Distributed Hash Tables

CSE 486/586: Distributed Systems

Ethan Blanton

Department of Computer Science and Engineering

University at Buffalo

Content Addressing

Globally unique names can eliminate naming authorities.

If all names are globally unique, names cannot collide.

This means we must be able to choose globally unique names.

Content addressing is a solution for this!

In content addressing, an object's name is a cryptographic hash of its contents.

Cryptographic Hash Functions

Cryptographic hash functions compute a hash from an object.

This hash is of small, fixed size no matter the object size.

We will not cover hash functions carefully.

However, they have two properties that we care about:

- Pre-image Resistance: It is infeasible to create an object that maps to a hash, given the hash.
- Collision Resistance: It is extremely unlikely that two objects will hash to the same value.

These allow us to use hashes as globally unique names.

Distributed Hash Tables

Distributed Hash Tables (DHTs) are just what they sound like:

- hash tables (key-indexed key-value stores)
- distributed among multiple processes

Every key in a DHT is stored at one or more processes.

A value is stored alongside each key.

A History of DHTs

The DHT as a general data structure was introduced in 2001.[1]

Three DHT designs came about at about the same time:

- CAN [3]
- Chord [6]
- Pastry [4]

They use different techniques, but have similar properties.

[1] There had been some previous related work on content-addressed storage going back decades!

DHT Properties

All of those first-gen DHTs shared some properties:

- The set of participating nodes (processes) is dynamic
- Finding a key requires an expected number of messages logarithmic in the key space
- Some robustness to bad actors
- Each node must know only a small portionof the key space

They differ moderately to significantly on how they achieve this.

Key Space

Nodes are somehow distributed throughout the key space.

Values are stored (typically) at the closest node to their key.

Closest can be defined in many ways:

- CAN: Cartesian distance on a plane
- Chord: Distance around the circumference of a ring
- Pastry: Similar to Chord

Value Storage

Hypothetically, DHTs can store any value.

In practice, they tend to store small identifiers.

For example, a URL at which a file can be retrieved.

Some DHTs can store multiple values for robustness.

Storing large values is problematic:
The value is stored by *someone else!*

Routing

Nodes are distributed through the same key space as values.

Every node keeps contact information for a small number of neighbor nodes distributed carefully through the key space.

Retrieving the value for a key involves iteratively:

1. Identify the closest node I know to the key
2. Ask that node for the closest node it knows to the key
3. Add that node to my set of known nodes
4. Repeat

Neighbor distribution ensures that every step makes progress.

Uses

DHTs are used in many distributed applications:

- BitTorrent
- Ethereum
- IPFS: The InterPlanetary File System
- Amazon Dynamo (sort of)

Kademlia

Kademlia [2] is a slightly newer (2002) DHT.

It builds on the early DHTs to provide some additional features:

- Nodes automatically learn about new nodes via queries
- It prefers long-lived nodes in routing, protecting it from churn[2] and certain types of attacks

An extension, S/Kademlia, makes it more robust to attacks [1]

Kademlia is used in some high-profile projects (like BitTorrent and Ethereum).

[2]Nodes joining and leaving the network

Operations

Kademlia offers four operations:

Ping: Checks a node address to see if it is online

Store: Stores a value at a node

Find-Node: Returns the *k* nodes known to the request recipient closest to an address.

Find-Value: The same as Find-Node except the node sends only the value if it knows it

Key Space

Kademlia uses a 160-bit key space based on SHA-1 [5].

(It actually doesn't care where the keys come from.)

Choosing node keys can be complicated; S/Kademlia suggests a scheme not dissimilar to Bitcoin Proof-of-Work.

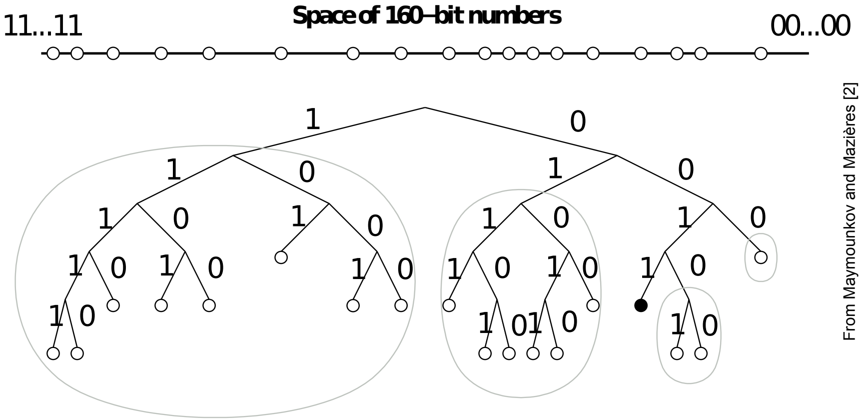
The Kademlia key space is linear.

Each node divides the space into a tree of *k*-buckets.

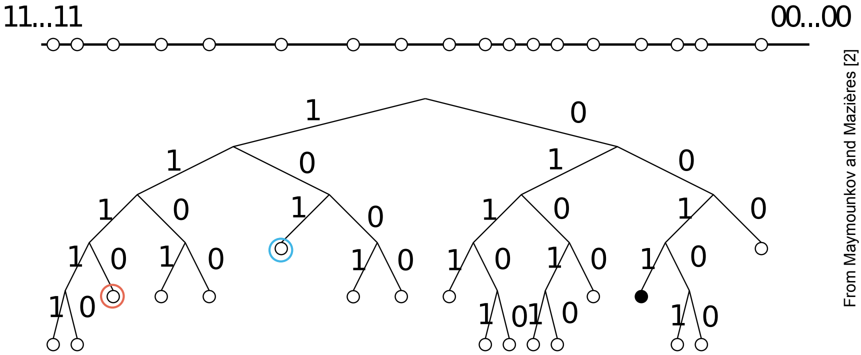
XOR is used to compute distance in key space.

k-Buckets

Suppose a node *n*'s address (key) begins with the bits 0011. . . .

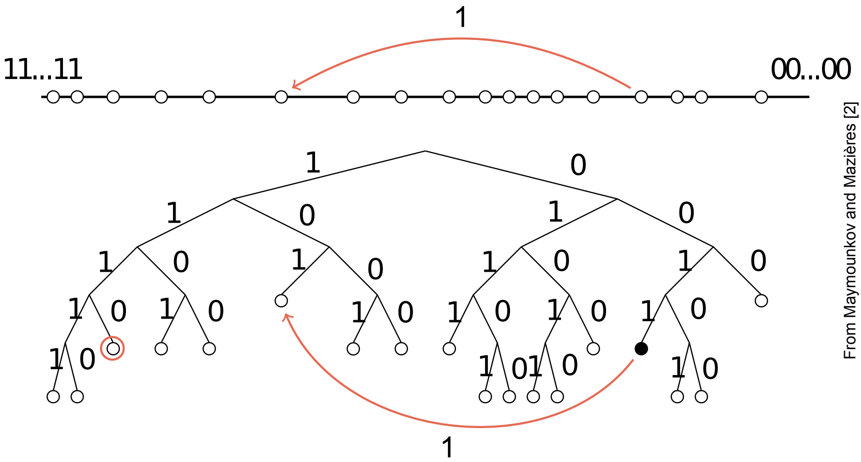


Key Lookup



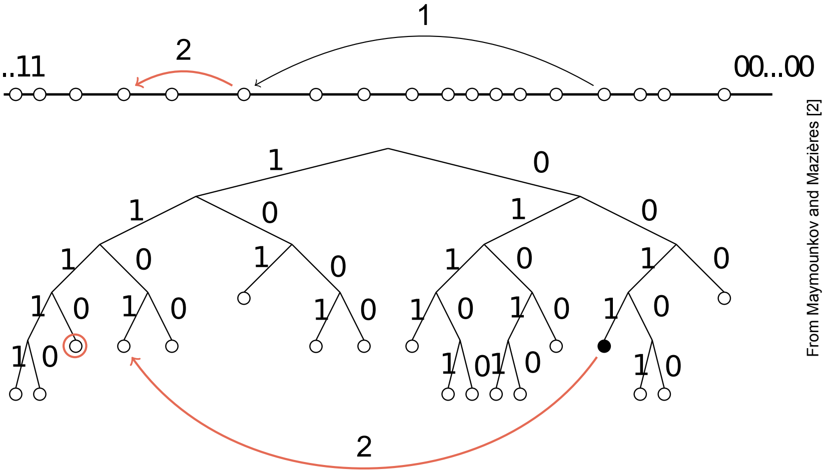
From Maymounkov and Mazières [2]

Key Lookup



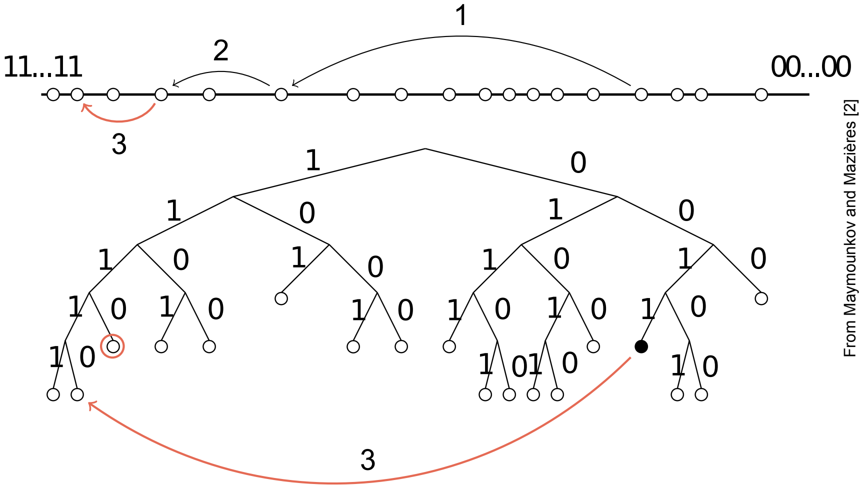
From Maymounkov and Mazières [2]

Key Lookup



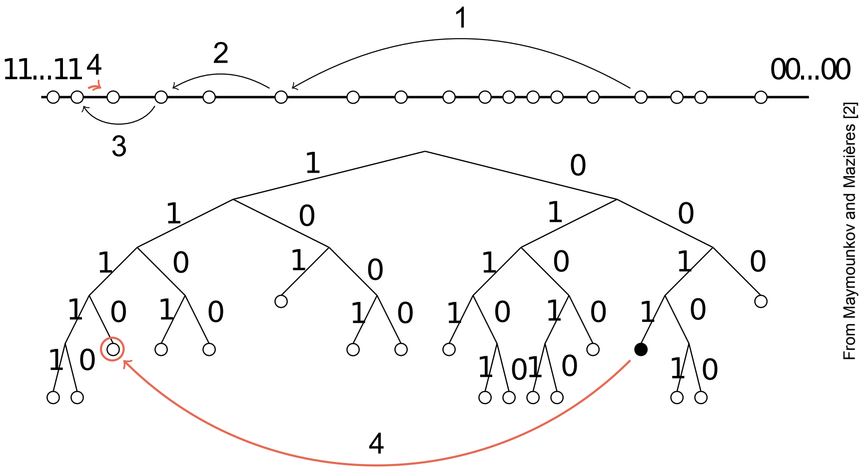
From Maymounkov and Mazières [2]

Key Lookup



From Maymounkov and Mazières [2]

Key Lookup



Joining the DHT

A node joins the DHT by:

1. Selecting its node address *n*
2. Contacting any node in the DHT to look up *n*
3. Inserting the nodes it finds into its *k*-buckets
4. Optionally selecting another node address and repeating

Note that the first lookup will find its closest neighbor!

Nearby nodes store keys that belong to *n* as they discover *n*.

Reliability

Two parameters, k and α, configure Kademlia reliability.

Roughly speaking:

- k controls how resilient it is to churn
- α controls how robust it is to adverserial nodes [1]

We'll look at these two tunables.

Understanding *k*

Every key, value pair is stored at the *k* closest nodes to the key.

This means that *k* nodes can fail before a key is lost.

Every node stores *k* neighbors in the *k*-bucket at each fork of the routing tree.

This means that *k* nodes can fail before a branch is lost.

Therefore, increasing *k* means:

- Data storage is more robust
- More nodes can fail before routing slows down
- Storage costs go up
- Routing tables increase in size

Understanding α

Find-Node or Find-Value is sent to α nodes at each iteration.

This means that:

- $\alpha - 1$ adverserial nodes can give incorrect answers and a key can still be found[3].
- The fastest out of α responses can be used to start the next iteration of lookup.

Therefore, increasing α means:

- More adverserial nodes can be present in the network
- Communication costs go up
- Lookup latency goes down

[3] This is a change in S/Kademlia [1]

Summary

- Distributed hash tables use globally unique names to avoid naming authorities
- Names are often cryptographically secure hash values
- DHTs provide key-value lookups in $O(\log n)$ messages, where n is the size of the key space
- Kademlia is a DHT with desirable properties:
  - Robust to adverserial nodes
  - Deals well with churn
  - Self-maintaining structure
- Kademlia is used in large distributed systems

References I

Required Readings

[2] Petar Maymounkov and David Mazières. “Kademlia: A Peer-to-peer Information System based on the XOR Metric”. In: *Proceedings of the International Workshop on Peer-to-Peer Systems*. Mar. 2002, pp. 53–65. URL: https://pdos.csail.mit.edu/~petar/papers/maymounkov-kademlia-lncs.pdf.

Recommended Readings

References II

[6] Ion Stoica et al. “Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications”. In: *Proceedings of the ACM Special Interest Group on Data Communications*. Aug. 2001, pp. 149–160. URL: http://conferences.sigcomm.org/sigcomm/2001/p12-stoica.pdf.

Optional Readings

[1] Ingmar Baumgart and Sebastien Mies. “S/Kademlia: A Practicable Approach towards Secure Key-Based Routing”. In: *International Conference on Parallel and Distributed Systems*. Dec. 2007, pp. 1–8. DOI: 10.1109/ICPADS.2007.4447808. URL: https://search.lib.buffalo.edu/permalink/01SUNY_BUF/12pkqkt/cdi_ieee_primary_4447808.

References III

[3] Sylvia Ratnasamy et al. “A Scalable Content-Addressable Network”. In: *Proceedings of the ACM Special Interest Group on Data Communications*. Aug. 2001, pp. 161–172. URL: http://conferences.sigcomm.org/sigcomm/2001/p13-ratnasamy.pdf.

[4] Antony Rowstron and Peter Druschel. “Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems”. In: Lecture Notes in Computer Science (Nov. 2001), pp. 329–350. URL: https://search.lib.buffalo.edu/permalink/01SUNY_BUF/12pkqkt/cdi_pascalfrancis_primary_14045913.

References IV

[5] *Secure Hash Standard (SHS)*. Tech. rep. FIPS PUB 180-4. National Institute of Standards and Technology, Aug. 2015. URL: http://dx.doi.org/10.6028/NIST.FIPS.180-4.

To retrieve a copy of this material, or related materials, see `https://www.cse.buffalo.edu/~eblanton/`.