DHT (continued)
Recap

• Nodes arranged in a ring with positions labeled $0..2^m-1$
  • $m = 128, 160, 256$

• A node’s position is based on a hash of its identity
  • $P(\text{collision among } N \text{ nodes}) \approx 1-e^{-n(n-1)/2^{m+1}}$
  • For $m=128$, $N=1,000,000,000$, this is $\approx 0.000000000000000000001469…$
Recap

• Nodes arranged in a ring with positions labeled 0..$2^m-1$

• A node’s position is based on a hash of its identity

• A key $x$ is stored at node with first position greater than $x$ on the ring
  • Each node stores approximately $1/N$ of all keys

File cnn.com/index.html with key K42 stored here
Recap

- Nodes arranged in a *ring* with positions labeled $0..2^m-1$
- A node’s *position* is based on a hash of its identity
- A key $x$ is stored at node with first position greater than $x$ on the ring
- A node’s fingers are based on $id + 2^i \pmod{2^m}$ for $i = 0, ..., m-1$
  - Up to $m$ fingers, though only $O(\log N)$ distinct on average
Recap

• Nodes arranged in a ring with positions labeled 0..2^m-1
• A node’s position is based on a hash of its identity
• A key x is stored at node with first position greater than x on the ring
• A node’s fingers are based on \(id + 2^i \pmod{2^m}\) for \(i = 0, \ldots, m-1\)
• Search for key x proceeds by using largest finger that makes progress towards key
Analysis

Search takes $O(\log(N))$ time

Proof

- (intuition): *at each step, distance between query and peer-with-file reduces by a factor of at least 2* (why?)
  
  Takes at most $m$ steps: $2^m$ is at most a constant multiplicative factor above $N$, lookup is $O(\log(N))$

- (intuition): after $\log(N)$ forwardings, distance to key is at most $2^m / N$ (why?)
  
  Number of node identifiers in a range of $2^m / N$
  is $O(\log(N))$ with high probability (why?)
  
  So using *successors* in that range will be ok
Analysis (contd.)

• $O(\log(N))$ search time holds for file insertions too (in general for routing to any key)
  • “Routing” can thus be used as a building block for
    • All operations: insert, lookup, delete
• $O(\log(N))$ time true only if finger and successor entries correct
• When might these entries be wrong?
  • When you have failures
Search under peer failures

Say \( m = 7 \)

Who has \( \text{cnn.com/index.html} \)?

(hashes to \( K42 \))

File \( \text{cnn.com/index.html} \) with key \( K42 \) stored here

Lookup fails

(N16 does not know N45)
Search under peer failures

Say $m=7$

One solution: maintain $r$ multiple *successor* entries
In case of failure, use successor entries

Who has `cnn.com/index.html`?
(hashes to K42)

File `cnn.com/index.html` with key K42 stored here
Search under peer failures (2)

Say $m=7$

Who has `cnn.com/index.html`?
(hashes to K42)

Lookup fails
(N45 is dead)

File `cnn.com/index.html` with key K42 stored here
Search under peer failures (2)

One solution: replicate file/key at $r$ successors and predecessors

Say $m=7$

Who has \( \text{cnn.com/index.html} \) (hashes to K42)

File \( \text{cnn.com/index.html} \) with key K42 stored here

K42 replicated
Need to deal with dynamic changes

✓ Peers fail
  • New peers join
  • Peers leave
    • P2P systems have a high rate of *churn* (node join, leave and failure)

→ Need to update *successors* and *fingers*, and copy keys
New peers joining

Introducer directs N40 to N45 (and N32)
N32 updates successor to N40
N40 initializes successor to N45, and inits fingers from it

Say $m=7$
New peers joining

Introducer directs N40 to N45 (and N32)
N32 updates successor to N40
N40 initializes successor to N45, and inits fingers from it

*\textit{N40 periodically talks to its neighbors to update finger table}*

\textit{Stabilization Protocol} (to allow for “continuous” churn, multiple changes)

Say $m=7$
Lookups

Average Messages per Lookup

Number of Nodes

log N, as expected
Chord Protocol: Summary

- $O(\log(N))$ memory and lookup costs
- Hashing to distribute filenames uniformly across key/address space
- Allows dynamic addition/deletion of nodes
DHT Deployment

• Many DHT designs
  • Chord, Pastry, Tapestry, Koorde, CAN, Viceroy, Kelips, Kademlia, ...

• Slow adoption in real world
  • Most real-world P2P systems unstructured
    • No guarantees
    • Controlled flooding for routing
  • Kademlia slowly made inroads, now used in many file sharing networks

• Distributed key-value stores adopt some of the ideas of DHTs
  • Dynamo, Cassandra, etc.