Kademlia

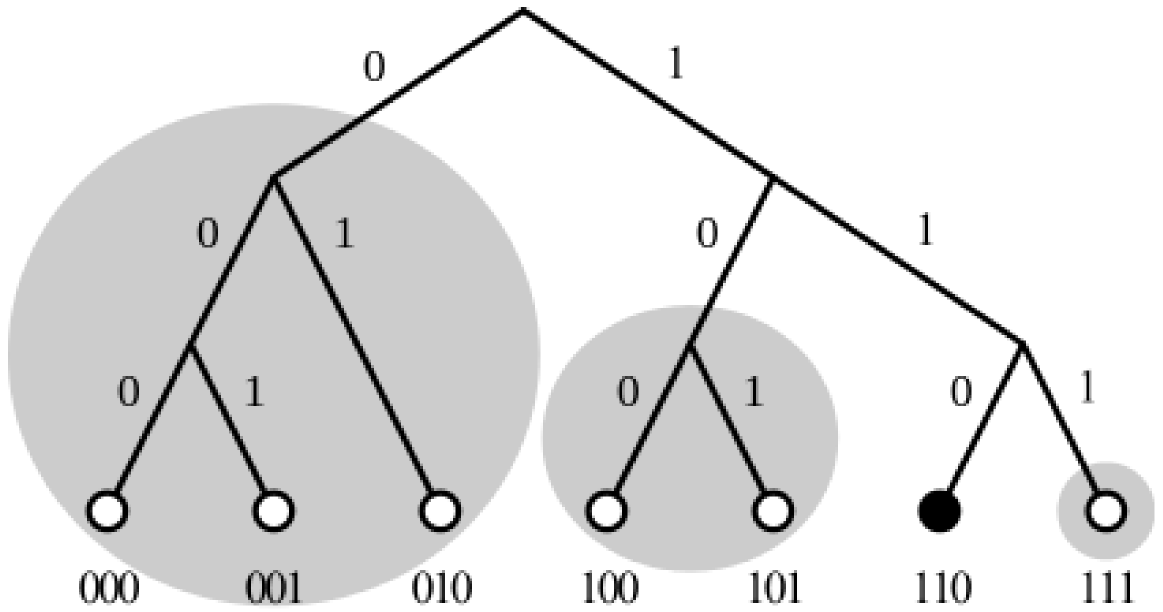
A distributed hash table for decentralized peer-to-peer network

By

Priyabrata Dash (AKA Priyab Satoshi)

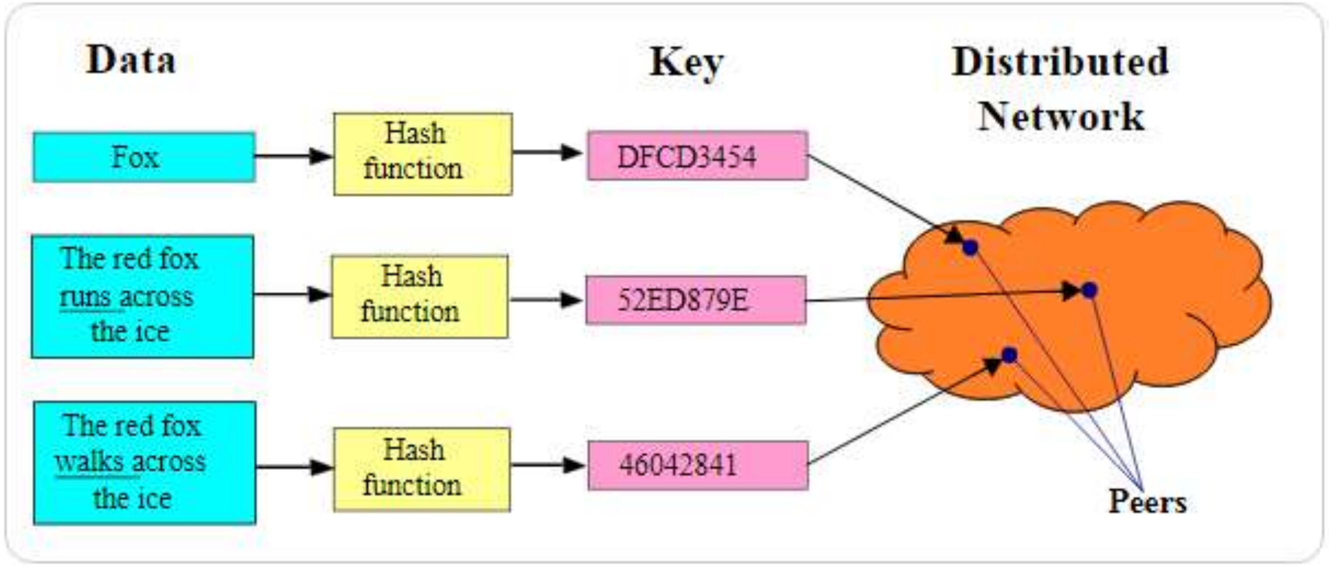
Twitter: twitmyreview

Medium: medium/Crypt-Bytes-Tech



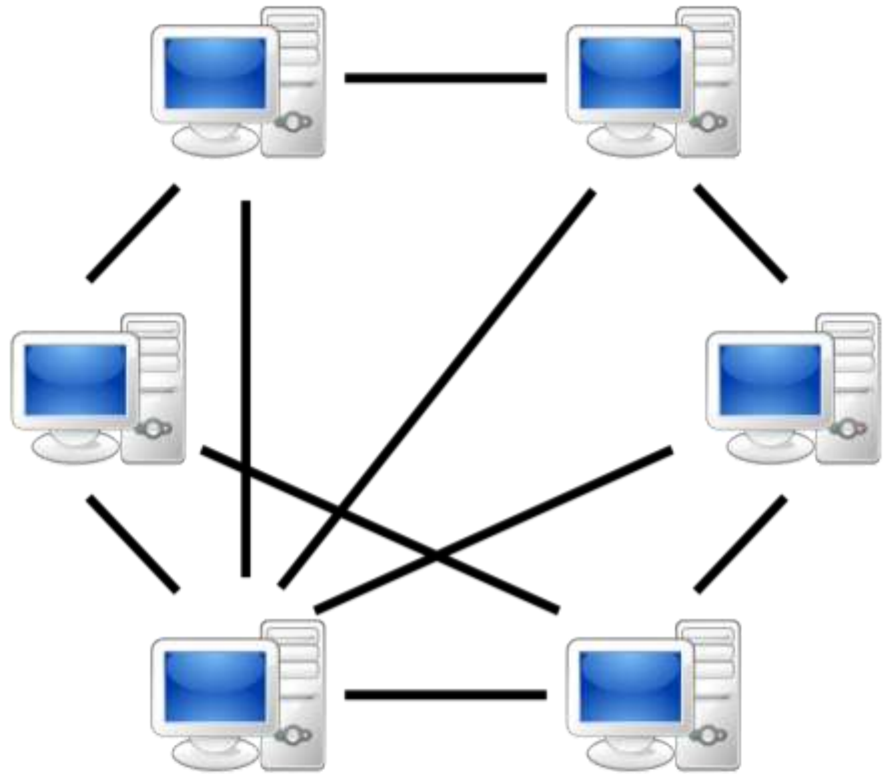
Agenda

- What is Peer to Peer Network
- What is DHT
- Introduction to Kademlia
- System Design
- Academic Significance
- Uses
- Implementations

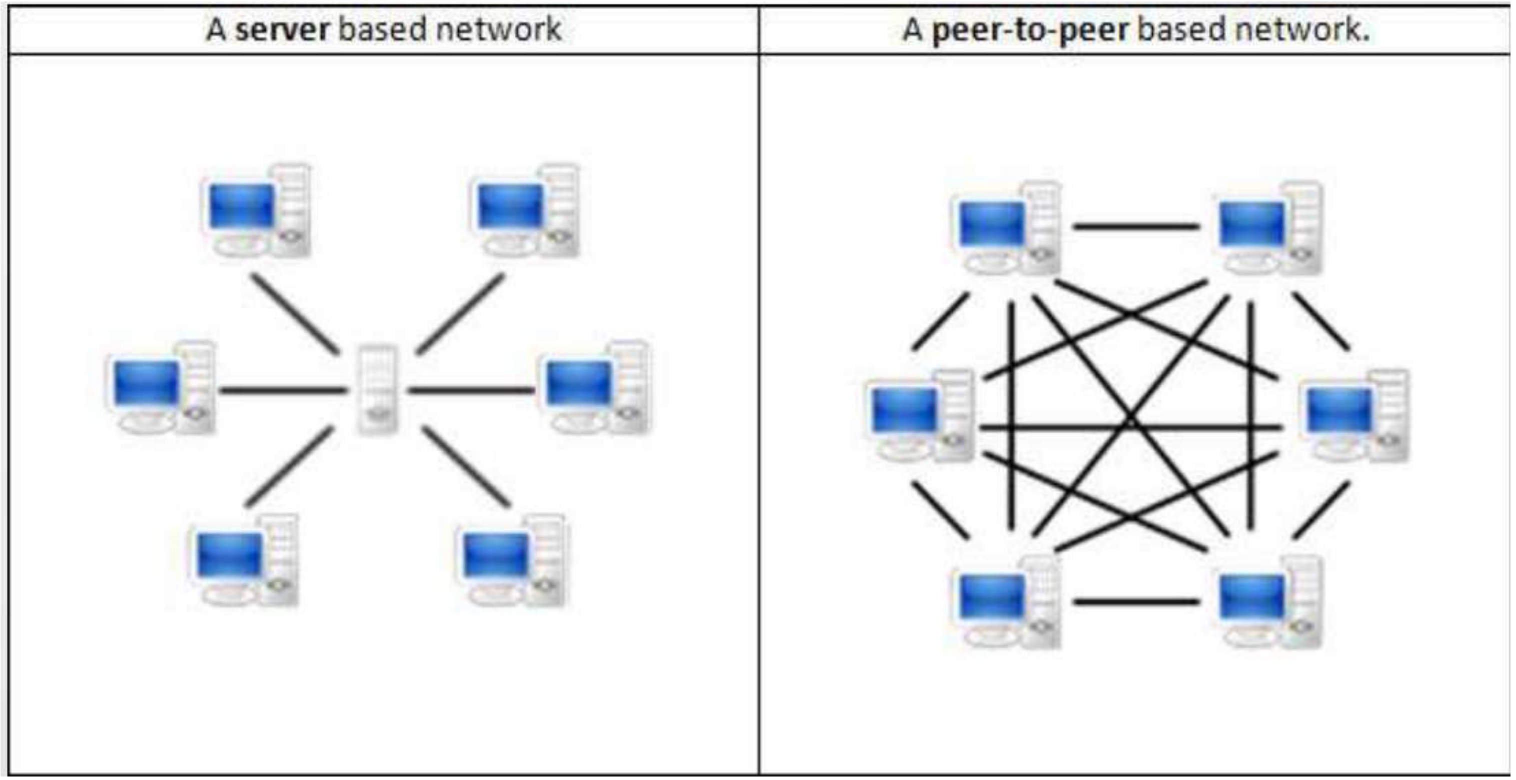


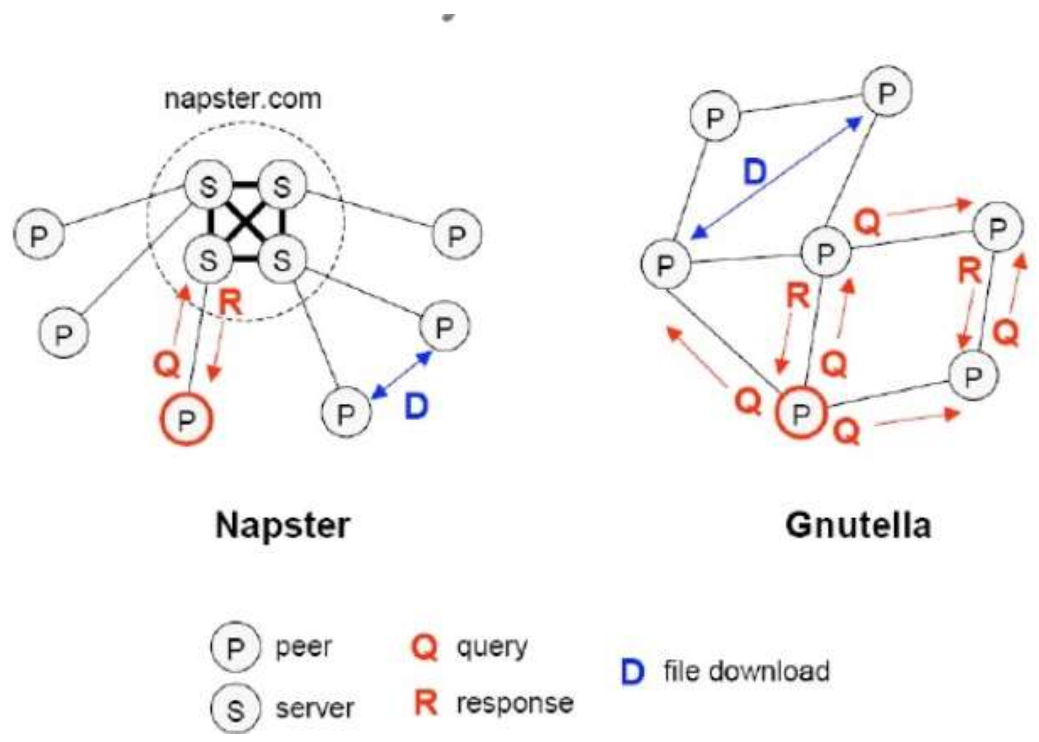
What is Peer to Peer Network

- P2P computing is the sharing of computer resources and services by direct exchange between systems.
- Peers are equally privileged
- Resources
    - processing power
    - disk storage or network bandwidth
- A distributed system architecture
    - No centralized control
    - Typically many nodes, but unreliable and heterogeneous
    - Nodes are symmetric in function
    - Take advantage of distributed, shared resources (bandwidth, CPU, storage) on peer-nodes
    - Fault-tolerant, self-organizing
    - Operate in dynamic environment, frequent join and leave is the norm



PEER-TO-PEER VS CLIENT-SERVER





Types & Categories of P2P Systems

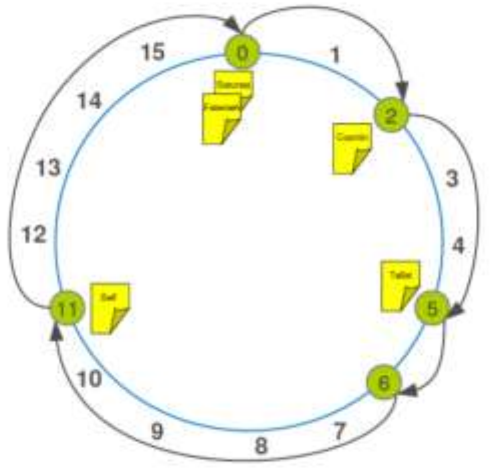
Types

- Pure P2P system: a P2P system that has no central service of any kind
- Hybrid P2P system: a P2P system which depends partially on central servers or allocates selected functions to a subset of dedicated peers

Categories

- Unstructured
  - No restriction on overlay structures and data placement
  - Napster, Gnutella, Kazza, Freenet, Bit torrent
- Structured
  - Distributed hash tables (DHTs)
  - Place restrictions on overlay structures and data placement
  - Chord, Pastry, Tapestry, CAN

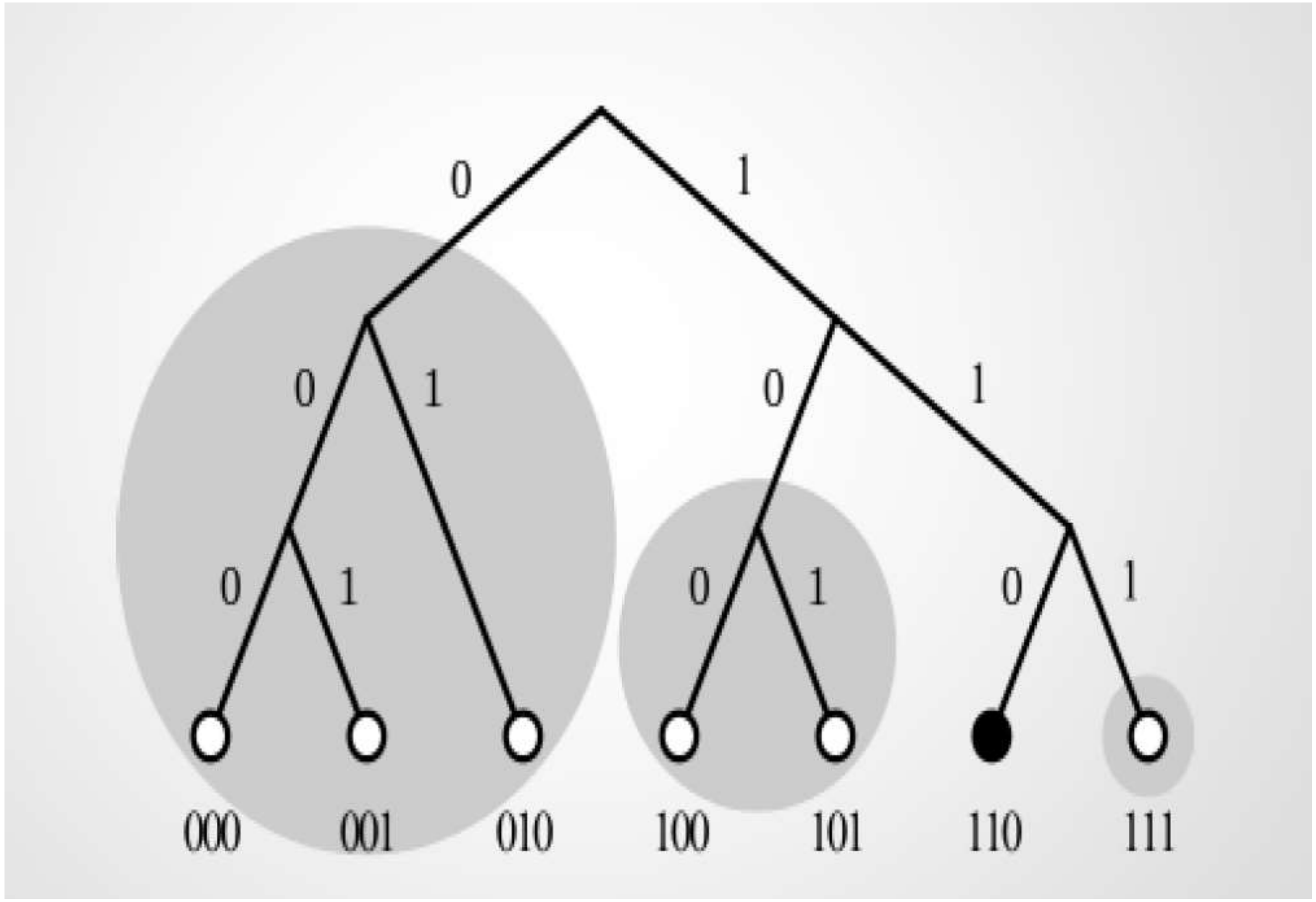
What is DHT (Distributed Hash table)



- A **distributed hash table** (**DHT**) is a class of a decentralized distributed system that provides a lookup service similar to a hash table: (*key*, *value*) pairs are stored in a DHT, and any participating node can efficiently retrieve the value associated with a given key.
- Responsibility for maintaining the mapping from keys to values is distributed among the nodes, in such a way that a change in the set of participants causes a minimal amount of disruption.
- This allows a DHT to scale to extremely large numbers of nodes and to handle continual node arrivals, departures, and failures.
- Distribute data over a large P2P network
  - Quickly find any given item
  - Can also distribute responsibility for data storage
- What's stored is key/value pairs
  - The key value controls which node(s) stores the value
  - Each node is responsible for some section of the space
- Basic operations
  - Store(key, val)
  - val = Retrieve(key)

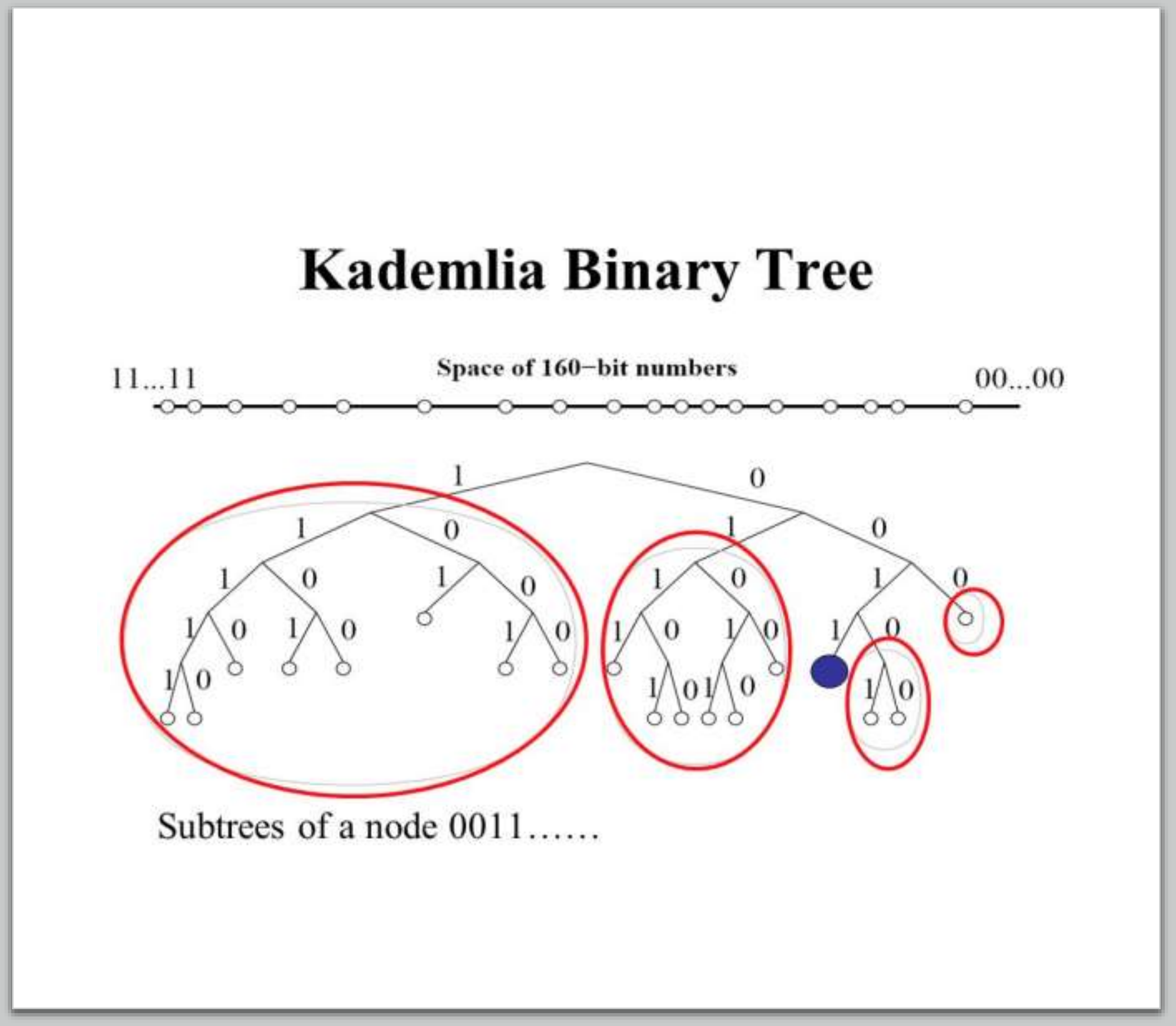
Introduction to Kademlia

- Distributed Hash Table for decentralized peer to peer computer network designed by Petar Maymounkov and David Mazières in 2002
- Specifies the structure of the network and the exchange of information through node lookups.
- Kademlia nodes communicate among themselves using UDP.
- Each node is identified by a number or node ID
- The node ID serves not only as identification, but the Kademlia algorithm uses the node ID to locate values (usually file hashes or keywords).
- The Kademlia network is made up of a wide range of nodes, which interact with each other through User Datagram Protocol (UDP). Each node on the network is identified by a unique binary number called node ID. The node ID is used to locate values (block of data) in the Kademlia algorithm. The values are also interlinked within a Kademlia network with a specific value's key, a binary number of fixed length.
- One major goal of P2P systems is object lookup: Given a data item X stored at some set of nodes in the system, find it. Unlike Chord, CAN, or Pastry Kademlia uses Tree-based routing.



Kademlia Binary Tree

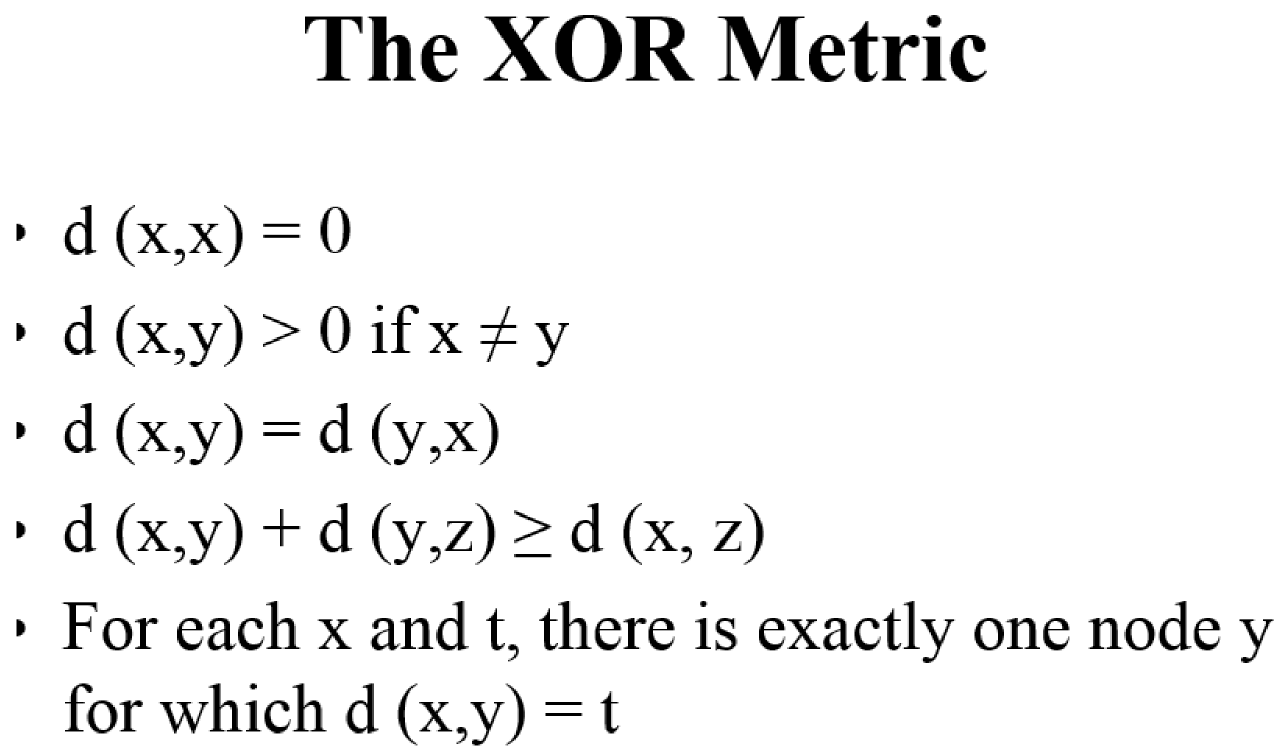
- Treat nodes as leaves of a binary tree.
- Start from root, for any given node, dividing the binary tree into a series of successively lower subtrees that don’t contain the node.
- Every node keeps touch with at least one node from each of its subtrees. (if there is a node in that subtree.) Corresponding to each subtree, there is a k-bucket.
- Every node keeps a list of (IP-address, Port, Node id) triples, and (key, value) tuples for further exchanging information with others


Kademlia Binary Tree

Kad System Design- *Distance Calculation*

- Kademlia uses a "distance" calculation between two nodes
- The closeness between two objects measured as their bitwise XOR interpreted as an integer.

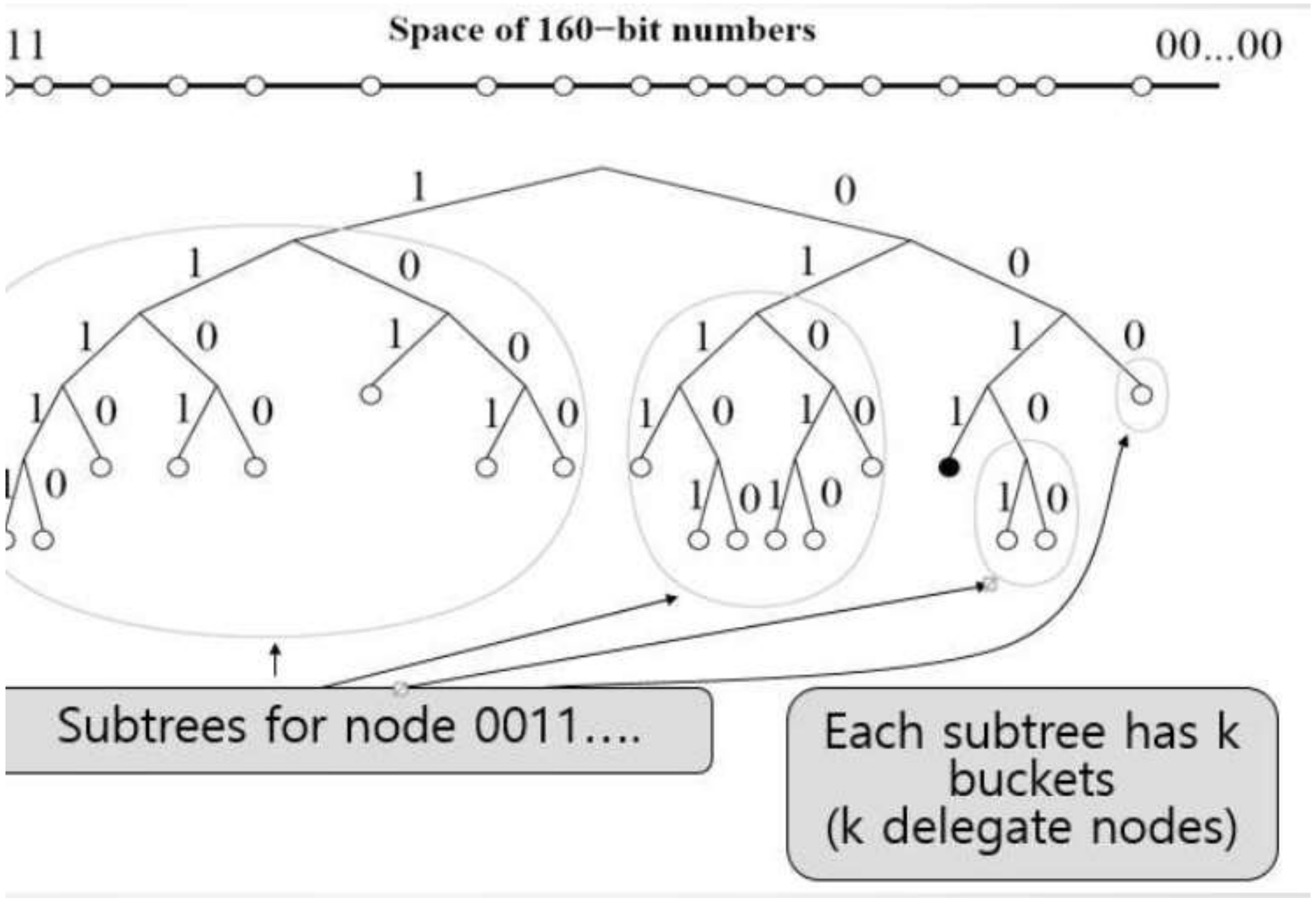
  distance(a, b) = a XOR b
- Distance is computed as the *(XOR)* of the two node IDs
- Keys and Node IDs have the same format and length
- *Exclusive or* was chosen because it acts as a distance function between all the node IDs.
- **Specifically:**
  - The distance between a node and itself is zero
  - It is symmetric: the "distances" calculated from A to B and from B to A are the same
  - It follows the triangle inequality: given A, B and C are vertices (points) of a triangle, then the distance from A to B is shorter than (or equal to) the sum of the distance from A to C and the distance from C to B.
- A **basic** Kademlia network with 2n nodes will only take *n* steps (in the worst case) to find that node.



The XOR Metric

- $d(x,x) = 0$
- $d(x,y) > 0$ if $x \neq y$
- $d(x,y) = d(y,x)$
- $d(x,y) + d(y,z) \geq d(x,z)$
- For each x and t, there is exactly one node y for which $d(x,y) = t$

Kad System Design- *Routing tables*

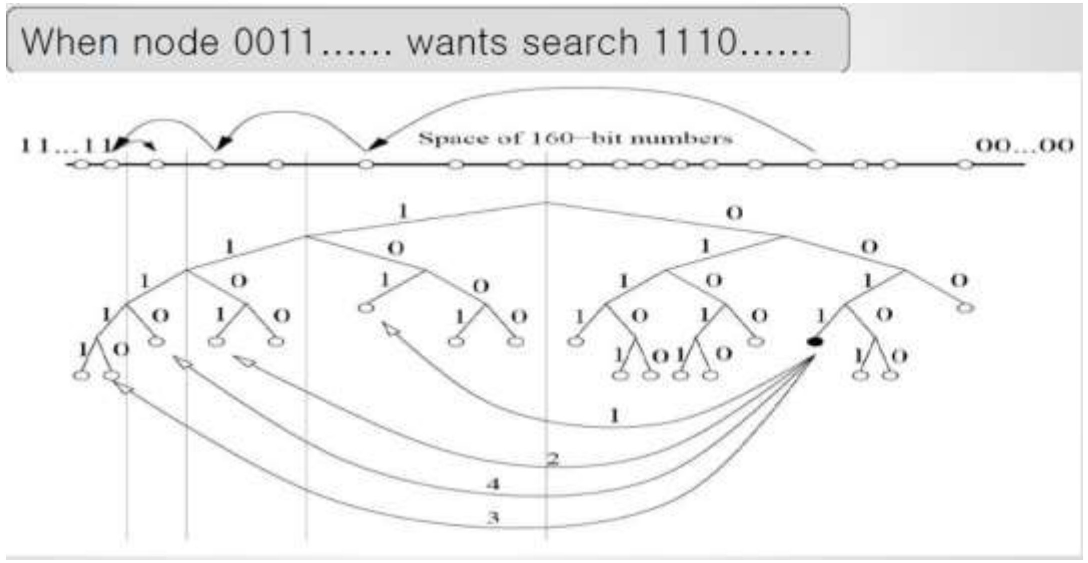


11
Space of 160−bit numbers
00...00
1
0
1
0
1
0
1
0
1
0
1
0
1
0
1
0
1
0
1
0
1
0
1
0
1
0
1
0
1
0
1
0
Subtrees for node 0011....
Each subtree has k buckets (k delegate nodes)

- Nodes, files and key words, deploy SHA-1 hash into a 160 bits space.
- Every entry in a list holds the necessary data to locate another node. Every node maintains information about files, key words close to itself.
- Data in list contains
  IP address, *port*, and *node ID* of another node
- The nth *list* must have a differing nth bit from the node's ID
- The first n-1 bits of the candidate ID must match those of the node's ID
- First *list* as 1/2 of the nodes in the network are far away candidates
- The next *list* can use only 1/4 of the nodes in the network (one bit closer than the first), etc.

Kad System Design- **Protocol messages**

- **Kademlia has four messages**
  - PING — used to verify that a node is still alive.
  - STORE — Stores a (key, value) pair in one node.
  - FIND_NODE — The recipient of the request will return the k nodes in his own buckets that are the closest ones to the requested key.
  - FIND_VALUE — Same as FIND_NODE, but if the recipient of the request has the requested key in its store, it will return the corresponding value.
- Each RPC message includes a random value from the initiator. This ensures that when the response is received it corresponds to the request previously sent
- Every node keeps touch with **at least one node from each of its subtrees**. (if there is a node in that subtree.) Corresponding to each subtree, there is a **k-bucket**.
- Every node keeps a list of (IP-address, Port, Node id) triples, and (key, value) tuples for further exchanging information with others

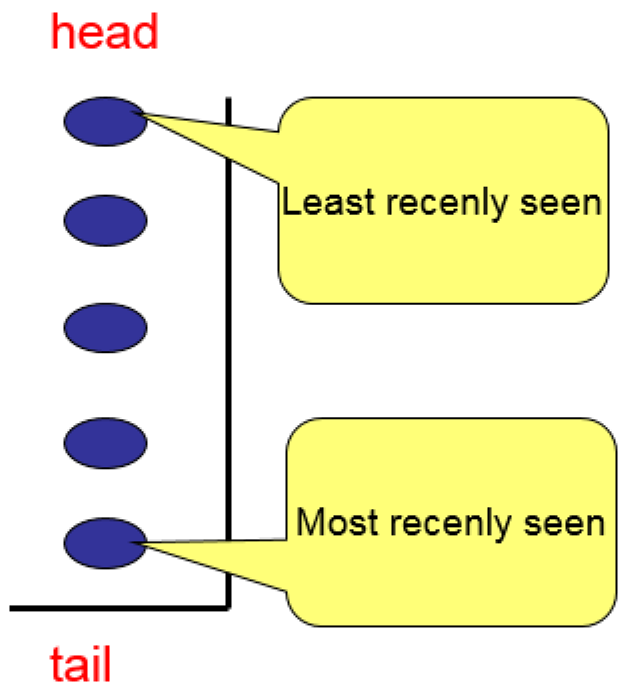


Kad System Design-
Locating nodes

- The most important is to locate the k closest nodes to some given node ID.
- Kademlia employs a recursive algorithm for node lookups. The lookup initiator starts by picking a nodes from its closest non-empty k-bucket.
- The initiator then sends parallel, asynchronous FIND_NODE to the α nodes it has chosen.
- α is a system-wide concurrency parameter, such as 3.
- the initiator resends the FIND_NODE to nodes it has learned about from previous RPCs.
- If a round of FIND_NODES fails to return a node any closer than the closest already seen, the initiator resends the FIND_NODE to all of the k closest nodes it has not already queried.
- The lookup could terminate when the initiator has queried and gotten responses from the k closest nodes it has seen.

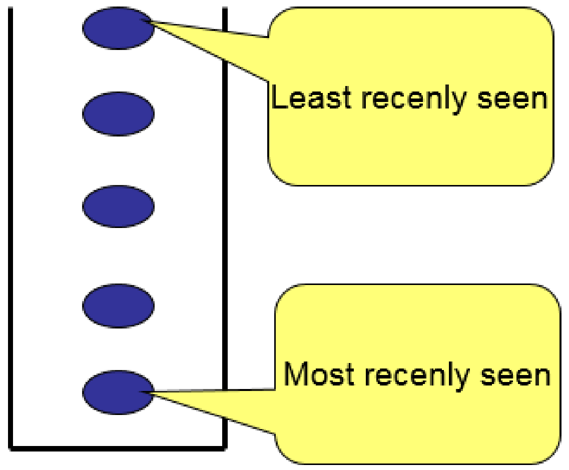
Kad System Design- **Locating resources**

- Information is located by mapping it to a key. A hash is typically used for the map. The storer nodes will have information due to a previous STORE message. Locating a value follows the same procedure as locating the closest nodes to a key, except the search terminates when a node has the requested value in his store and returns this value.
- The values are stored at several nodes (k of them) to allow for nodes to come and go and still have the value available in some node. Periodically, a node that stores a value will explore the network to find the k nodes that are close to the key value and replicate the value onto them. This compensates for disappeared nodes.
- Also, for popular values that might have many requests, the load in the storer nodes is diminished by having a retriever store this value in some node near, but outside of, the k closest ones. This new storing is called a cache. In this way the value is stored farther and farther away from the key, depending on the quantity of requests. This allows popular searches to find a storer more quickly. Because the value is returned from nodes farther away from the key, this alleviates possible "hot spots". Caching nodes will drop the value after a certain time depending on their distance from the key.
- Some implementations (e.g. Kad) do not have replication nor caching. The purpose of this is to remove old information quickly from the system. The node that is providing the file will periodically refresh the information onto the network (perform FIND_NODE and STORE messages). When all of the nodes having the file go offline, nobody will be refreshing its values (sources and keywords) and the information will eventually disappear from the network.



Kad System Design- **Joining the network**

- A node that would like to join the net must first go through a bootstrap process. In this phase, the joining node needs to know the IP address and port of another node—a bootstrap node (obtained from the user, or from a stored list)—that is already participating in the Kademlia network. If the joining node has not yet participated in the network, it computes a random ID number that is supposed not to be already assigned to any other node. It uses this ID until leaving the network.
- The joining node inserts the bootstrap node into one of its *k-buckets*. The joining node then does a FIND_NODE of its own ID against the bootstrap node (the only other node it knows). The "self-lookup" will populate other nodes' *k-buckets* with the new node ID, and will populate the joining node's k-buckets with the nodes in the path between it and the bootstrap node. After this, the joining node refreshes all *k-buckets* further away than the k-bucket the bootstrap node falls in. This refresh is just a lookup of a random key that is within that *k-bucket* range.
- Initially, nodes have one *k-bucket*. When the *k-bucket* becomes full, it can be split. The split occurs if the range of nodes in the *k-bucket* spans the node's own id (values to the left and right in a binary tree). Kademlia relaxes even this rule for the one "closest nodes" *k-bucket*, because typically one single bucket will correspond to the distance where all the nodes that are the closest to this node are, they may be more than k, and we want it to know them all. It may turn out that a highly unbalanced binary sub-tree exists near the node. If *k* is 20, and there are 21+ nodes with a prefix "xxx0011....." and the new node is "xxx000*11001*", the new node can contain multiple *k-buckets* for the other 21+ nodes. This is to guarantee that the network knows about all nodes in the closest region.



How is the bucket updated?

Academic Significance & benefits

- While the XOR metric is not needed to understand Kademlia, it is critical in the analysis of the protocol.
- The XOR arithmetic forms an abelian group allowing closed analysis. Other DHT protocols and algorithms require simulation or complicated formal analysis in order to predict network behavior and correctness.
- Using groups of bits as routing information also simplifies the algorithms.
- Since the network is distributed, by definition, there's no one master table of ID->address mappings.
- Nodes don't have to (and usually don't) know about all the other nodes.
- The process of "finding" a node is basically to ask known nodes "closest" to the target not so much about the target node directly, but about *what nodes are closer* to the target.
- The result of that query gives you the next group of nodes to query, and the process repeats -- and because a node would return results that are closer than it is, each iteration tends to find nodes closer and closer to the target till you finally reach a node that can say "Oh, node X? He's right over there.“
- The nodes in the k-buckets are the stepping stones of routing.
- By relying on the oldest nodes, k-buckets promise the probability that they will remain online.
- DoS attack is prevented since the new nodes find it difficult to get into the k-bucket

Uses

- Some of the Use cases of Kademlia are given below:
  - File Sharing
  - Botnet
  - File System
  - IP Overlay Routing
  - P2P Network Discovery
  - Peer to Peer Messaging
- Kademlia is used in file sharing networks.
  - If a node wants to share a file, it processes the contents of the file, calculating from it a number (hash) that will identify this file within the file-sharing network
  - Hashes and the node IDs must be of the same length
  - It then searches for several nodes whose ID is close to the hash
  - A searching client will use Kademlia to search the network for the node whose ID has the smallest distance to the file hash, then will retrieve the sources list that is stored in that node

Implementation

- I2P[7]
- Kad Network — developed originally by the eMule community to replace the server-based architecture of the eDonkey2000 network.
- Ethereum — The node discovery protocol in Ethereum's blockchain network stack is based a slightly modified implementation of Kademlia.[8]
- Overnet network: With KadC a C library for handling its Kademlia is available. (development of Overnet is discontinued)
- BitTorrent Uses a DHT based on an implementation of the Kademlia algorithm, for trackerless torrents.
- Osiris sps (all version): used to manage distributed and anonymous web portal.
- Retroshare — F2F decentralised communication platform with secure VOIP, instant messaging, file transfer etc.
- Tox - A fully distributed messaging, VoIP and video chat platform
- Gnutella DHT — Originally by LimeWire[9][10] to augment the Gnutella protocol for finding alternate file locations, now in use by other gnutella clients.[11]
- IPFS – A peer-to-peer distributed filesystem.[12]
- TeleHash is a mesh networking protocol that uses Kademlia to resolve direct connections between parties.[13]
- IPFS
- OpenDHT

Reference

- Pease refer below link to access some of the references I used for this presentation: https://www.reddit.com/r/indiacryptogrp/comments/8ej9zv/reading_list_kademlia/

Thank You & QA