TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

TU Dresden // Professur für Rechnernetze
INF-PM-ANW Distributed Systems

Distributed Hash Tables

Philipp Matthes
E-Mail: philipp.matthes@tu-dresden.de
Website: https://philippmatth.es

Contents

1. Basics and motivation
   Why do we need Distributed Hash Tables?

2. Functional aspects of Distributed Hash Tables
   How do they work?

3. Comparison between different concrete approaches
   Which different approaches to Distributed Hash Tables do exist?

4. Illustration of a selected Distributed Hash Table in use
   Research project “Peerbridge” Blockchain Messenger

Basics and motivation

Why do we need Distributed Hash Tables?

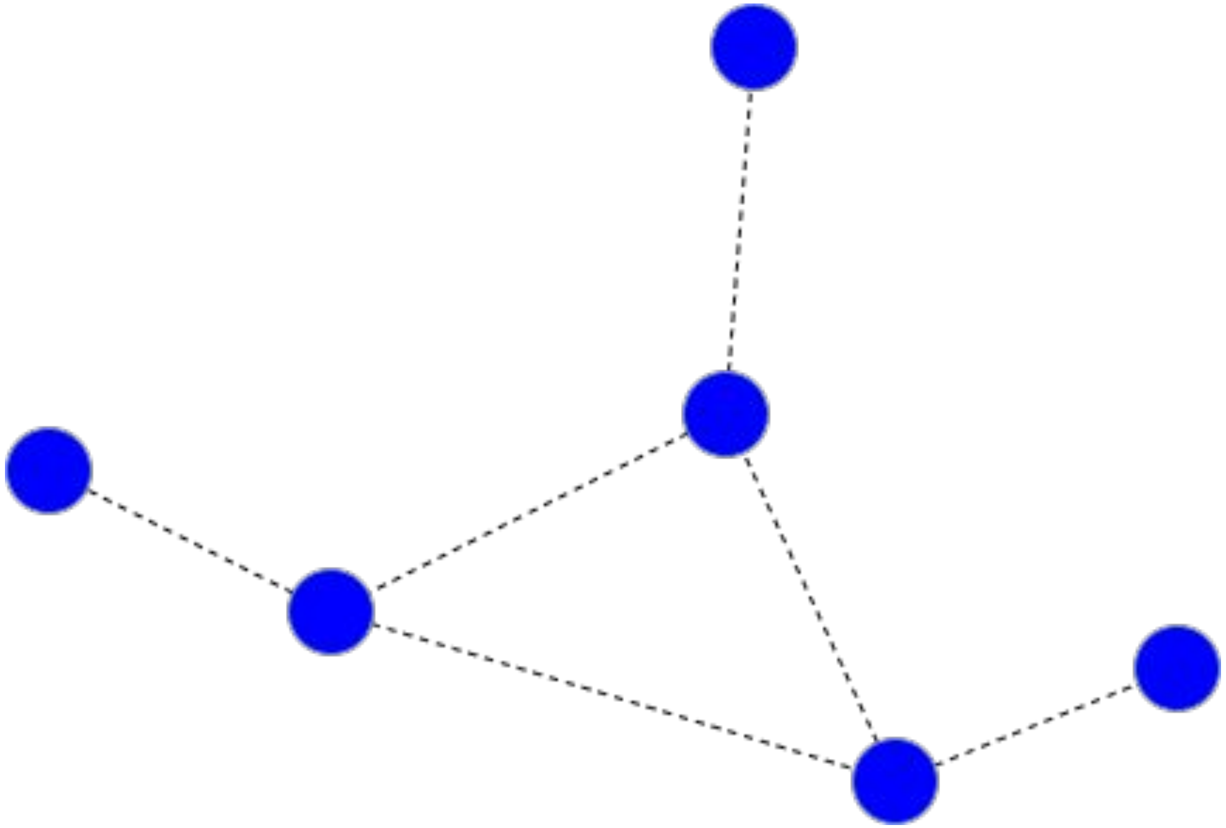
Peer-to-Peer Networks

- Is a decentralized network structure similar to a mesh
- Nodes connect directly to each other
- No central server needed

Advantages

- Scalability is usually very high
- No centralized control (every client has the same rights)
- Adaptivity and Flexibility, ...

Used by many: Filesharing Systems (e.g. BitTorrent), Jitsi (with 2 participants), many more



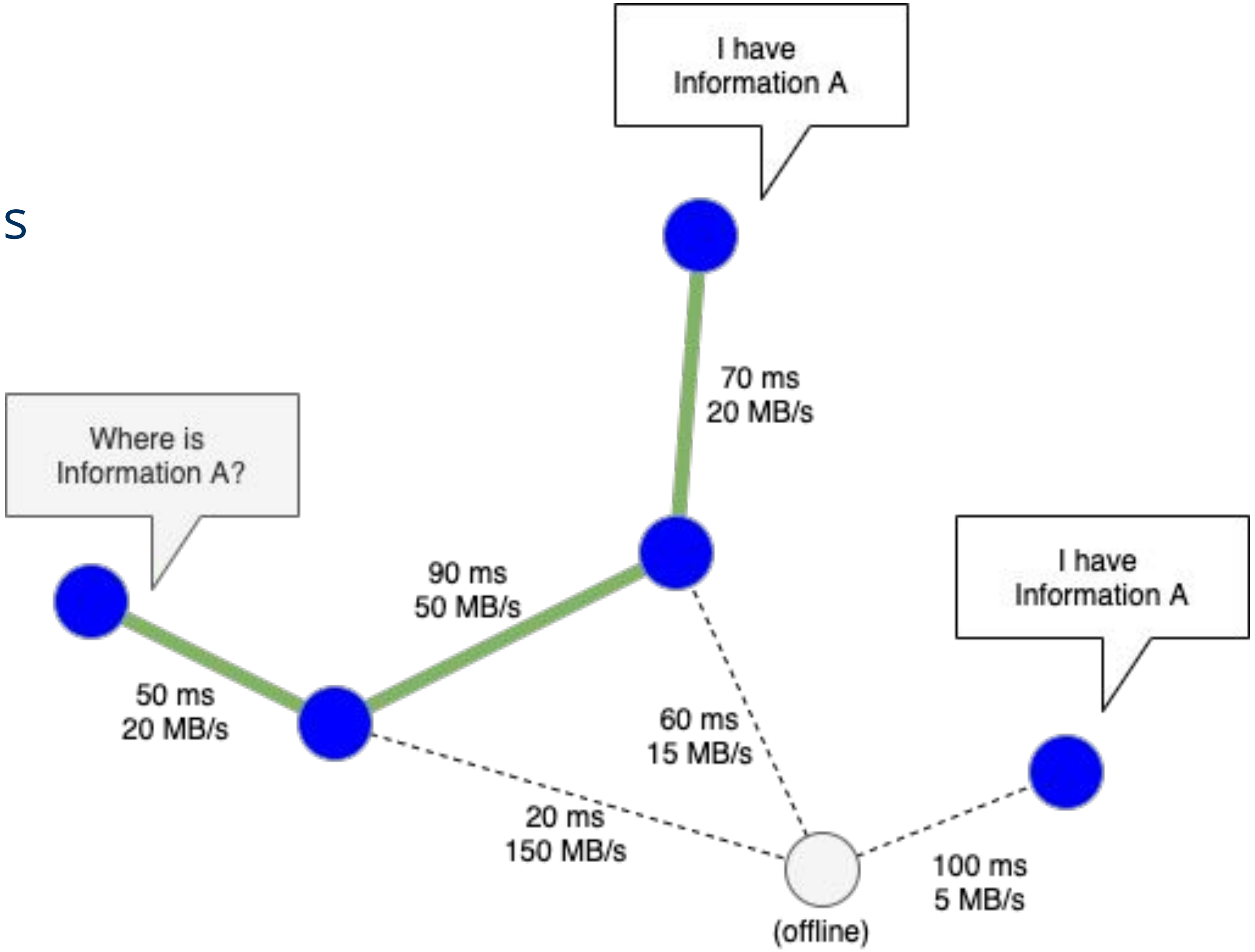
Challenges of Peer-to-Peer Networks

Trust Problems

- Nodes may distribute malicious data
- Nodes may spy on network traffic
- Nodes may violate distributed protocols

Routing and Connection Problems

- Nodes can go **online** and **offline** spontaneously
- Nodes need to **locate** the requested information (ideally without asking everyone)
- Nodes need to be able to create a **physical connection** to receive the information



Note: this is an incomplete selection of problems

Solutions to the Challenges of Peer-to-Peer Networks

Trust Problems
- Nodes may distribute malicious data
- Nodes may spy on network traffic
- Nodes may violate distributed protocols

Trust Solutions
Cryptographic encryption and signatures,
Onion-Principle,
Blockchain and Consensus-Protocols

Routing and Connection Problems
- Nodes can go **online** and **offline** spontaneously
- Nodes need to **locate** the requested information (ideally without asking everyone)
- Nodes need to be able to create a **physical connection** to receive the information

Routing and Connection Solutions
Centralized Lookup Service,
Overlay-Networks together with
Distributed Hash Tables

Note: this is an incomplete selection of problems

Distributed Hash Tables

Definition: A Distributed Hash Table (short DHT) is a data structure to locate and distribute information (such as files) in peer-to-peer networks.

Dissect this definition for a better understanding:

- It is a data structure on each node, which stores hash values (How?)
- The hash values relate to the stored information (How?)
- Via the hash values, the information can be located (How?)
- This solves the previously shown routing and connection problems (How?)

Functional aspects of Distributed Hash Tables
How do they work?

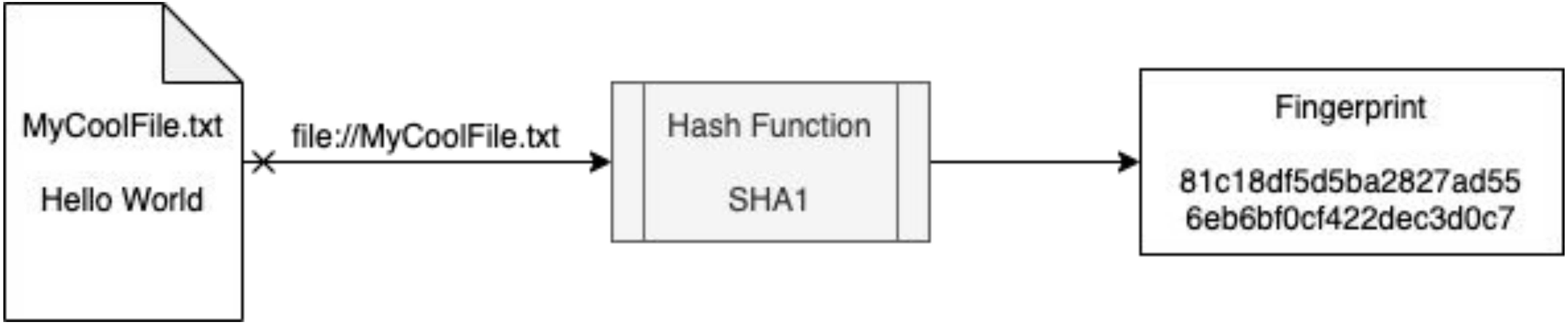
The Basic Algorithm behind DHTs

Information Upload:
Take File A, which needs to be uploaded into the P2P network.

1. The DHT creates a fingerprint of File A using a **hash function on the file's URI**
2. In the DHT, the fingerprint responsibilities are evenly partitioned between all nodes using **keyspace partitioning**
3. The DHT maps the generated fingerprint to the responsible nodes
4. The DHT keeps track of all nodes in the network using an **overlay network**
5. Get a physical route to the responsible nodes using the **overlay network** and upload

Information Download:
Ask the DHT for the responsible nodes (using the file's URI), connect to the nodes (the same way as using the upload) and download

1. The DHT creates a fingerprint of File A using a **hash function on the file's URI**

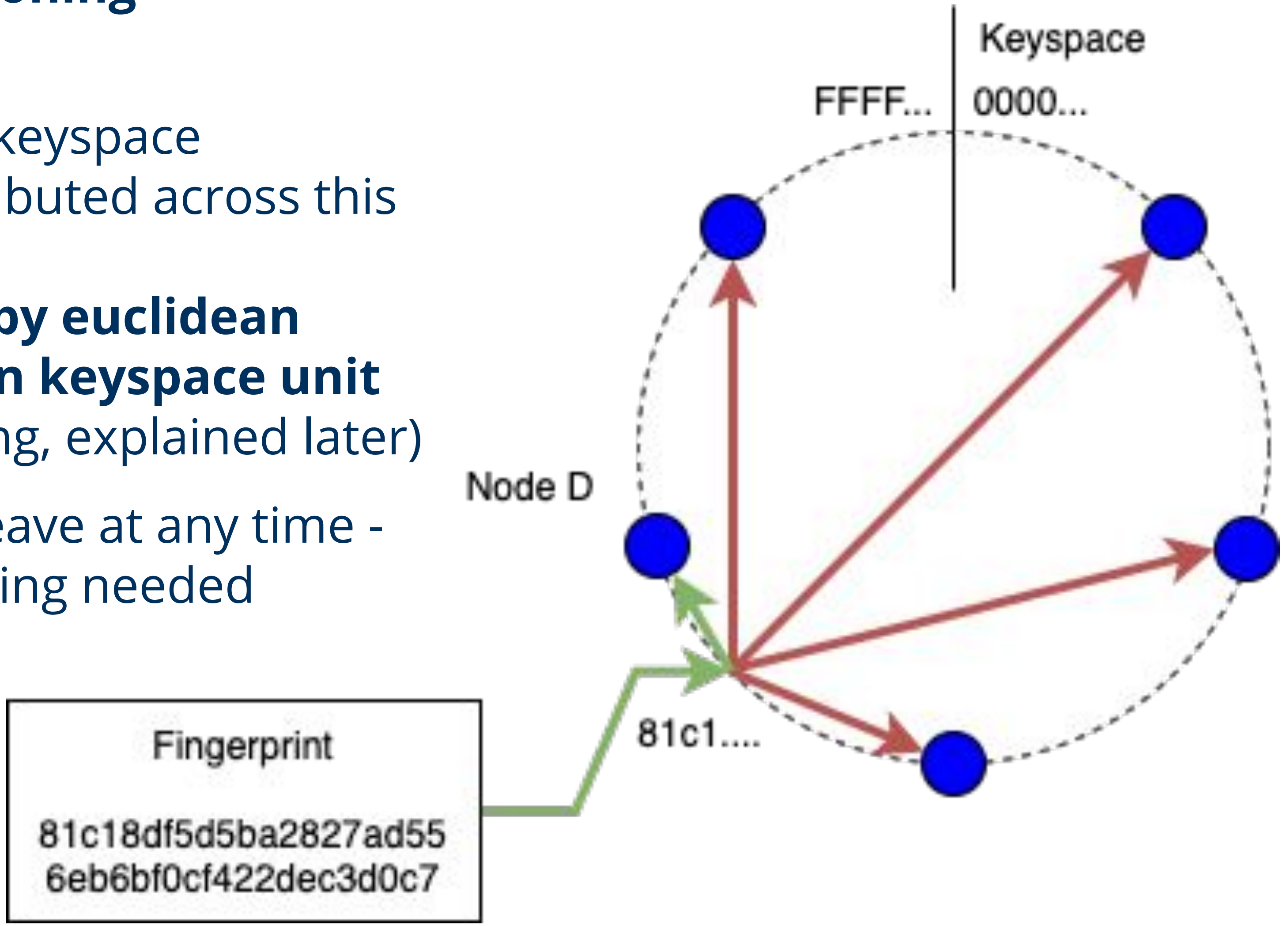


- Hash functions are cryptographic one-way functions, which means it is practically impossible to generate the original input from the hash
- **Advantages**:
  - The hash is always of the same length and does not disclose the original URI, therefore providing more anonymity
  - (Important for DHTs) We can apply so called **keyspace partitioning**

2. In the DHT, the fingerprint responsibilities are evenly partitioned between all nodes using **keyspace partitioning**

- The fingerprint maps to a keyspace
- (3.) Nodes are evenly distributed across this keyspace
- **Select responsible node by euclidean distance to fingerprint on keyspace unit circle** (in consistent hashing, explained later)

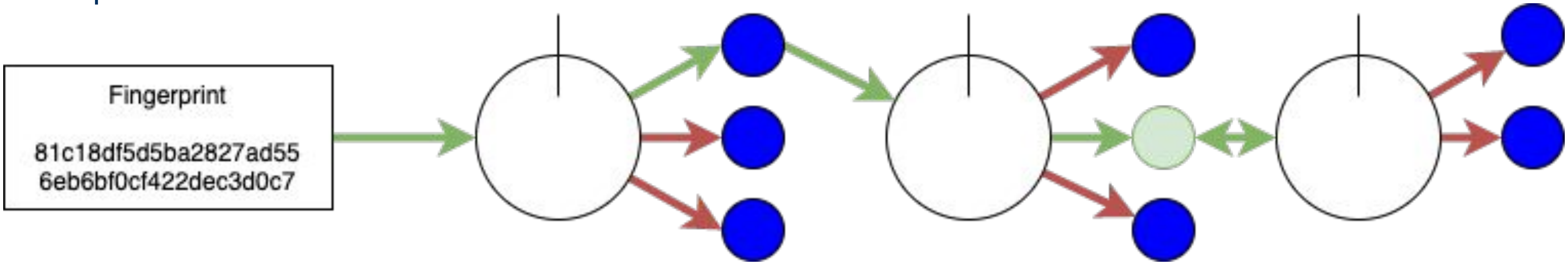
Note: Nodes may join or leave at any time - algorithms for joining/leaving needed



4. The DHT keeps track of all nodes in the network using an **overlay network**

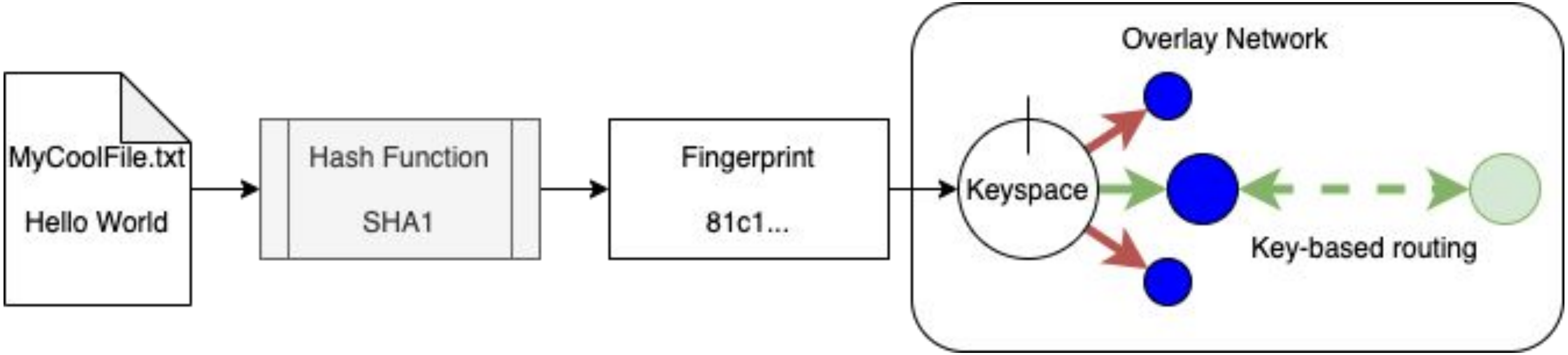
- Each node holds a set of links to other nodes (its nearest neighbors)
- Handles connection (+continuous joining and leaving) in the P2P network via UDP
- Links contain the node's key and its physical network address
- Use **key-based routing** to find the responsible node
- Key-based routing: use the key distance to traverse the network (coming closer to the responsible node)

5. Get a physical route to the responsible nodes using the **overlay network** and upload



Three important parts

1. **A hashing algorithm** to distribute hashes evenly across the keyspace
2. **A keyspace partitioning algorithm** to evenly map the keyspace to nodes
3. **An overlay network** to route to physical nodes



Comparison between different concrete approaches

Which different approaches to Distributed Hash Tables do exist?

Variations in the hashing function (1/2)

Remember: the hashing function maps nodes and information evenly to a keyspace (using **keyspace partitioning**)

What happens if a node leaves/joins the network?

- Consequence: If the nodes change, they should reorganize along the keyspace to keep approximately equal responsibilities
- Problem: How to avoid that every node needs to reassign a new slot?

Solution: use a special kind of hashing function!

Variations in the hashing function (2/2)

A brief overview

Rendezvous Hashing (Special Case: Consistent Hashing)	**Locality-preserving Hashing**
Distribute objects evenly (using randomized Hash function) across nodes by assigning (distance) scores: On disconnect of a node, reassign the nearby keys to the nearest nodes. Consequence: Only a small portion of the keys needs to be reassigned! Consistent Hashing: use euclidean distance on the unit circle (as shown earlier) as distance metric	Use a hashing algorithm, that produces similar hashes for similar data: Then, assign similar objects to similar nodes. Can be more efficient under certain circumstances, but due to the missing randomization: even distribution of objects across sites is serious challenge!

Note: Key distance != Geographic distance - Nodes may be geographically separated!

Variations in the overlay network (1/2)

Remember: **overlay networks** span a virtual network over a physical network to keep track of nodes and their physical addresses - important for P2P communication!

- Each node has its own routing table - for DHTs, the routing is **key-based** (either a node knows the responsible node, or forwards to the least distant node)
- There are two important metrics about the overlay network's topology:

  In comparison to the number of nodes *n*...

  - **Maximum route length**: how long is the route through the network in the worst case? - affects network speed and latency
  - **Maximum degree**: how many nodes are connected to each node in the worst case? - affects lookup overhead

Variations in the overlay network (2/2)
A brief overview

Max. Degree	**Max. Route Length**	**Example overlay networks**
O(log n)	*O(log n)*	Chord **Kademlia** Pastry Tapestry
O(log n)	*O(log n / log(log n))*	Koorde

How does an overlay network work exactly?

An example for overlay networks: Kademlia

Distance metric: **bitwise *XOR* between two fingerprints as unsigned integer**

Good topological properties for routing:

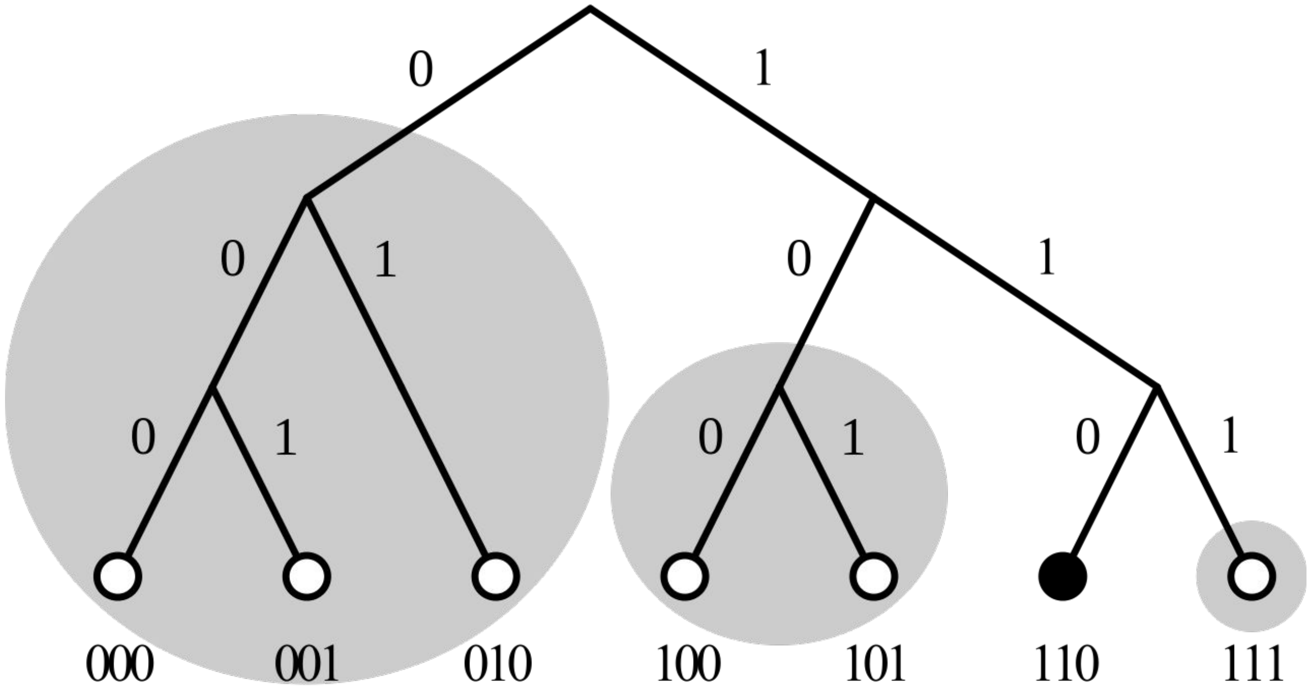
- The distance between a node and itself is 0
- *XOR* is symmetric: distances are the same in both directions
- Follows the triangle inequality (direct distance between two node is shorter than the distance over an intermediate node)

Good performance properties:

- Bitwise *XOR* is very easy and cheap to compute
- Basic logic operation in every processor

Routing in Kademlia in a Nutshell

According to their *XOR* distance, nodes are stored into a tree structure on encounter. Kademlia orders them into **k-buckets**, containing at maximum k nodes.



By Limaner - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=33298754

How to find node **011** from node **110**:

1. Traverse the tree structure and lookup the nearest node (is **010**)
2. Ask the node, if it knows another node which is closer to **011**
3. **010** returns: **011** is nearby and returns physical address

Finding an information works in the same way, using the key distance as explained earlier!

Routing in Kademlia in Reality

Many technical and logical challenges:

- Practicable key spaces have many more possible keys (e.g. 2^{128}): nearby k-buckets may be empty (containing no known nodes)
- K-buckets may also be full, when a new node is discovered: ping least recently seen node (to see if it is still alive) and potentially fallback to a replacement cache
- Orchestrate all functionalities on a small set of protocol messages:
  - **PING** to check if a node is still alive
  - **STORE** to save new data into a node using a key
  - **FIND_NODE** to lookup a node closest to a key
  - **FIND_VALUE** to lookup data to a given key

Joining and Leaving the Network in Kademlia

New nodes undergo a process called **bootstrapping**:

1. Know another participating node in the Kademlia network, called **bootstrap node** in this context
2. Compute a random unique id, which will be recognized by other nodes
3. Insert the bootstrap node into the (previously empty) k-buckets
4. As a last step, perform a “self-lookup” to populate other nodes’ k-buckets:
   a. Use the FIND_NODE request to lookup the previously generated id by contacting the bootstrap node
   b. The other nodes will react to this by looking up the new node and populating their own k-buckets with the new node

Passively leaving the network:
not responding and being replaced by other close nodes

Examples for Implementations of Kademlia

Famous implementations:

- **BitTorrent** (uses Kademlia to provide torrents)
- **I2P** (an anonymous overlay network layer)
- **Ethereum** (blockchain that uses Kademlia to discover new nodes)

Other implementations:

- Research project “**Peerbridge**” Blockchain Messenger

Illustration of a selected Distributed Hash Table in use

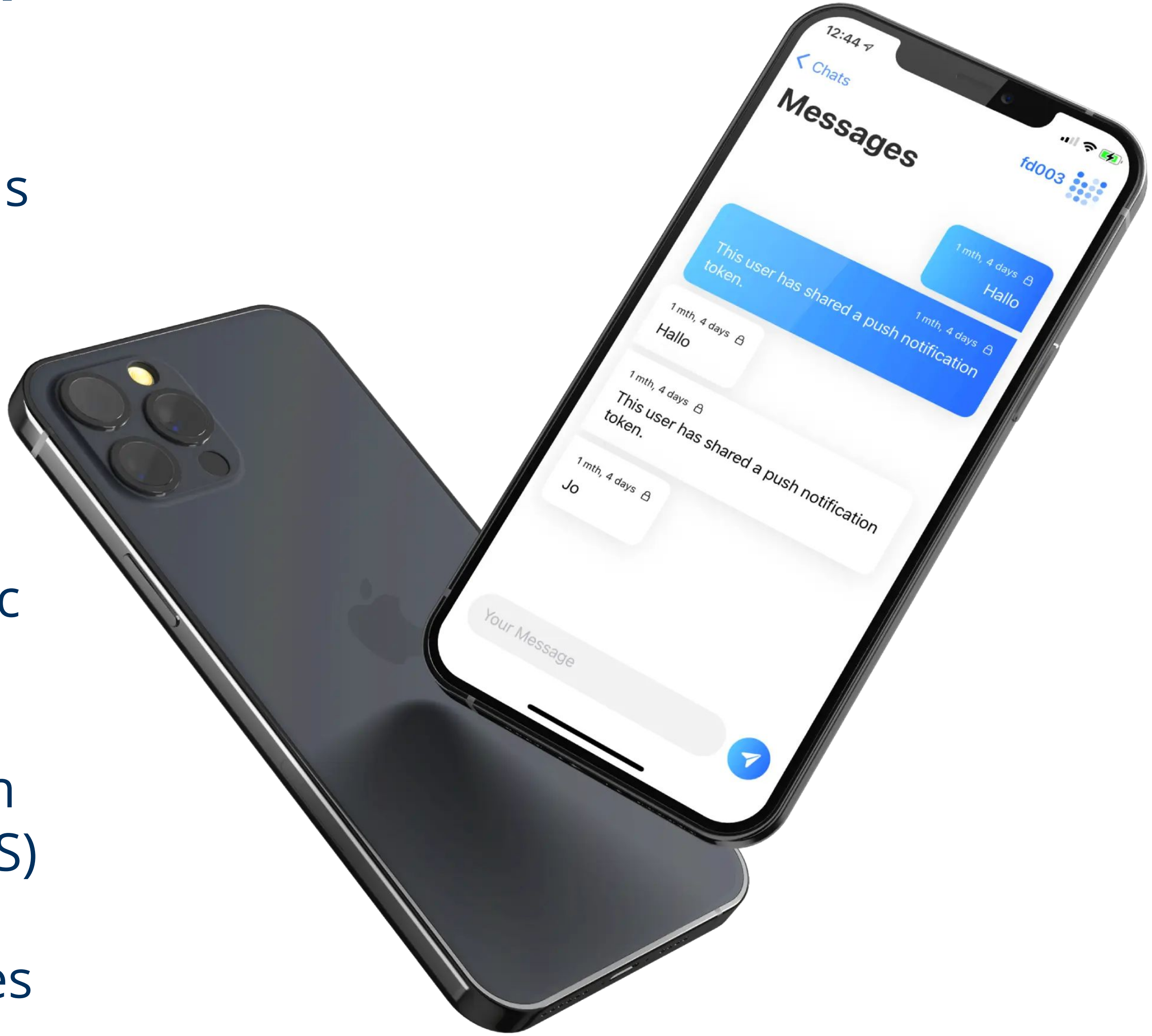
Research project “Peerbridge” Blockchain Messenger

Peerbridge Blockchain Messenger

... is a research project to explore more possibilities of providing fully anonymous messaging.

Approach:

- Messages are encrypted, signed and stored publicly within a distributed blockchain
- Users and nodes identify using public keys
- Use push notifications to trigger message fetches from the blockchain
- Use a Delegated Proof of Stake (DPoS) to provide consensus about the blockchain's contents across all nodes



Blockchains are P2P systems

There are two types of nodes:

- Those which store the blockchain and provide the messaging APIs
  In Peerbridge these are based on a simple Go server
- Those which send and receive messages (paying a fee, called “gas price”)
  In Peerbridge these are the client applications for iOS and Android devices

The communication works as follows:

1. Users of the iOS and Android applications send encr. data to a blockchain node
2. The blockchain node communicates this data with the other blockchain nodes
3. The blockchain nodes write this data into a new block of the blockchain
4. After agreeing on consensus (using DPoS) one of the blockchain nodes forges the data into the blockchain (persisting it forever)
5. The recipient pulls this data from the nearest available blockchain node

Note: Users may opt-in to send their push notification token for a push-initiated message fetch.

Peerbridge uses Kademlia to discover new Nodes

New clients and new nodes should be able to join the P2P network without a centralized lookup service.

Solution: use a DHT in which the nodes' public keys are applied together with Kademlia to find new nodes.

Remember: Kademlia requires a bootstrap node to work properly.

- On servers, this bootstrap node can be passed as a part of configuration
- On mobile applications, we cannot apply this paradigm

How do we provide this bootstrap node on mobile applications?

Providing a Bootstrap Node using QR Codes

Use invitation-based joining:

- When a user wants to start a new chat, he/her shows the other user a qr code
- The qr code contains the following information:
  - The recipient’s public key used to encrypt messages and to identify him/her in the blockchain
  - One or multiple known nearby (or random) blockchain nodes with their physical addresses
- Use the received blockchain nodes to bootstrap Kademlia
- Note: This requires a seed application, which is preconfigured with the initial blockchain node(s)

Conclusion: **users can use the blockchain without the need of an initial configuration**

Conclusion
Summing up the most important points

Conclusion

- **Distributed Hash Tables can be used to solve routing and localization challenges in P2P networks.**
- They do this by providing an ability to lookup physical addresses and routing pathways for given resource URLs.
- They consist of three major parts: A hashing algorithm, keyspace partitioning and an overlay network.
- Hashing algorithms have different properties, which affect the keyspace partitioning.
- Overlay networks manage routing between nodes and continuous joining/leaving. They have different properties which affect their performance.
- As a concrete overlay network algorithm, Kademlia solves the routing problem by employing a tree structure together with k-buckets.

Further Resources

Peerbridge Blockchain Messenger: https://github.com/peerbridge

An overview on P2P overlay network schemes:

Liz, Crowcroft; et al. (2005). "A survey and comparison of peer-to-peer overlay network schemes" (PDF). IEEE Communications Surveys & Tutorials. 7 (2): 72–93. CiteSeerX 10.1.1.109.6124. doi:10.1109/COMST.2005.1610546: https://www.cl.cam.ac.uk/teaching/2005/AdvSysTop/survey.pdf

An analysis of the Etherium blockchain P2P network:

Lucianna Kiffer, Dave Levin, and Alan Mislove. 2017. Stick a fork in it: Analyzing the Ethereum network partition. In Proceedings of the 16th ACM Workshop on Hot Topics in Networks (HotNets-XVI). Association for Computing Machinery, New York, NY, USA, 94–100. DOI:https://doi.org/10.1145/3152434.3152449