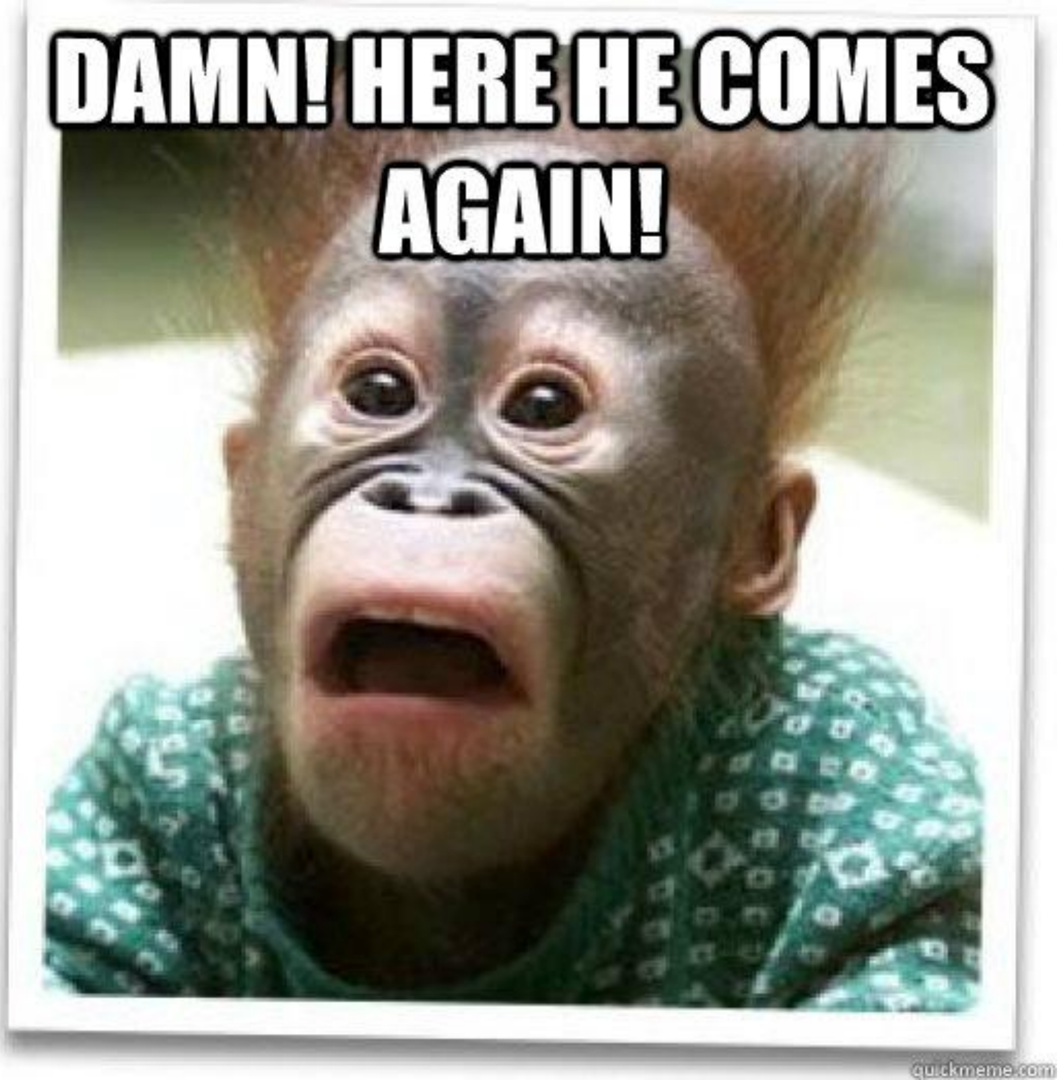
P2P how and Kademlia

- Kishan Sagathiya, @kishansagathiya

Software Engineer at Protocol Labs,
Member of IPFS

Speaking again at Golang Meetup



What is a P2P system

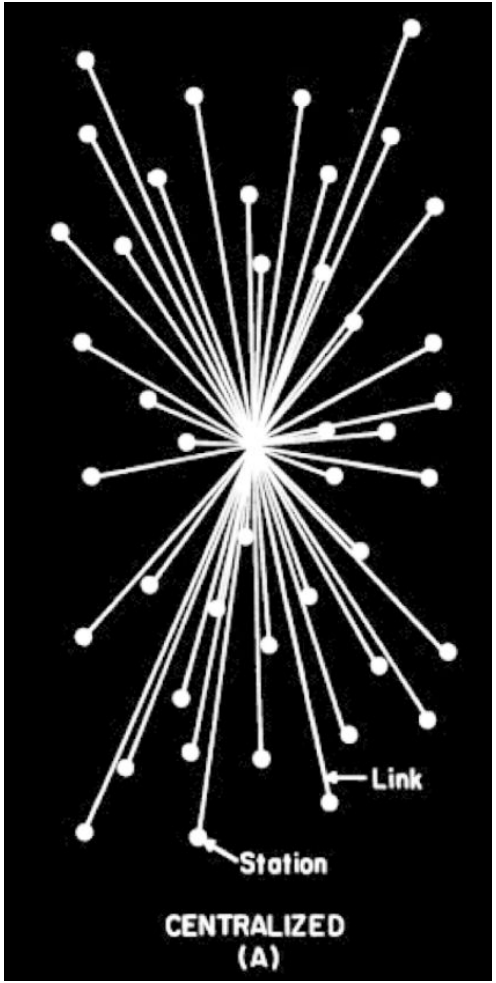
- All node are equal
- No center
- Network of nodes talking to each other
- There are no server

Examples, IPFS, Ethereum, Bitcoin, Tor, Bittorrent

Client-Server World

ipfs.io -> Domain Name System -> 209.94.90.1

- Domain Name System consists of a network of Domain Name Servers
- Databases with domain name to IP info
- First contact normally are ISPs



DNS

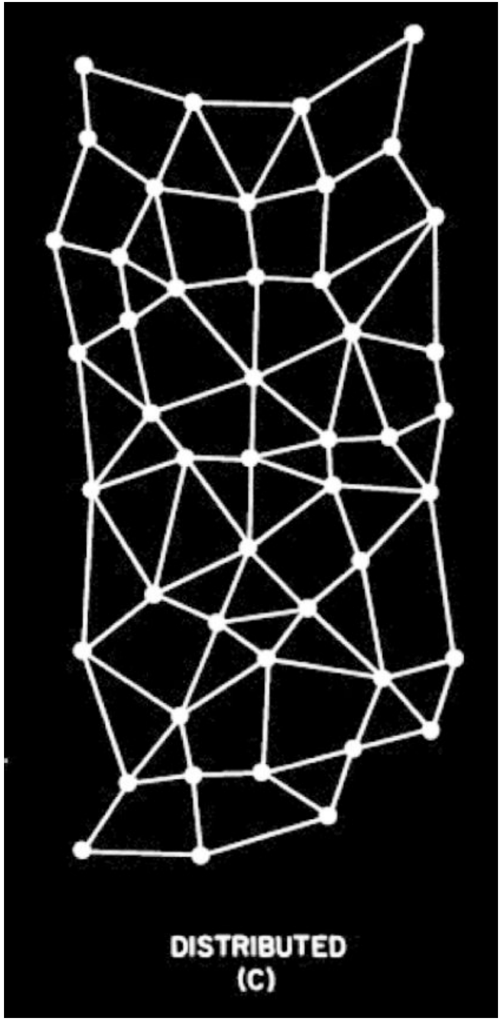
- Hierarchical lookup
- Root is critical
- Run in serious secure data center

They work pretty well, but they are not decentralized

P2P World

Often hash -> IP Address

IPFS, Ethereum, Bittorent etc.



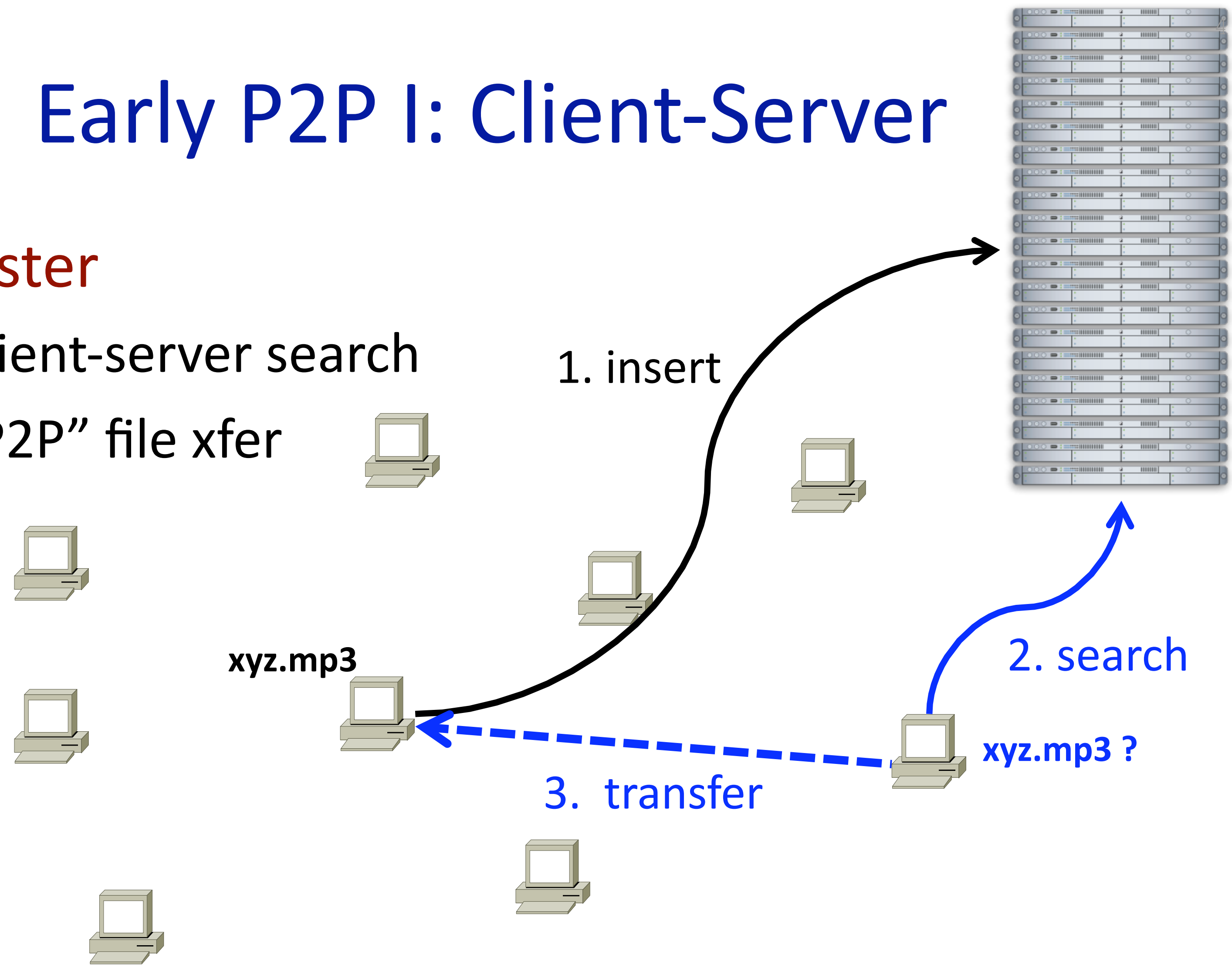
P2P as Overlay Networking

- P2P applications need to:
  - Track identities & IP addresses of peers
    - May be many and may have significant churn
  - Route messages among peers
    - If you don't keep track of all peers, this is "multi-hop"
- *Overlay network*
  - Peers doing both naming and routing
  - IP becomes "just" the low-level transport

Early P2P

Early P2P I: Client-Server

- Napster
  - Client-server search
  - "P2P" file xfer



Napster

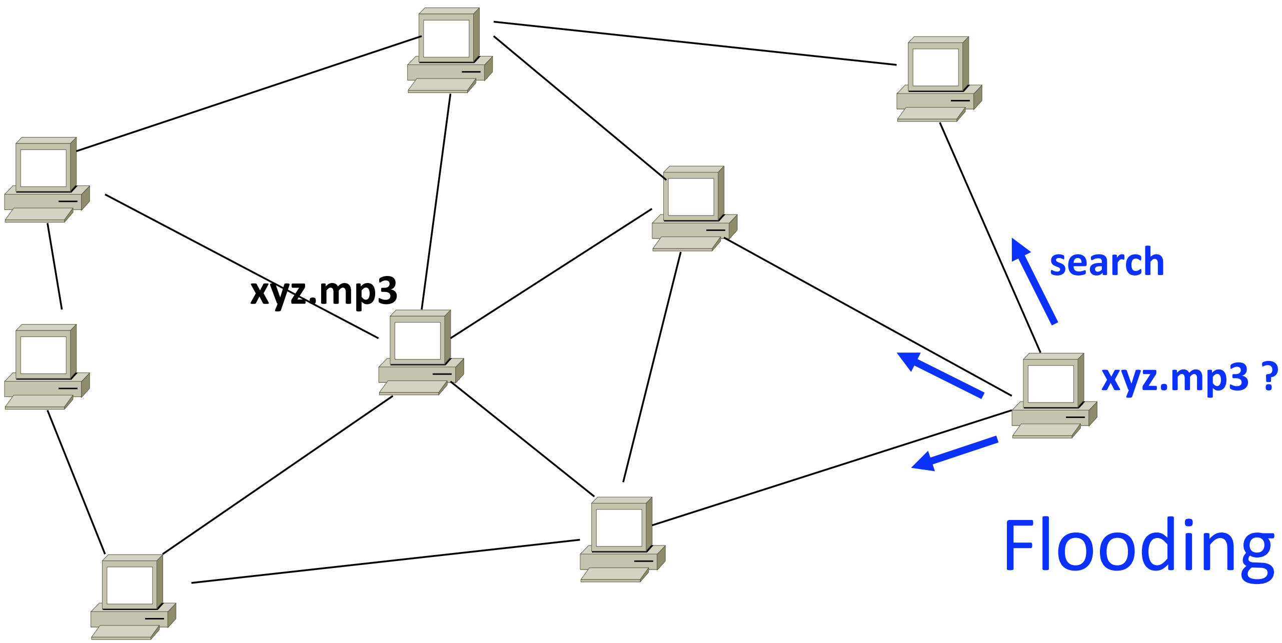
- Data on peers
- Server containing information about file location

“Despacito” → 11.23.45.67

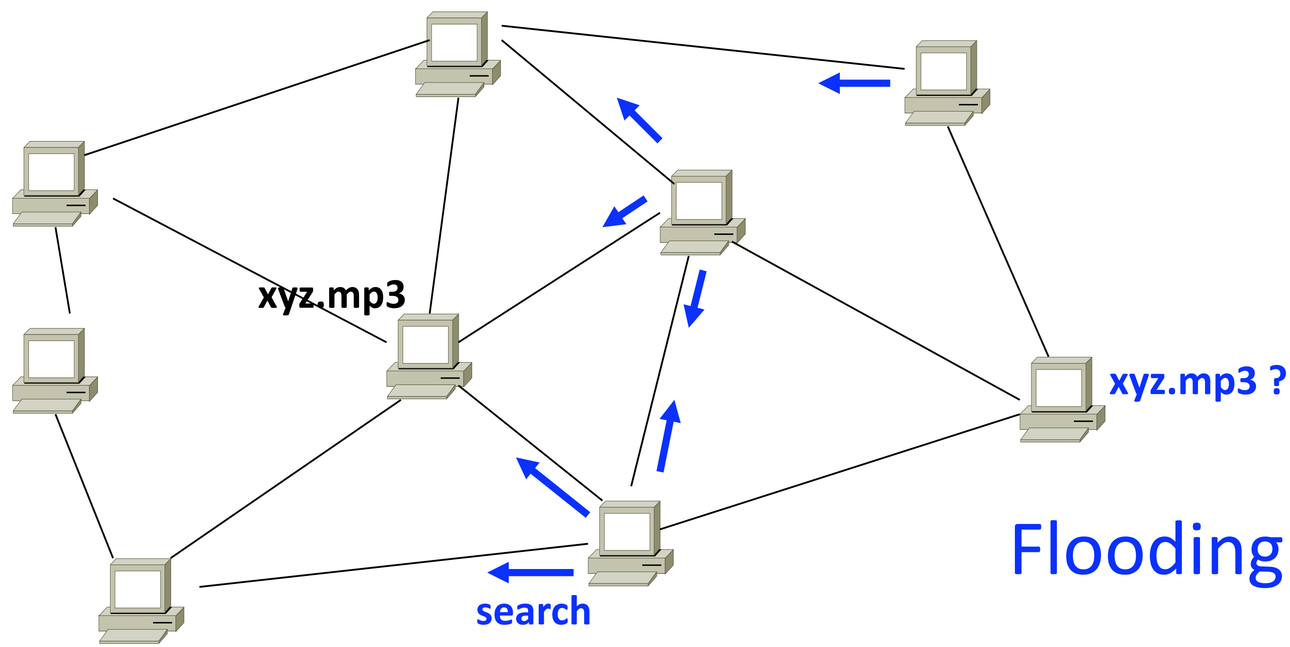
Point of Centralization

Legal center

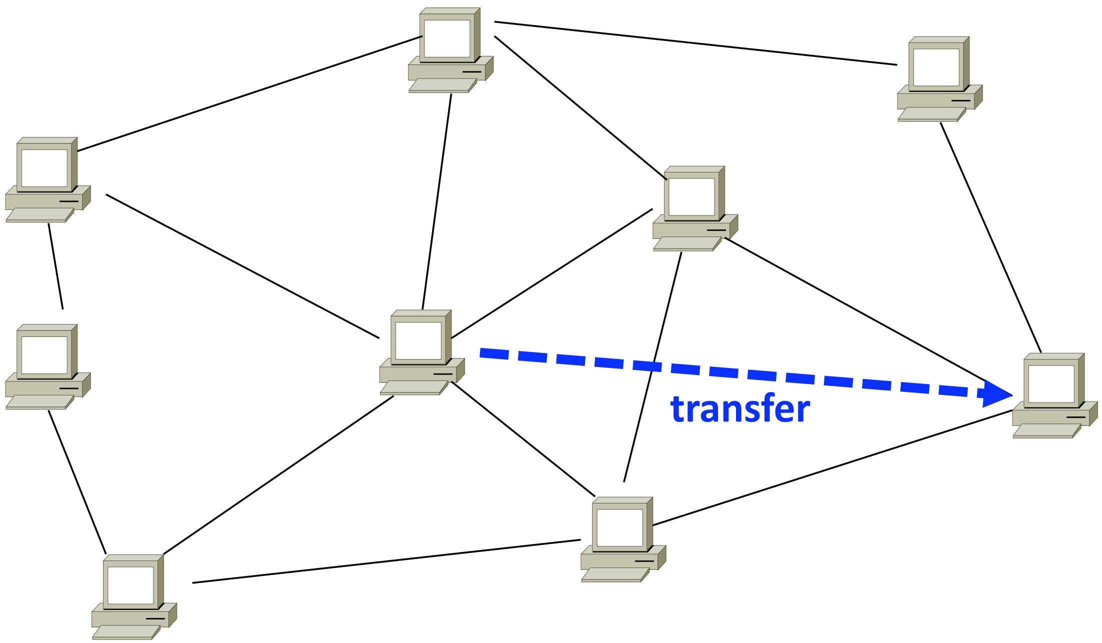
Early P2P II: Flooding on Overlays



Early P2P II: Flooding on Overlays

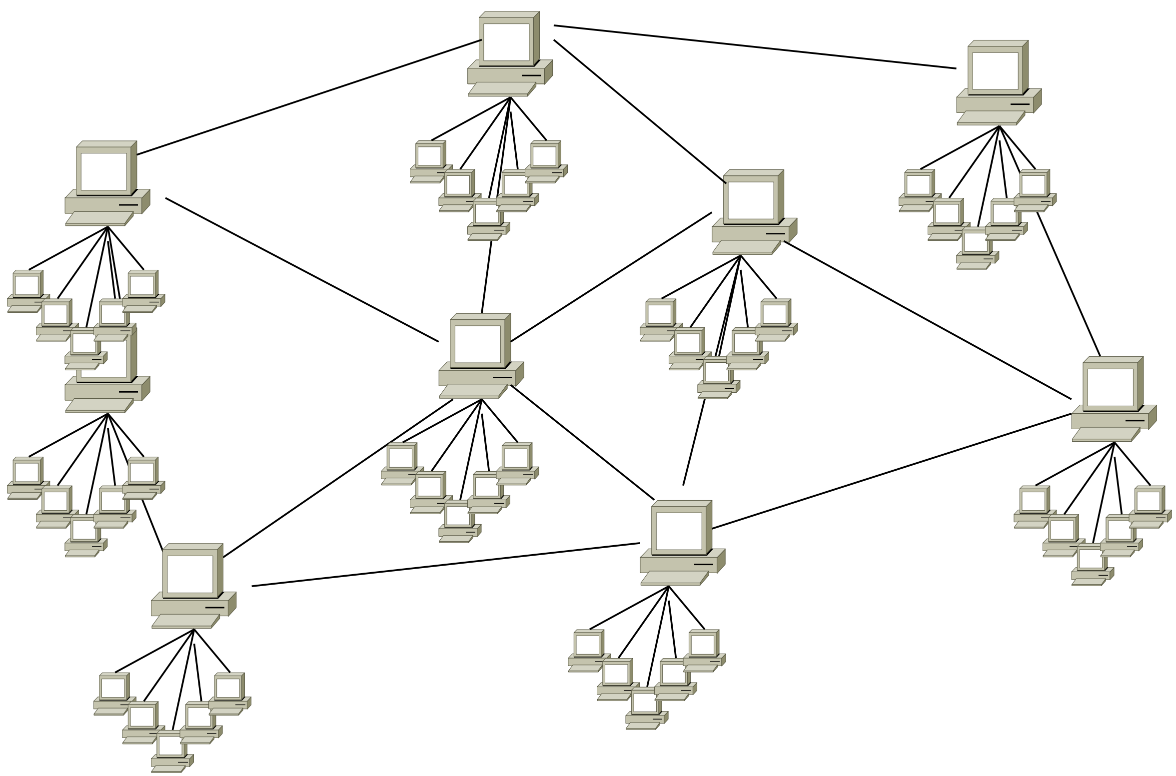


Early P2P II: Flooding on Overlays



Early P2P II: "Ultra/super peers"

- Ultra-peers can be installed (KaZaA) or self-promoted (Gnutella)
  - Also useful for NAT circumvention, e.g., in Skype



Lessons and Limitations

- Client-Server performs well
  - But not always feasible: Performance not often key issue!
- Things that flood-based systems do well
  - Organic scaling
  - Decentralization of visibility and liability
  - Finding popular stuff
  - Fancy *local* queries
- Things that flood-based systems do poorly
  - Finding unpopular stuff
  - Fancy *distributed* queries
  - Vulnerabilities: data poisoning, tracking, etc.
  - Guarantees about anything (answer quality, privacy, etc.)

Structured Overlays:
Distributed Hash Tables

Hash Tables

- Contains a Key-Value pair
- If a user supplies a key, hash table returns a value

insert(key, value)

lookup(key)

delete(key)

Time Complexity approx $\theta(1)$ (upper bound)

Distributed Hash table

are distributed over the network

Distributed such that….

Cheap operations

- Insertion/deletion
- Key movement
- lookups

Small routing table size

Store resource locations

I present to you

Kademlia

Core ideas

- Uniform ID Space
- Closeness
- Local view

Uniform ID Space

$$0 \longleftrightarrow 2^{256} - 1$$

0x00...00 0xff...ff

- sha256 Address space
- for files and nodes.

Uniform ID Space

- Can use SHA512 as well, or any other hash with large enough range
- But it must always give unique values

- larger than number of atoms in the universe

Closeness

- Distance is calculated with exclusive or (XOR)

Could be between

- two nodes
- a node and a key(what we use to locate files)

$$\begin{array}{r} 5 \\ \wedge \quad 3 \\ \hline 6 \end{array} \quad = \quad \begin{array}{r} 0\,1\,0\,1 \\ \wedge\ 0\,0\,1\,1 \\ \hline 0\,1\,1\,0 \end{array}$$

Closeness

- Geographically distant nodes can be possibly be `neighbours`

Local View

- A node know more about its neighbourhood

Protocol Messages

- PING
- STORE
- 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

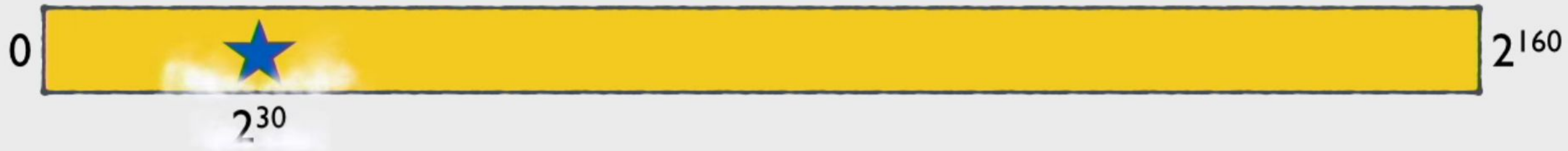
Joining the network

- IP and port of at least one node in the network
- Compute a random ID
- Add bootstrap node to a k-bucket

- Informations about other nodes are stored in buckets of size k (say 16)

Joining the network

- Run a FIND_NODE of you own ID against bootstrap node
- Self-lookup will populate other nodes buckets with your node ID
- And yours with nodes in the path
- Refresh with lookup of any random key within the k-bucket range



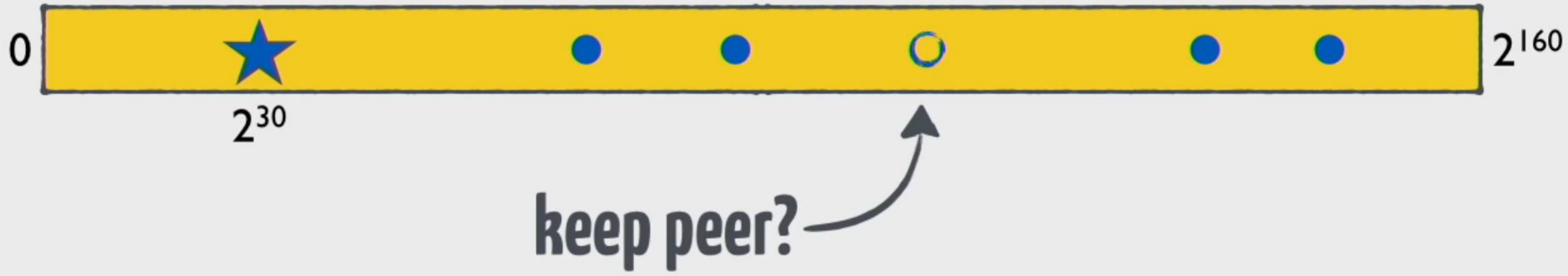
Range of your bucket is 0 to 2^160

Let’s keep K=4

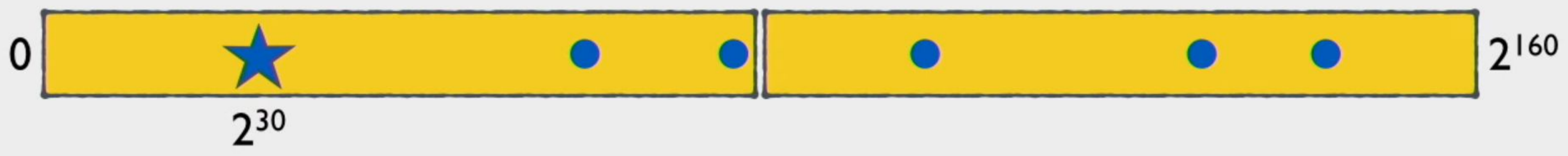
Bucket size can’t surpass K

- if new peer would make bucket K+1
  - if our ID is in the bucket
    - split the bucket, & add the new peer
  - else
    - ping all peers in bucket
    - if some peer is dead
      - replace dead peer with new peer
    - else
      - throw away new peer

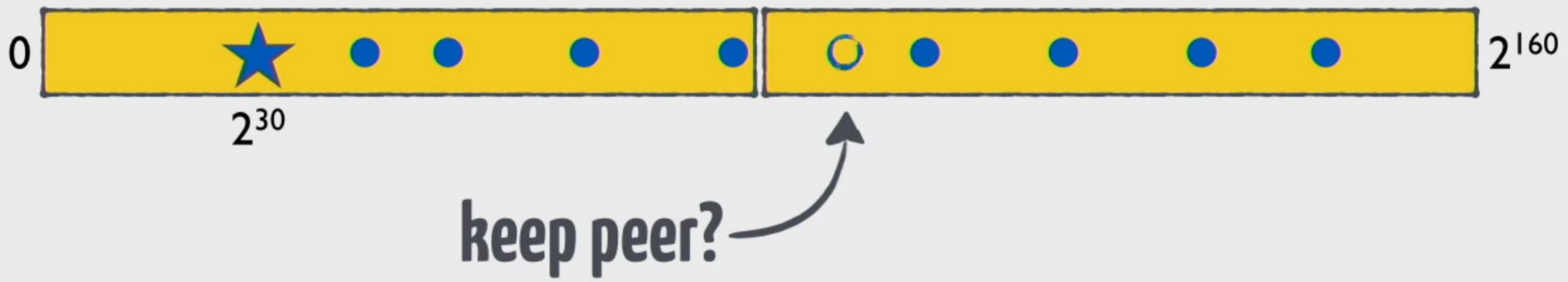
- Got new nodes
- Don’t count yourself



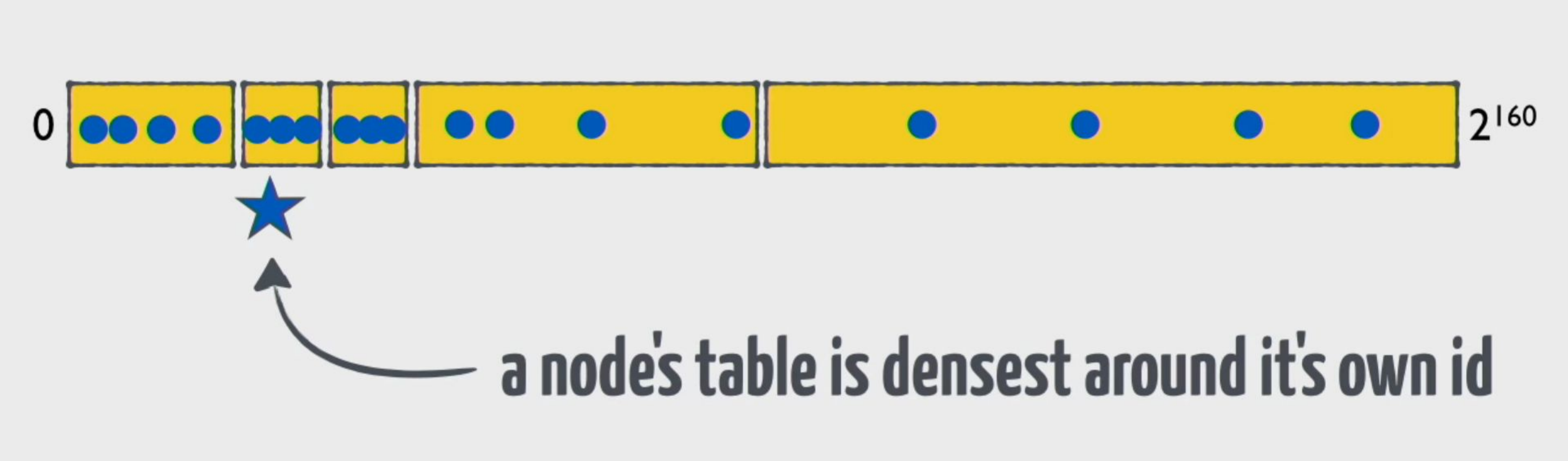
Yes



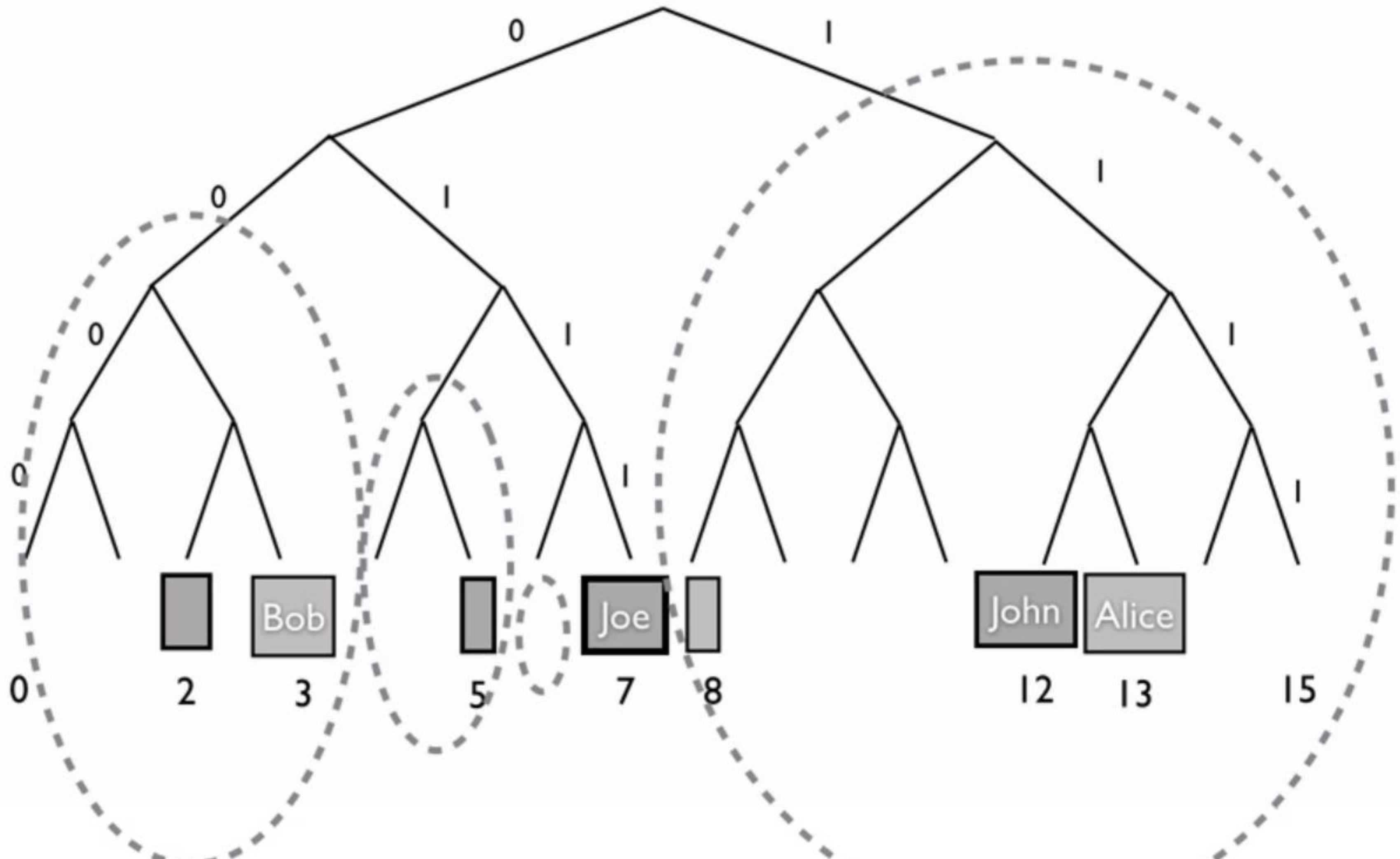
Split in half (by range)



No



More info about nodes near you (Local view)



0
1
0
1
1
0
1
1
0
1
1
Bob
Joe
John
Alice
0
2
3
5
7
8
12
13
15

Joe

Dest	Node
0111	7 (self)
011x	_
01xx	5
0xxx	2
xxxx	12

John

	Dest	Node
	1100	12 (self)
13 (1101)	110x	13
	11xx	_
	1xxx	8
	xxxx	7

Locating nodes

- FIND_NODE against `alpha` closest nodes from it’s own k-buckets
- Recipients will return k closest nodes from it’s own k-buckets to the desired key
- Requester will update with the results, select k closest nodes and make queries to them

Locating nodes

- Iteration stops when you stop getting any new closer nodes
- When iteration stops, best k nodes in the result list are closest nodes to that key from the whole network

Locating nodes

- log(n) complexity

Locating Resources

- Information is located by mapping it to key
- Same as locating nodes
- But if the requested value is available in your store, return the value

Storing

- Stored at several k-nodes to allow nodes to come and go
- Node will explore it’s own network to find k-closest nodes to the key and replicate the value onto them

- Not all implementation would have replication


TALK IS CHEAP
SHOW ME THE CODE

go-libp2p-kad-dht

- implements a distributed hash table that satisfies the ipfs routing interface. This DHT is modeled after kademlia with S/Kademlia modifications.
- https://github.com/libp2p/go-libp2p-kad-dht

- Lookups https://github.com/libp2p/go-libp2p-kad-dht/blob/b99a6ee931a8331ccfb8292bee6d3e5c03edf5e1/routing.go#L273

S/Kademlia

- Secured Kademlia
- Node lookups over disjoint paths
- K closest nodes, d independent lookup bucket, parallel independe d lookups, each node is used only once

S/Kademlia

- requires nodes to create a PKI key pair, derive their identity from it, and sign their messages to each other.
- One scheme includes a proof-of-work crypto puzzle to make generating Sybills expensive.

Implementation

- Bittorrent for trackerless torrents (magnet links)
- IPFS
- Tox – A fully distributed messaging, VoIP and video chat platform
- I2P- Invisible Internet Protocol, an anonymous network layer that allows for censorship-resistant, peer to peer communication

References

- Software Daily Podcast for Kademlia https://www.youtube.com/watch?v=3Y6vLTzM7zA
- XOR distance and Basic Routing https://www.youtube.com/watch?v=w9UObz8o8lY
- https://en.bitcoinwiki.org/wiki/Kademlia
- S/Kademlia https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.68.4986&rep=rep1&type=pdf
- WebTorrent talk https://www.youtube.com/watch?v=kxHRATfvnlw
- Kademlia Paper https://pdos.csail.mit.edu/~petar/papers/maymounkov-kademlia-lncs.pdf

Reference

- https://www.cs.princeton.edu/courses/archive/spr11/cos461/docs/lec22-dhts.pdf
- https://computer.howstuffworks.com/dns.htm