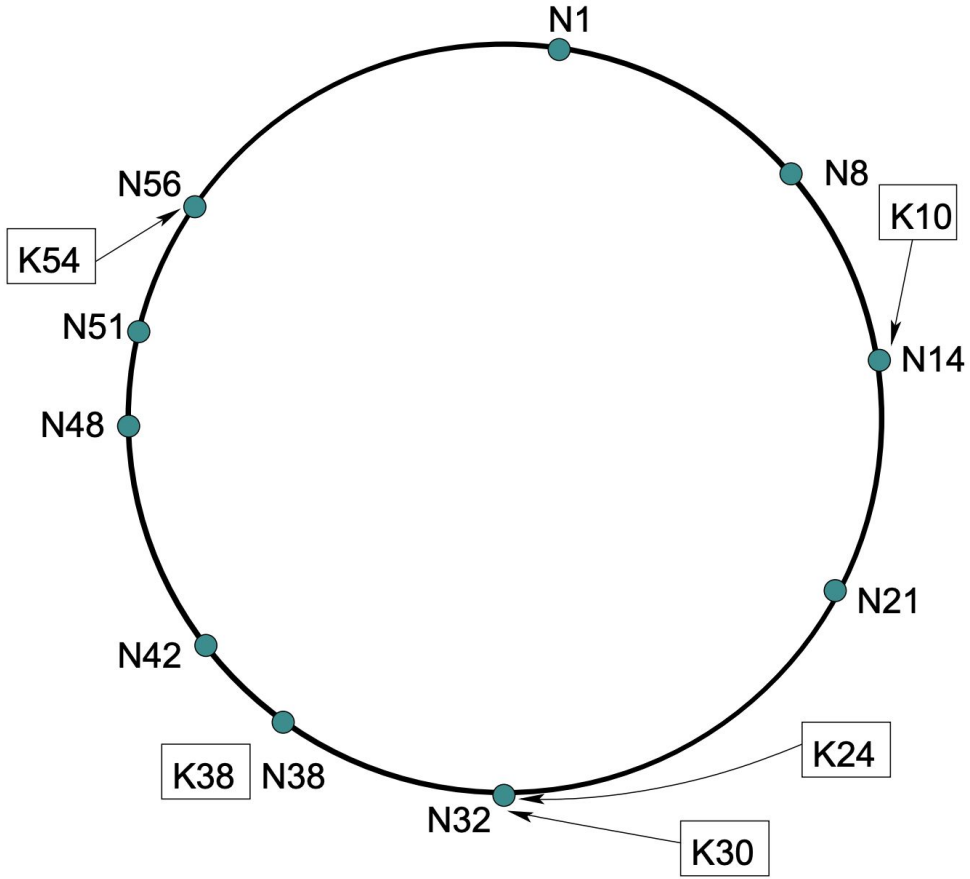
Many concurrent work on this topic...

> In 2001, four systems—CAN,[6] Chord,[7] Pastry, and Tapestry—ignited DHTs as a popular research topic. A project called the Infrastructure for Resilient Internet Systems (Iris) was funded by a $12 million grant from the United States National Science Foundation in 2002.[8] Researchers included Sylvia Ratnasamy, Ion Stoica, Hari Balakrishnan and Scott Shenker.

Consistent Hashing

Identifier circle

- Modulo 2^m (m-bit identifier)
- *id(key) = hash(key)*
- *id(node) = hash(ip(node))*
- *successor(k)* (first clockwise node)



Load Balance of Consistent Hashing

- Variations are high
- Introduce virtual nodes for a more uniform hashing result

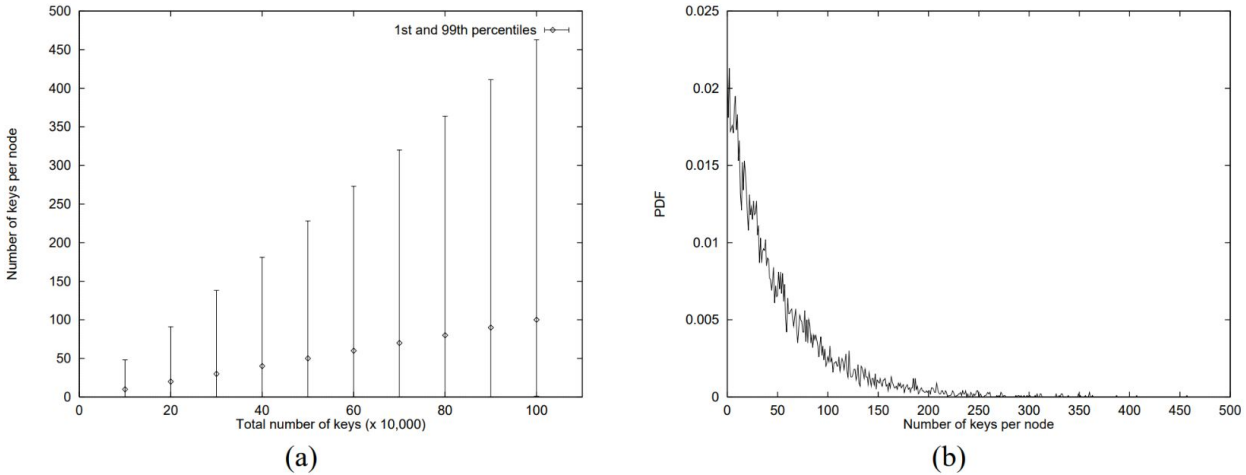


Fig. 8. (a) The mean and 1st and 99th percentiles of the number of keys stored per node in a 10^4 node network. (b) The probability density function (PDF) of the number of keys per node. The total number of keys is 5×10^5.

Finger Table

- $n.finger[i] = successor(n+2^{i-1})$

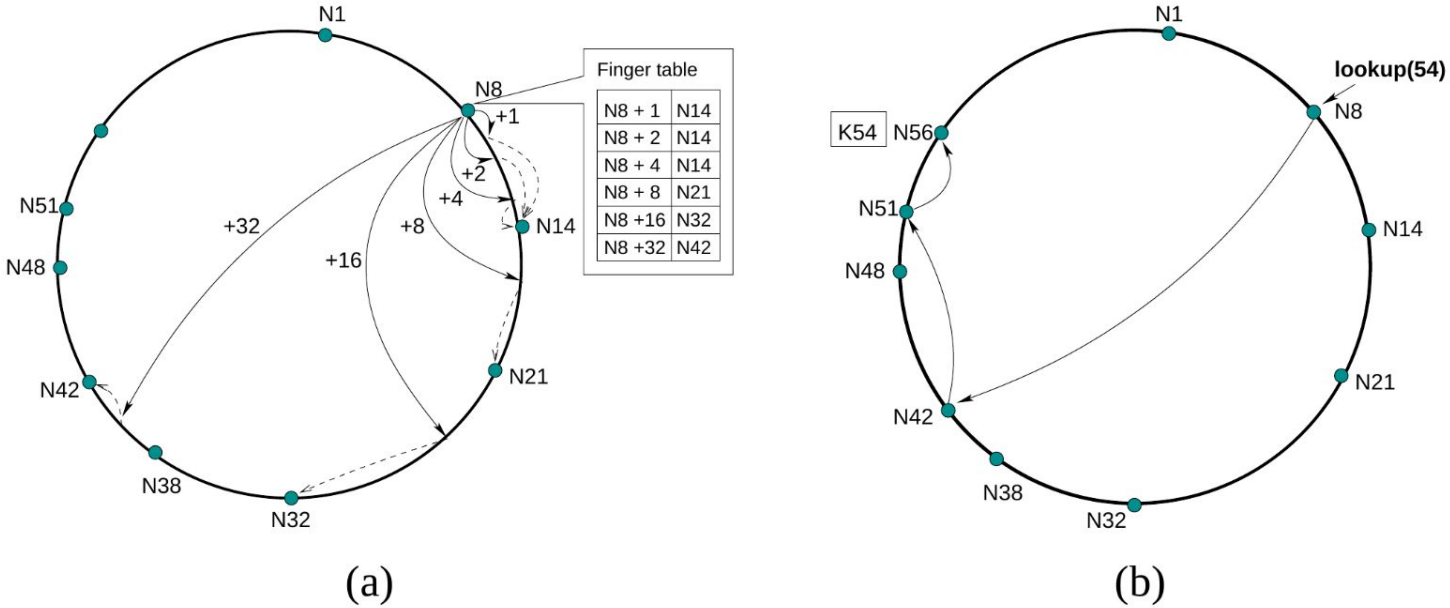


Fig. 4. (a) The finger table entries for node 8. (b) The path a query for key 54 starting at node 8, using the algorithm in Figure 5.

Routing w/ Finger Table

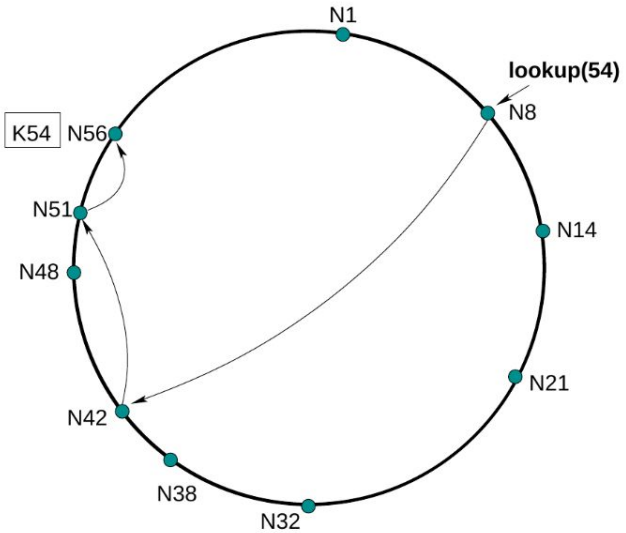
```
// ask node n to find the successor of id
n.find_successor(id)
  if (id ∈ (n, successor])
    return successor;
  else
    n' = closest_preceding_node(id);
    return n'.find_successor(id);

// search the local table for the highest predecessor of id
n.closest_preceding_node(id)
  for i = m downto 1
    if (finger[i] ∈ (n, id))
      return finger[i];
  return n;
```

Fig. 5. Scalable key lookup using the finger table.

(a)

Finger table

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



(b)

Fig. 4. (a) The finger table entries for node 8. (b) The path a query for key 54 starting at node 8, using the algorithm in Figure 5.

Path Length of Chord Routing Protocol

- Path length is linear to log(*N*) (about ½ log(*N*))

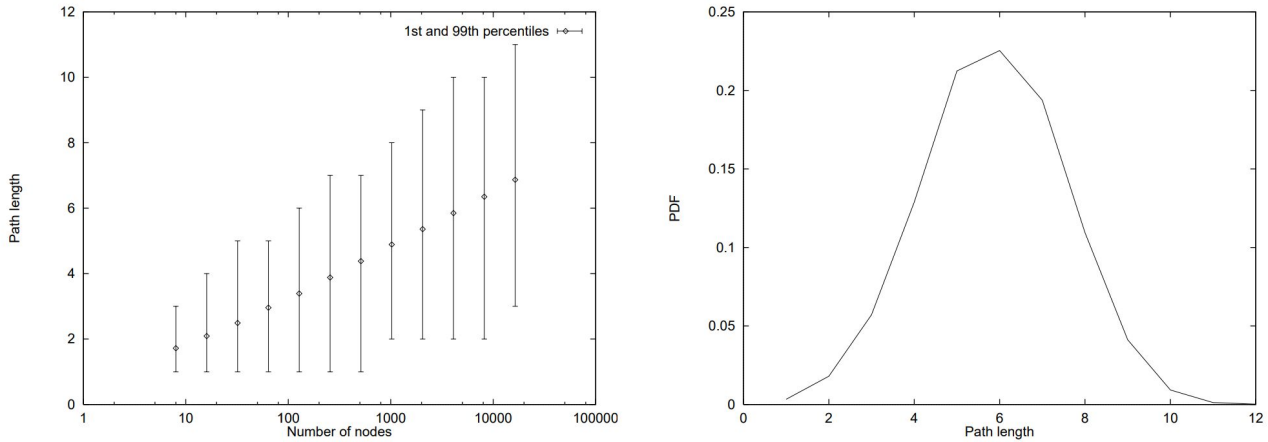


Fig. 10. (a) The path length as a function of network size. (b) The PDF of the path length in the case of a 2^{12} node network.

Similar Idea from Power of 2

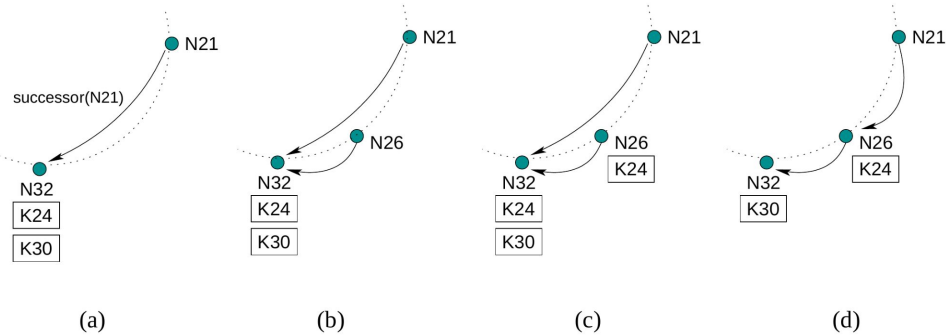
- Cut search space half each time
- Examples
  - Skip list
  - Segment tree
  - Binary search
  - Lowest common ancestor

Discussion #1

1. Nodes can distribute all around the world, and communication between different pairs of nodes has different latency. The path finding algorithm mentioned before might not be optimal, can you improve the efficiency of the neighbor selection/forwarding selection in Chord?

https://tinyurl.com/cse550chord

Node Joins

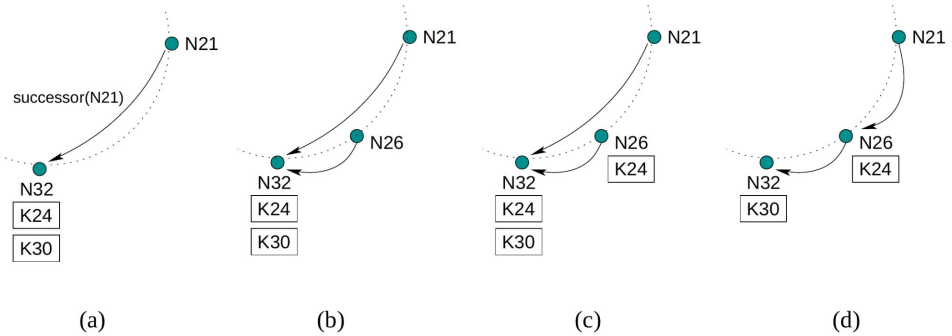


- How to find correct *successor(k)*
  - Successor pointers
  - Finger tables

```
// create a new Chord ring.
n.create()
    predecessor = nil;
    successor = n;

// join a Chord ring containing node n'.
n.join(n')
    predecessor = nil;
    successor = n'.find_successor(n);
```

Node Joins

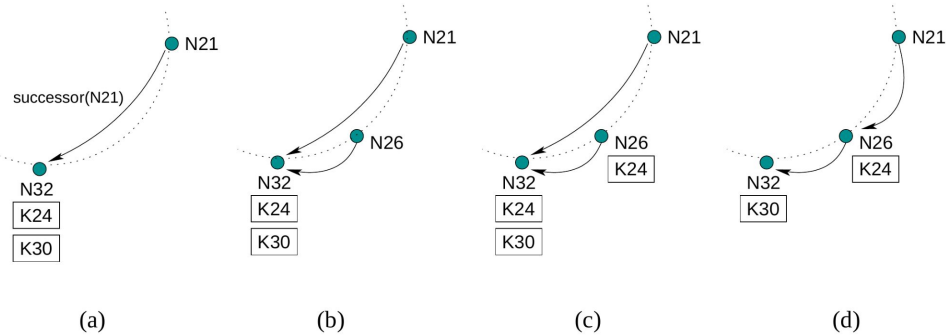
- How to find correct *successor(k)*
  - Successor pointers
  - Finger tables




```
// create a new Chord ring.
n.create()
    predecessor = nil;
    successor = n;

// join a Chord ring containing node n'.
n.join(n')
    predecessor = nil;
    successor = n'.find_successor(n);
```

```
// called periodically. verifies n's immediate
// successor, and tells the successor about n.
n.stabilize()
    x = successor.predecessor;
    if (x ∈ (n, successor))
        successor = x;
    successor.notify(n);

// n' thinks it might be our predecessor.
n.notify(n')
    if (predecessor is nil or n' ∈ (predecessor, n))
        predecessor = n';

// called periodically. checks whether predecessor has failed.
n.check_predecessor()
    if (predecessor has failed)
        predecessor = nil;
```

Node Joins



- How to find correct *successor(k)*
  - Successor pointers
  - Finger tables

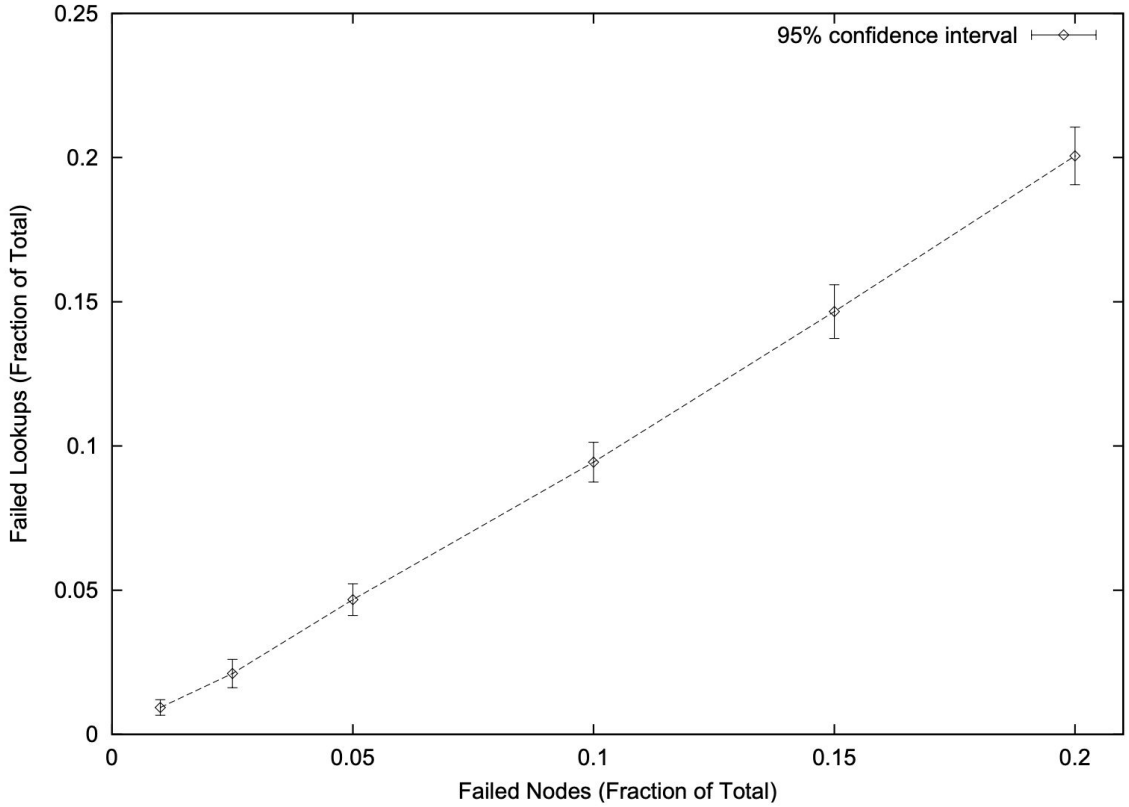
```
// called periodically. refreshes finger table entries.
// next stores the index of the next finger to fix.
n.fix_fingers()
    next = next + 1;
    if (next > m)
        next = ⌊log(successor − n)⌋ + 1;  // first non-trivial finger.
    finger[next] = find_successor(n + 2^(next−1));
```

Concurrent Operations and Failures

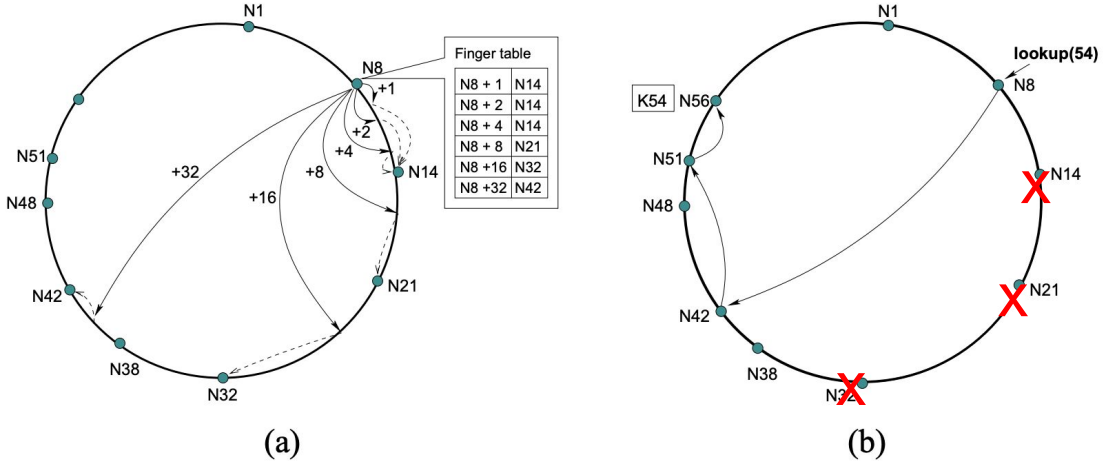
What if there are concurrent joins or leaves?

Stabilization doesn’t always work.

Fault Tolerance



Fault Tolerance



How to promote fault tolerance?

- Maintain a *successor list* of size *r*
- The probability that all successors fail is small

Replication and Authentication

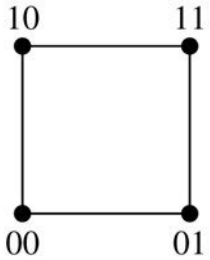
Application using Chord handle the replication and authentication themselves.

- Authentication: storing data x at key MD5(x)
- Replication
  - save x at *k* nodes succeeding the key.
  - storing x under two distinct Chord keys derived from the data’s application-level identifier, e.g. hash(x, 1), hash(x, 2)

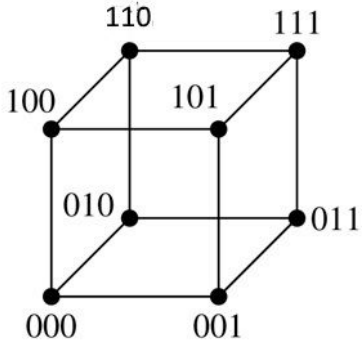
CAN: another DHT proposal

CAN uses a hypercube graph-like structure to route between nodes.

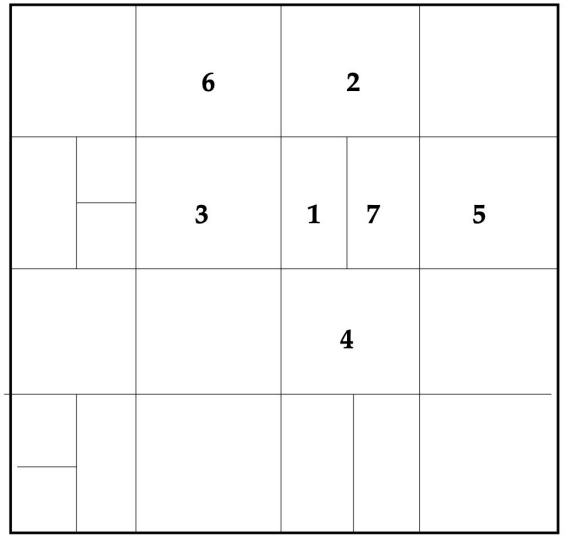
Q_1



Q_2



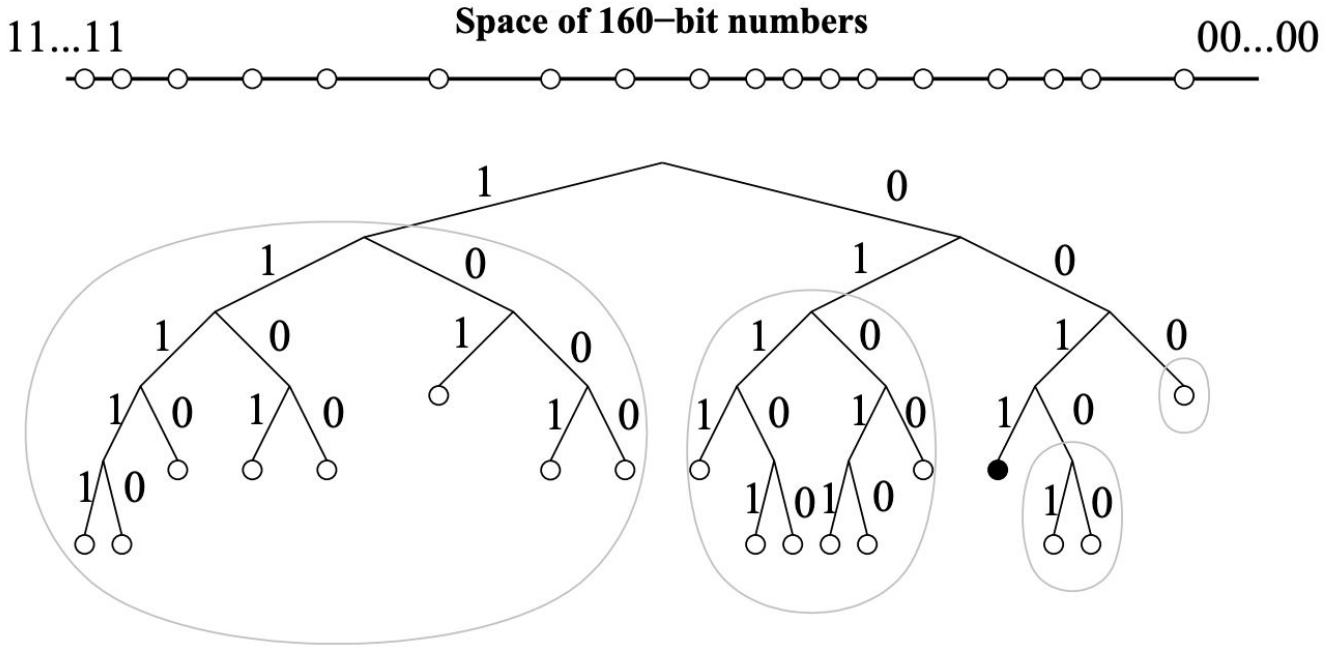
Q_3



1's coordinate neighbor set = {2,3,4,7}
7's coordinate neighbor set = {1,2,4,5}

Kademlia

Organize id’s as a tree-like structure.



Kademlia

XOR (bitwise exclusive or) metric:

$$d(x, y) = x \oplus y$$

Symmetric:

$$\forall x, y : d(x, y) = d(y, x)$$

Triangle Property:

$$d(x, y) + d(y, z) \geq d(x, z)$$

Security?

Sybil attack

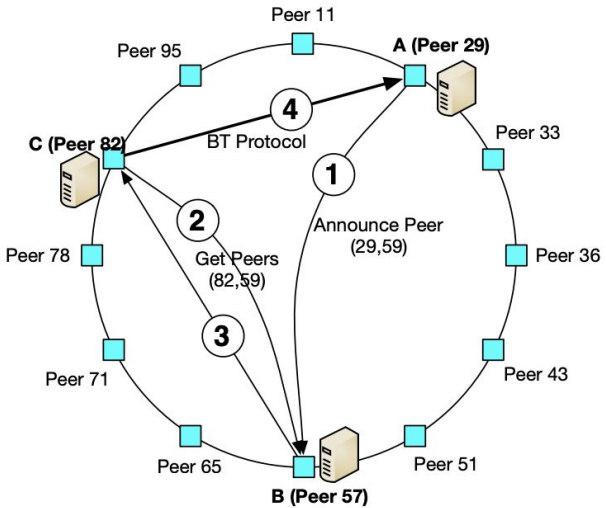
- Inject multiple fake identities into the system so to pollute users' routing table.
- Use these fake identities to perform further attacks.
- Horizontal/Vertical attack, and mixture of them.

Real-World Sybil Attacks in BitTorrent Mainline DHT

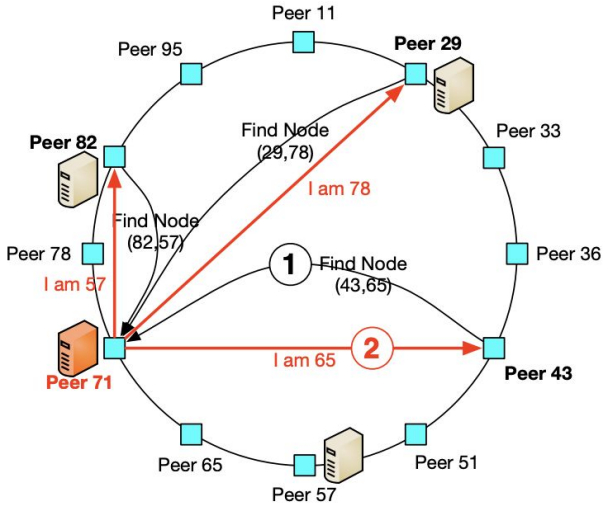
Security

Horizontal attack

- Pollute maximum number of routing table.



(a) Normal operations



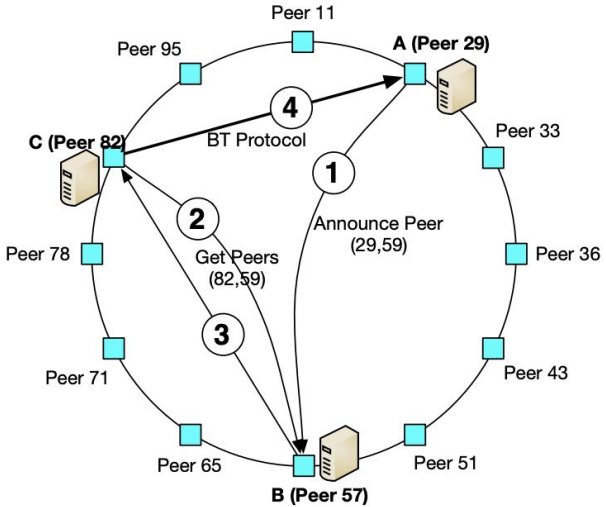
(b) Horizontal attack

Security

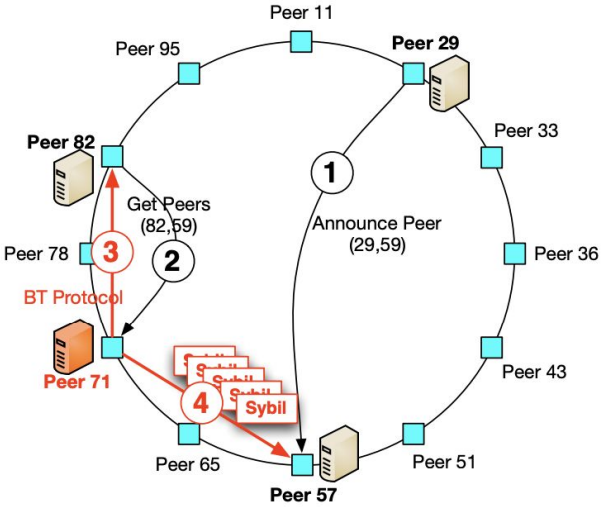
Vertical attack

Wait for target tID
for $i \leftarrow 1$ *to* k **do**
  Create sybil with $ID \leftarrow tID + i$
end

- Insert as many sybils as possible in one specific routing table.



(a) Normal operations



(c) Vertical attack

Potential threats

System wide

- Control most of the routing and introduce delay/lookup fail.

Content wide

- Pollute target content.

Privacy

- Attacker may know what content a user is requesting.

Security

Honeypot detector

1. Periodically find the k closest neighbor some random id’s in the system.
   a. To identify horizontal attack.
2. Periodically search for non-existent infohashes in the system.

Discussion #2

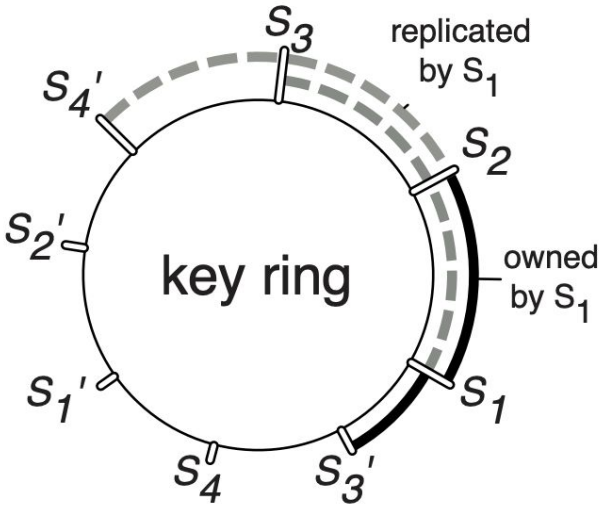
2. What issues arise if a large percentage of the network is in flux? How can we mitigate the mentioned issues?

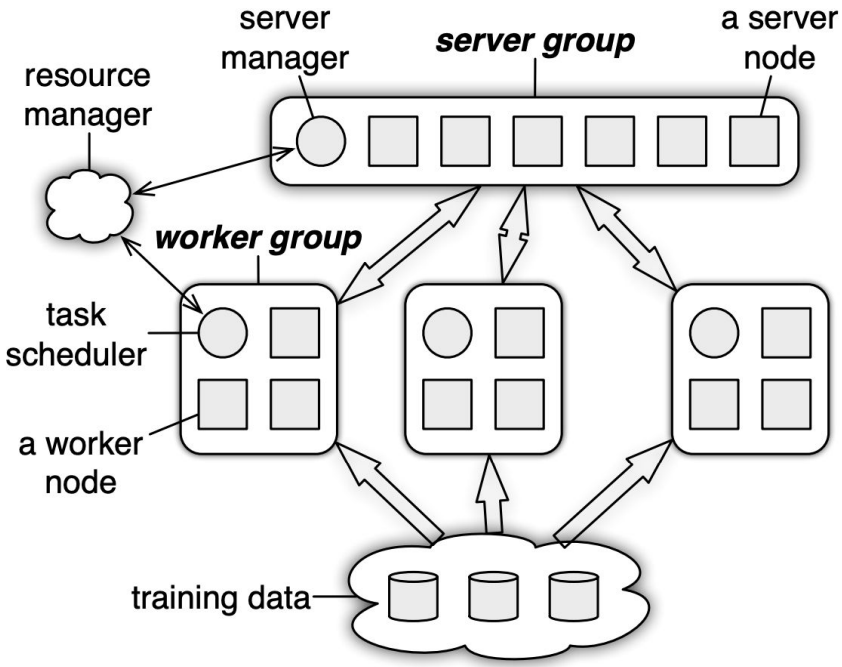
3. How to strengthen the DHT protocol to defend against sybil attacks?

https://tinyurl.com/cse550chord

Other Applications of DHT

Parameter Server for Distributed Machine Learning





[Scaling Distributed Machine Learning with the Parameter Server](#) (OSDI 2014)

Reference and Further reading

- MIT 6.824 notes
  - http://nil.csail.mit.edu/6.824/2017/notes/l-dht.txt
- CMU’s lecture notes
  - https://www.cs.cmu.edu/~dga/15-744/S07/lectures/16-dht.pdf
- Looking Up Data in P2P system
- The Impact of DHT Routing Geometry on Resilience and Proximity