Peer to Peer - Appunti

Francesco Lorenzoni

February 2024

Contents

Course info

...

Chapter 1

Introduction

Opposed to Client-server architectures where there are end-hosts and dedicated-hosts (servers), in P2P Systems there only end-nodes which directly communicate with each other; they have an "on/off" behaviour, and they handle **churn**[1]. However, in P2P systems servers are still needed, but only as *bootstrap servers*, typically allowing for new nodes to easily join the P2P network.

Peers' connection in P2P is called *transient*, meaning that connections and disconnections to the network are very frequent.
Notice that since each time a peers connects to the P2P network it may have a different IP address, resources cannot be located using IP, but a different method at application layer must be used.

Definition 1.1 (P2P System) *A peer to peer system is a set of autonomous entities (peers) able to auto-organize and sharing a set of distributed resources in a computer network.*
The system exploits such resources to give a service in a complete or partial decentralized way

Definition 1.2 (P2P System - Alternative definition) *A P2P system is a distributed system defined by a set on nodes interconnected able to auto-organize and to build different topologies with the goal of sharing resources like CPU cycles, memory, bandwidth. The system is able to adapt to a continous churn of the nodes maintaining connectivity and reasonable performances without a centralized entity (like a server)*

1.1 Blockchain concepts

Definition 1.3 (Blockchain)
- ◇ *a write-only, decentralized, state machine that is maintained by untrusted actors, secured by economic incentive*
- ◇ *cannot delete data*
- ◇ *cannot be shut down or censored*
- ◇ *supports defined operations agreed upon by participants*
- ◇ *participants may not know each other (public)*
- ◇ *in actors best interest is to play by the rules*

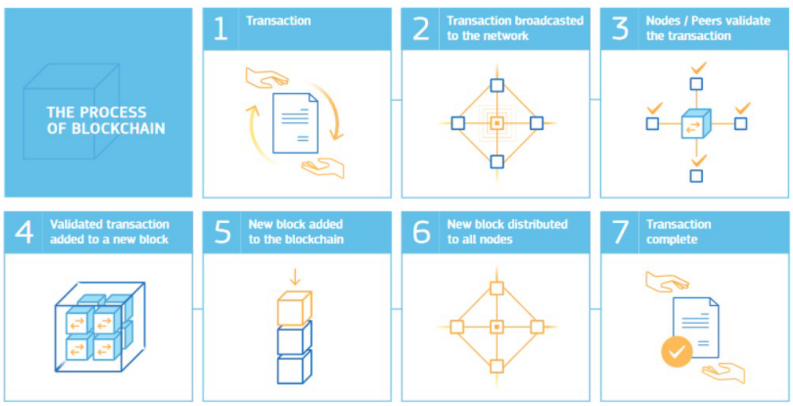


Figure 1.1: Blockchain process

[1] *"churn"* will be a recurring term. In italian it means "rimescolare"

Bitcoin were developed as an alternative way to exchange money which wouldn't need intermediaries such as banks. Today, *Ethereum* is becoming more and more popular.
NFT[2] allow to establish the owner of a digital artwork, by generating a token using a blockchain.

1.1.1 TriLemma

The Blockchain **trilemma** states that a blockchain **cannot** simultaneously provide *Decentralization*, *Security* and *Scalability*.

1.2 P2P Systems

1.2.1 Semi-Decentralized systems

An example is **Napster**, released in 2001. Napster used servers only to allow users to locate peers which could provide the desired file, delegating the actual file exchange to peers, allowing for a very few server needed.

For the first time users are called *peers*, and the systems implemented in this way *peer-to-peer systems*

Napster had many strengths common to many P2P systems, from whose emerges the ability of peers to act both as server and a client, but also suffered from weaknesses derived from its centralization, at least for "node discovery". Napster centralized server represents a design bottleneck, and also made it target of legal attacks.

1.2.2 Fully decentralized systems

Gnutella is similar to Napster, but here no centralized server exists. Peers establish *non-transient* direct connections to search files, not to actually transfer them.

Cons
1. High network traffic
2. No structured search
3. Free-riding

1.3 P2P Overlay network

In P2P systems there is an overlay network at application level operating on top of the underlying (IP) network.

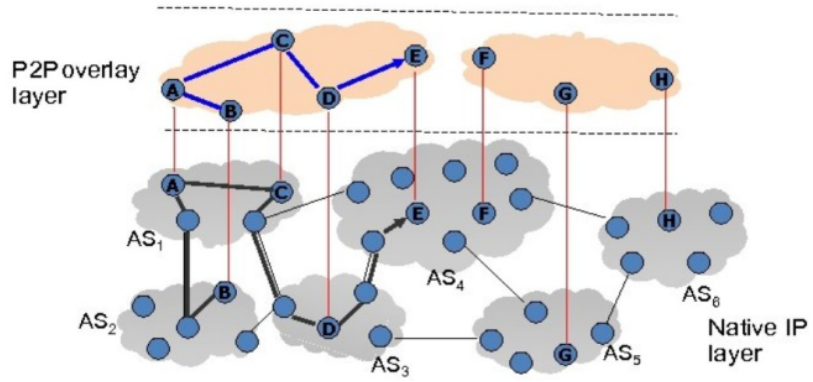


Figure 1.2: P2P Overlay networks

A P2P **protocol** —defined over the P2P overlay— defines the set of messages that the peers exchange.

1.3.1 Unstructured overlay

[2]Non Fungible Tokens

The two key issues here are:

- how to **bootstrap** on the network?
- how to **find content** without a central index?

Possible lookup algorithms are the following, but they all are not very scalable, and are costful in terms of performance:

- **Flooding**
- **Expanding ring**
- **Random walk**

`Gnutella`, `BitTorrent`, `BitCoin`

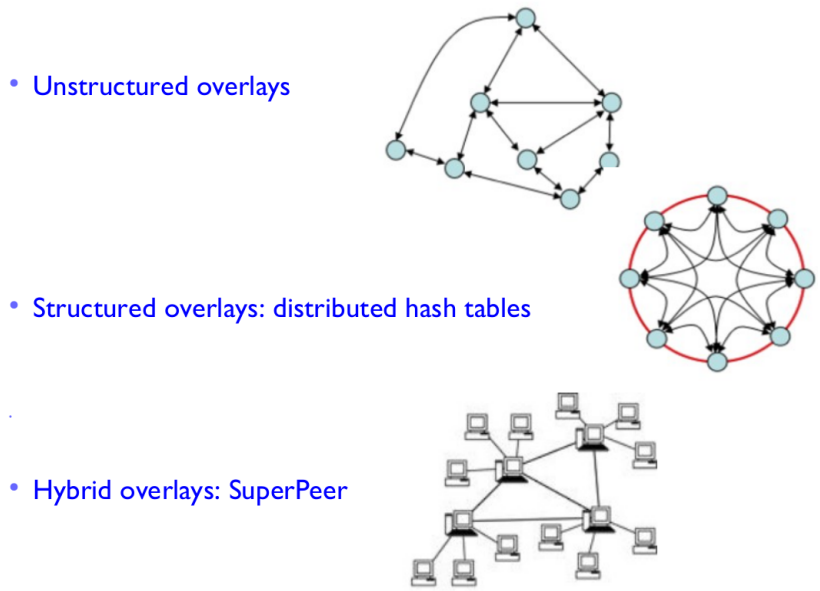


Figure 1.3: P2P Overlay Network classification

Flooding

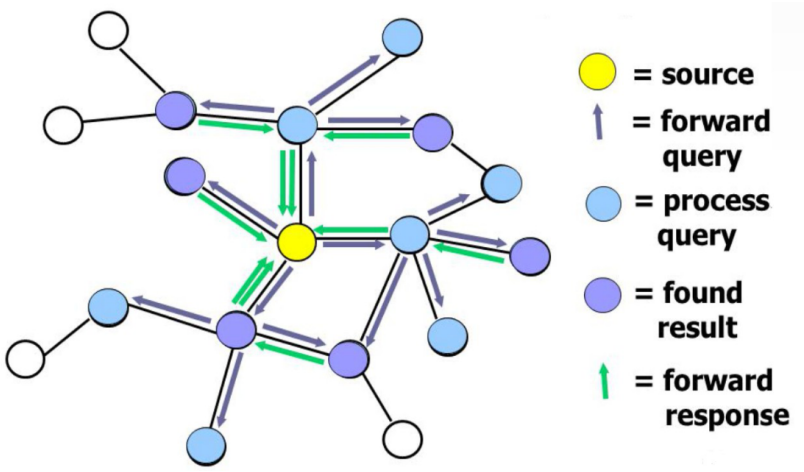


Figure 1.4: Flooding search in unstructured overlay
Messages have a *TTL* to limit the number of hops when propagating, but also a *unique identifier* to detect cycles.
Flooding is not only for searching, but also to propagate transactions in the P2P network underlying a **blockchain**

Expanding Ring

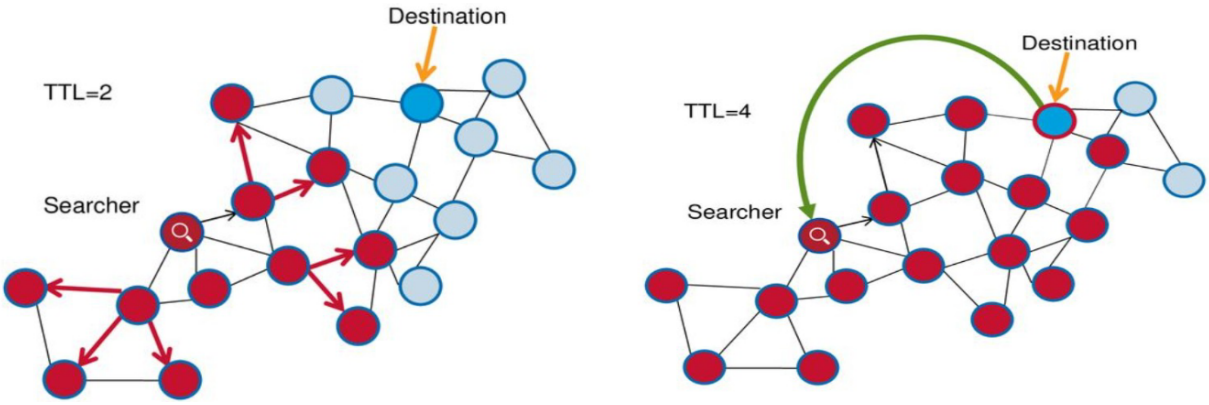


Figure 1.5: Expanding ring/Iterative Deepening
This tecnique consists in repeated flooding with an increasing TTL, implementing a BFS search.

Random walk

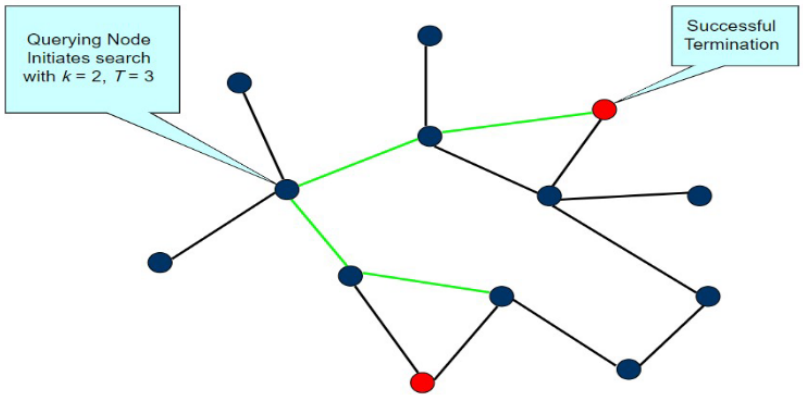


Figure 1.6: Random Walk
k indicates the number of "walkers" to be generated by the querying node. Each green path corresponds to a "walker".
A path is constructed by taking successive (single) steps in random directions defined by a *Markov Chain*, which is "memory-less". Note that only one successor node is chosen at each step.
The path is bounded by a TTL.

Random walk avoids and exponential increase in the amount of messages, which becomes an issue for vastly populated networks.
Paths can be stopped by a TTL but also by checking periodically with the destination wether the stop condition has been met.
A querying node can also bias its walks towards high-degree nodes: higher probability to choose the highest degree neighbor.

1.3.2 Structured overlays

The choice of the neighbours is defined according to a given criteria, resulting in a **structured** overlay network.
The goal is to guarantee scalability by providing:

- ◇ *key-based lookup*
- ◇ information lookup has a given *complexity* e.g. $\mathcal{O}(logN)$

1.3.3 Hierarchical overlays

Peers connect to **Super-Peers** which know (*"index"*) Peer resources. The flooding is restricted to Super-Peers, but still allowing for resources to be directly exchanged between the peers.

Lookup complexity is less and the scalability is improved, but there is a lower resistance to *super-peers* churn.

In some cases —such as Gnutella— peers are "*self-promoted*" to super-peers, while in others they are statically defined.

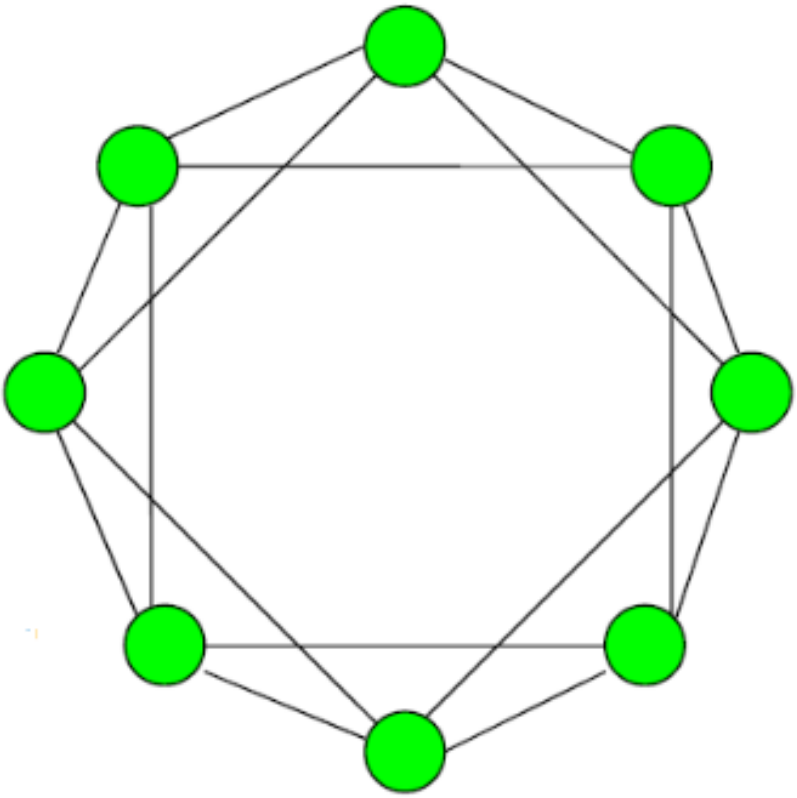


Figure 1.7: Structured overleay

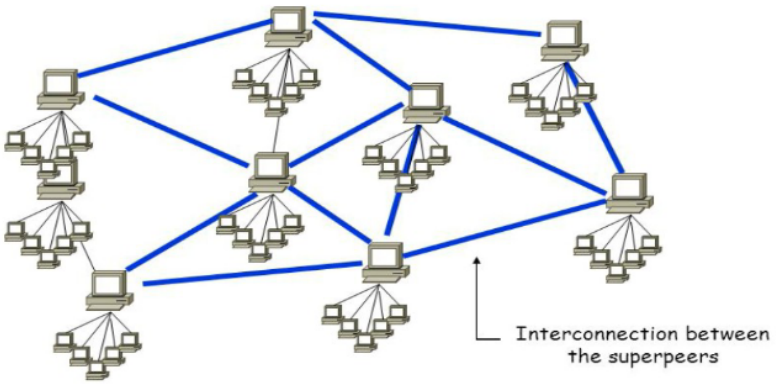


Figure 1.8: Hierarchical overlay

1.3.4 Summary

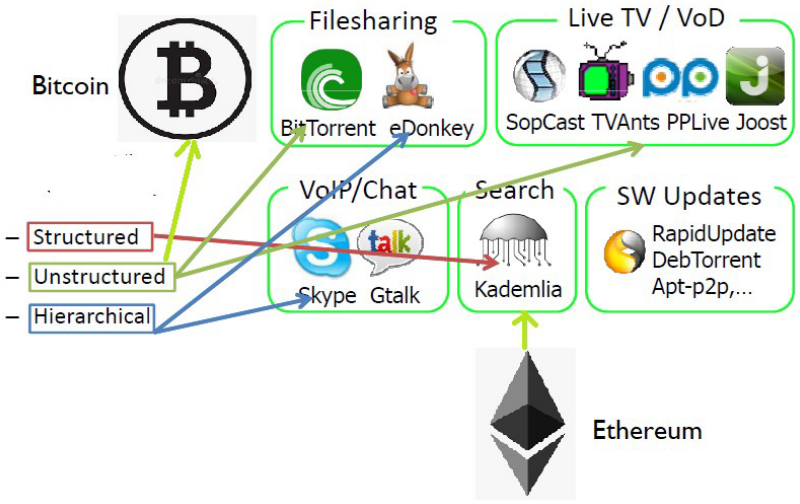


Figure 1.9: Overlay structure for known applications

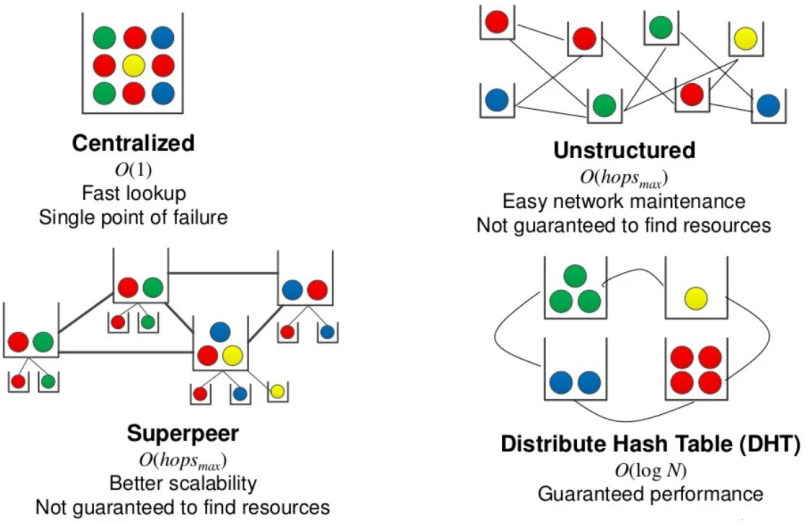


Figure 1.10: Overlays summary
DHT will be discussed in the next chapter

Chapter 2

Distributed Hash Tables

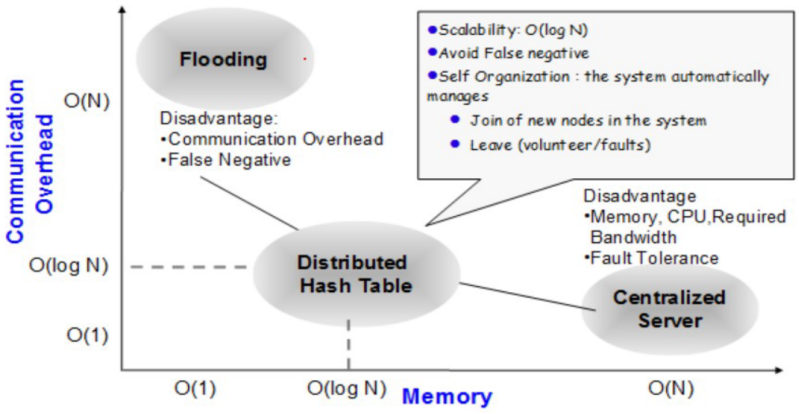


Figure 2.1: DHT Motivations

The key idea is to split the hash tables into several parts and distribute them to several servers, and to use hash of resources (or of the URLs of resources) as a key to map them to a dynamically changing set of web caches, but with each key mapped to single server; so that each machine (user) can locally compute which web cache should contain the required resource, refenced by an URL.
This technique is extended to DHT for P2P systems.

However, **rehashing** is a problem in dynamic scenarios if the hashing scheme depends directly on the number of servers: 99% of keys have to be remapped, resulting in a lot of messages exchange.

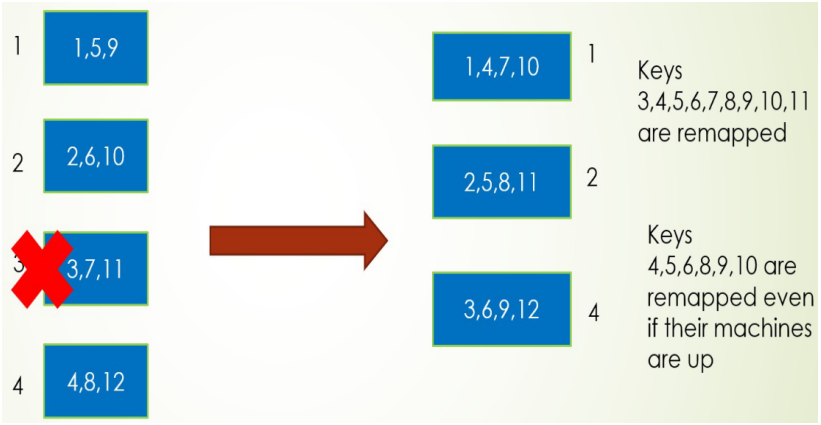


Figure 2.2: Rehashing problem

Consistent hashing is a set of hash techniques which guarantees that adding more nodes/remove nodes implies moving only a minority of data items. each node manages —instead of a set of sparse keys— an interval of consecutive hash keys, and intervals are joined/splitted when nodes join/leave the network and keys redistributed between adjacent peers.

2.1 Building DHT

- ◇ Use a logical name space, called *identifier space* consisting of identifiers $\{0, 1, 2, ..., N-1\}$
- ◇ define identifier space as a *logical ring* modulo N
- ◇ every node picks a random identifier through Hash H.

space N=16 {0,…,15}

- five nodes a, b, c, d, e
- H(a) = 6
- H(b) = 5
- H(c) = 0
- H(d) = 11
- H(e) = 2

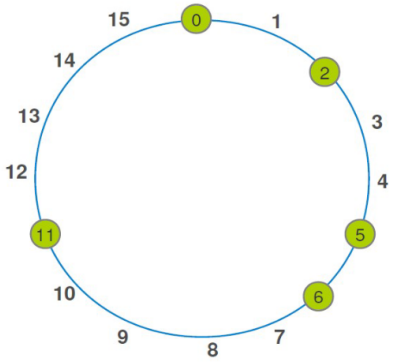


Figure 2.3: Identifier space

2.1.1 Peers joining and leaving

When a new node is **added**, we map the keys between the new node and the previous node in the hash ring to point to the new node;
those the keys will no longer be associated with their old nodes.

When a node is **removed** from the hash ring, only the keys associated with that node are rehashed and remapped rather than remapping all the keys.

In case a node suddenly disconnects from the network, all data stored on it are lost if they are not stored on other nodes; to avoid such a problem:

- ◇ introduce some redundancy (data replication)
- ◇ information loss: periodical information refresh

- the "crown" now is mapped to peer 0
- the absence of peer 11 does not affect keys belonging to other servers

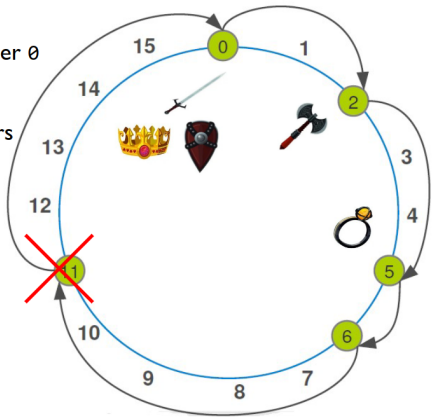


Figure 2.3: Peer 11 leaves the Network
In case a peer leaves, its keys can easily be remapped to its successor

When the hash table is **resized**, on the average,only $\frac{k}{n}$ keys need to be remapped on average, where k is the number of keys and n is the number of servers.

2.2 Data Lookup

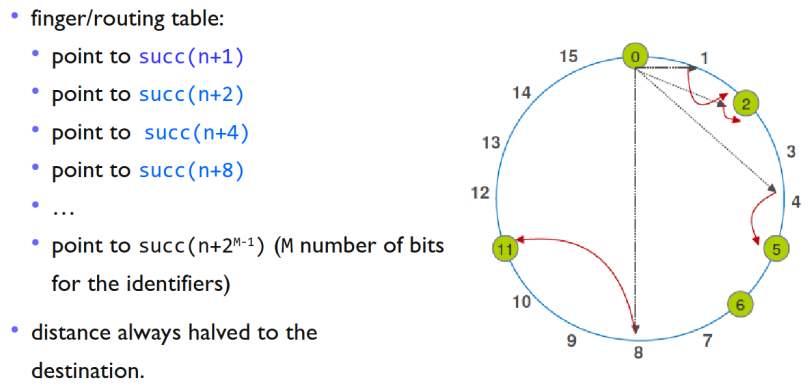


Figure 2.4: Exponential Search for DHT
The data lookup can be implemented by using exponential search, rather than performing a walk by asking each peer for its successor

Data Lookup can be sped up even more, by computing the hash $h(x)$ of the searched object, and propagating the query to farthest node[1] which has an identifier smaller than $h(x)$, which then recursively applies the same algorithm, until the object is found.

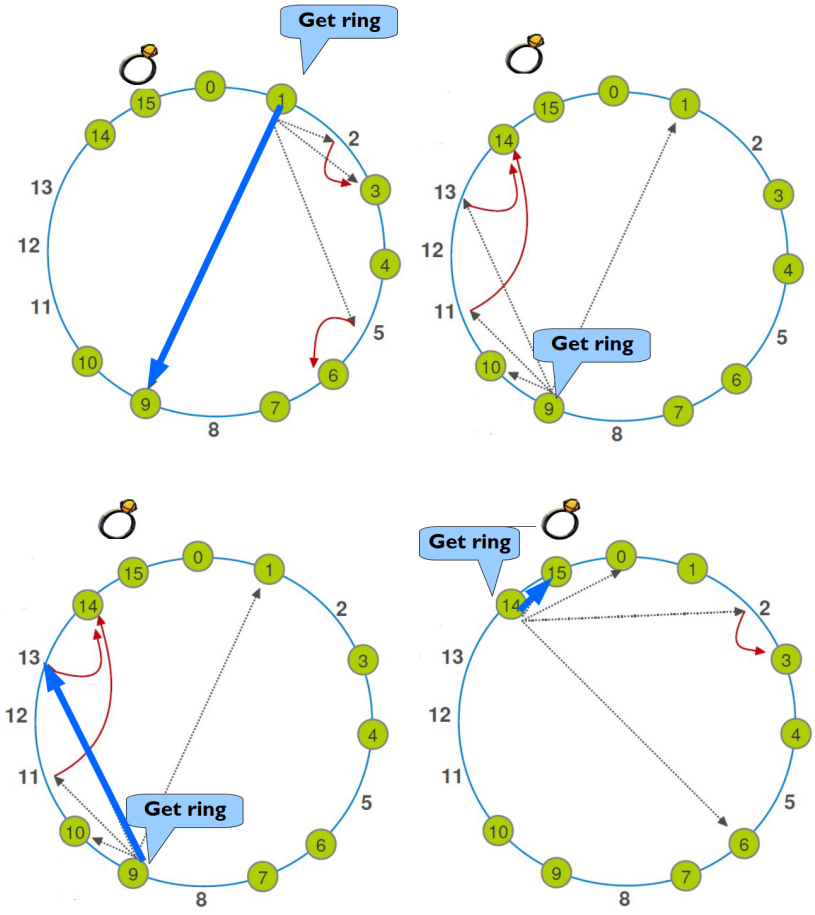


Figure 2.5: Lookup performed in the CHORD DHT

2.2.1 Addressing data

Data was usually addressed by **location**, a `http://` link to locate resources; Such link is an identifier that points to a particular location on the web.
This approach forces us all to pretend that the data are in only one location.

[1] Which is found using exponential search

IPFS instead uses **content addressing**, which exploits the cryptographic hash of the content to identify it.

2.2.2 API, Lookup and Various Properties

Most DHT provide a simple inferface `PUT,GET,Value`, usually without the possibility to move keys.

Approach	Memory for each node	Communication Overhead	Complex Queries	False Negatives	Robustness
Central Server	O(N)	O(1)	✓	✓	✗
Pure P2P (flooding)	O(1)	$O(N^2)$	✓	✗	✓
DHT	O(log N)	O(log N)	✗	✓	✓

Figure 2.6: Lookup time complexity comparison

DHT

- ◇ Routing is based on key (unique identifier)
- ◇ Key are uniformly distributed to the DHT nodes
  1. Bottleneck avoidance
  2. Incremental insertion of the keys
  3. Fault tolerance
- ◇ Auto organizing system
- ◇ Simplex and efficient organization
- ◇ The terms “Structured Peer-to-Peer“ and “DHT“ are often used as synonyms

Chapter 3

Kademlia

Kademlia is a protocol used by some of the largest public DHTs
- ◇ BitTorrent Mainline DHT
- ◇ Ethereum P2P network
- ◇ IPFS

It has three key charateristics which are not offered by other DHTs

1. routing information spreads automatically as a side-effect of lookups
2. flexibility to send multiple requests in parallel to speed up lookups by avoiding timeout delays (parallel routing)
3. iterative routing
   At each routing step of the query, the queried node sends a report to the starting querying node, even if it could not answer the query.

3.1 Structure

Kademlia exploits the leaves of a **Trie**[1] to define the logical identifier space;

Note that not all leaves correspond to nodes (peers)

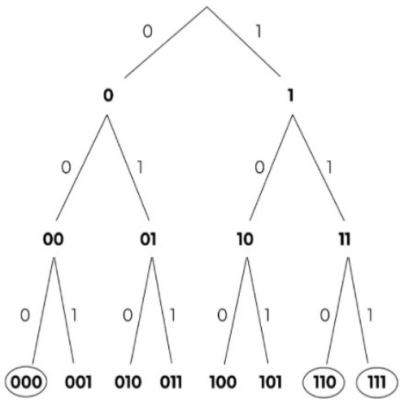


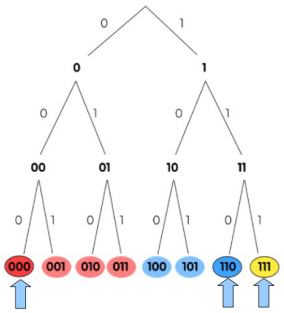
Figure 3.1: Trie

[1] k-ary search tree and prefix tree

3.1.1 Assigning keys to leaves

The rule to partition the keys (content) among the nodes must respect the rules of *consistent hashing*.

Definition 3.1 (Partitioning rule) *A key is assigned to the node with the "lowest common ancestor": Find the longest prefix between the key and the node identifier, and then assign the key to such node.*



- in the figure
  - leaves pointed by arrows correspond to nodes: red, blue and yellow leaves
  - pale coloured, not circled leaves, correspond to data key
- pale blue keys assigned to blue node
- pale red keys assigned to red node, no key assigned to the yellow node

3.2 XOR Metric

TODO

3.3 Routing Table

In order to look for data, Kademlia's key idea is to store a logarithmic number of node IDs and their corresponding IP addresses and some contact taken from the identifier trie.