A WebRTC DHT

Andres Ledesma (UCY) in cooperation with Mikael (Peerialism).

Preface I

- Existing DHT overlays have been optimized using **one criteria** (network proximity, social links, content caching or others).

- An ***adaptable overlay*** that uses **more than one criteria**, should improve **performance**, **reliability** and **availability** in distributed (social) applications.

Preface II

- Currently we are experimenting with **Web browsers** to build an **initial DHT overlay**.

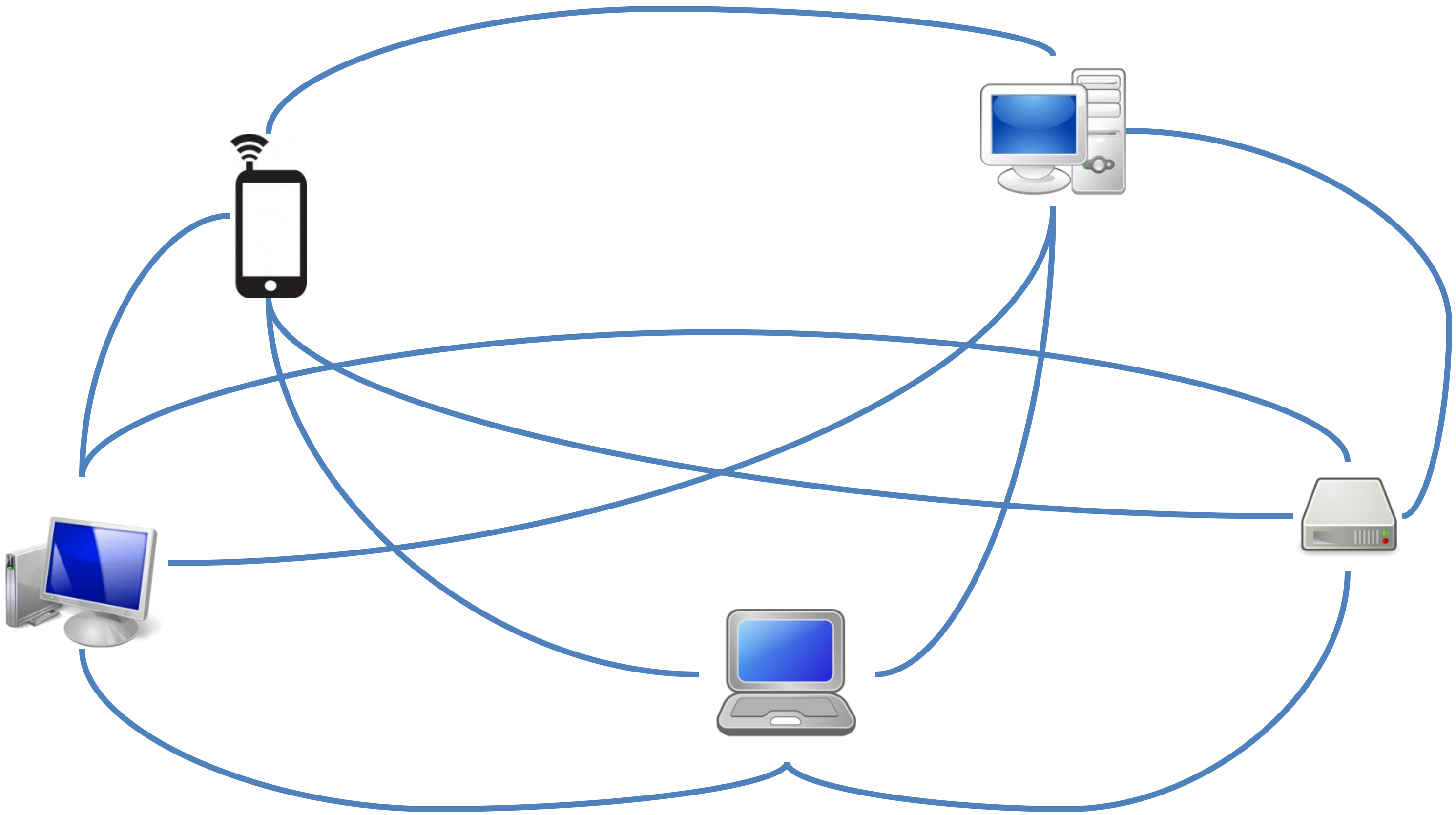
- After the implementation, we will extend the overlay to be ***adaptable***.

What do we mean by ***adaptable*** overlay?

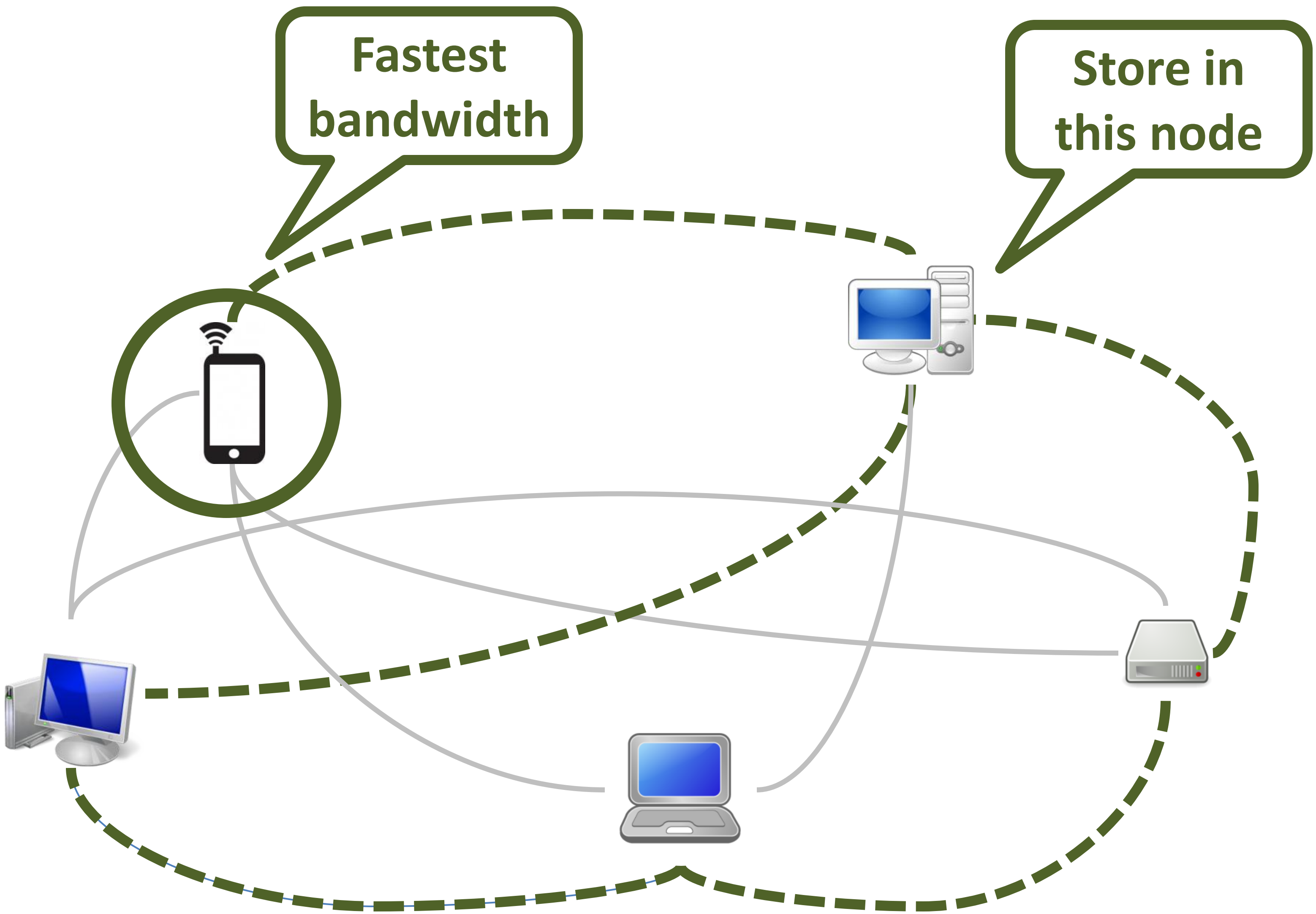
Different applications have different requirements.

- **A distributed YouTube would prefer high bandwidth among the peers.**
- **A cloud storage would prefer peers with sufficient storage and perhaps “good” reputation.**
- **A social network would favor peers that belong to actual friends.**

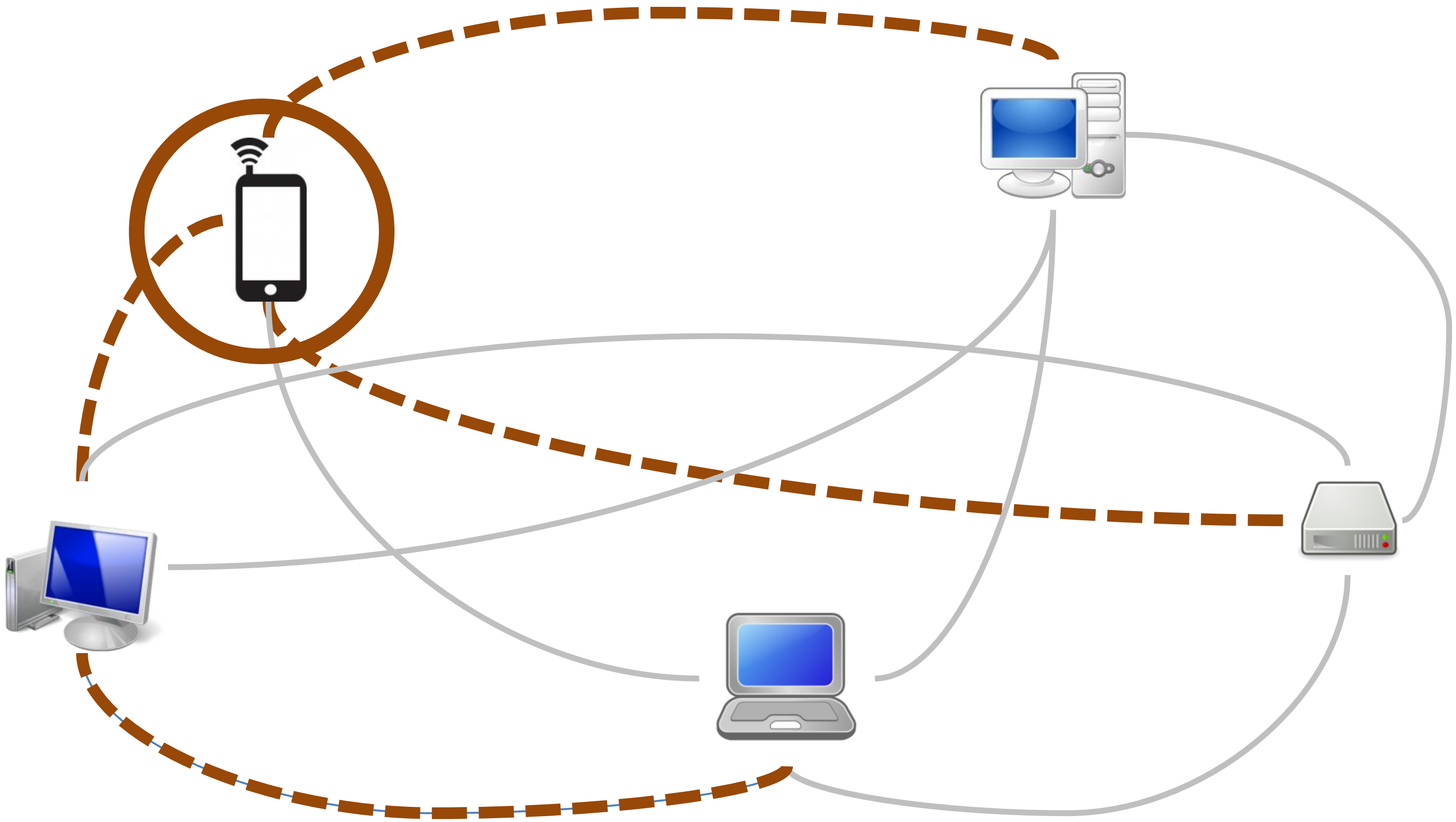
Example



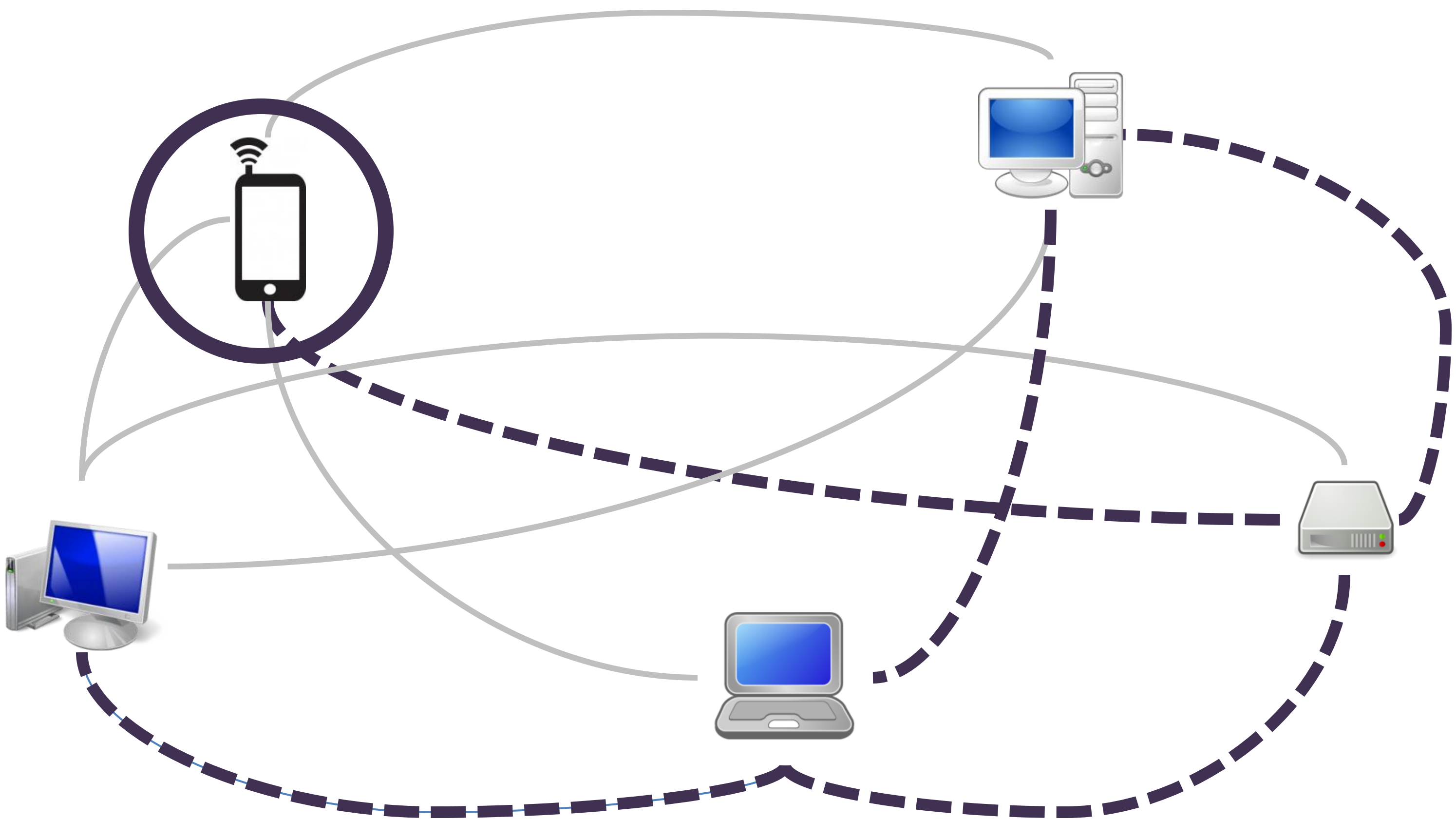
Fastest Possible Bandwidth



Social Links



Reputation



Problem Formulation

A node is a processing entity connected to a network. The network implements a Distributed Hash Table (DHT). Given that:

- Node $u \epsilon V$ and a set of functions f_i: $0 < i < C$ such that $f_i(u, v) \rightarrow R, u \epsilon V, v \epsilon N(u)$ (*v* is connected or it is considered to be a "neighbor" of *u*)
  - where C is the number of criterion specified by the application
  - where $N(u) = \{v: (u, v) \epsilon \exists\}$ (there is a connection between the nodes)
- Vector $w \epsilon 1 \times C$ where w_i is the element and represents the weight of f_i

Node u needs to find v such that:

$$v^* = \arg\max v \, [\sum_{i=0}^{C} w_i f_i(u, v)]$$

Problem Formulation

A node is a processing entity connected to a network. The network implements a Distributed Hash Table (DHT). Given that:

- Node $u \epsilon V$ and a set of functions $f_i : 0 < i < C$ such that

Among the options that a peer has to send data, use the one that gives the best match given a criteria.

- Vector $w \epsilon 1 \times C$ where w_i is the element and represents the weight of f_i

Node u needs to find v such that:

$$v^* = \arg\max v \left[\sum_{i=0}^{C} w_i f_i(u, v)\right]$$

Research Objectives

- Extend existing state-of-the-art DHT techniques
  - ***Adaptable*** overlays: adapt to application requirements
  - Backend for distributed and social applications
- **First step**: build an overlay for a DHT comprised of web browsers using WebRTC and HTML5.

Research Objectives

- Extend existing state-of-the-art DHT techniques
  - ***Adaptable*** overlays: adapt to application requirements
  - Backend for distributed and social applications

"Long-term" research direction

- **First** ... DHT comprised of web browsers using WebRTC and HTML5.

Research Objectives

- Extend existing state-of-the-art DHT techniques
  - ***Adaptable*** overlays: adapt to application requirements
  - Bac … plications

“Short-term” research direction

- **First step**: build an overlay for a DHT comprised of web browsers using WebRTC and HTML5.

Recap from Barcelona Sept. 2013

An ideal DOSN…

- No installation
- No extra hardware
- No configuration

All in the browser! ☺

A WebRTC DHT

Objective: Build a backend for distributed applications using web browsers.

- Select DHT to Implement
- **Build overlay**
- Test it
- Build apps with it

Select DHT to Implement: **Kademlia**

- Most suitable DHT for **unreliable peers**.
- Several improvements over the years.

Building a Reliable P2P System Out of Unreliable P2P Clients: The Case of KAD. Damiano Carra and Ernst W. Biersack. Institut Eurecom. Sophia-Antipolis, France. *ACM Conference on Emerging Network Experiment and Technology (CoNEXT), 2007.*

Active Research on Kademlia

- Fast Lookups
- Social analysis
- Network proximity
- Caching system

Active Research on Kademlia

<ins>**Embracing the Peer Next Door: Proximity in Kademlia.**</ins>
Sebastian Kaune, *et al*. Technische Universitat Darmstadt. KOM Multimedia Communications Lab. *International Conference on Peer-to-peer Computing 2008. IEEE Computer Society Press.*

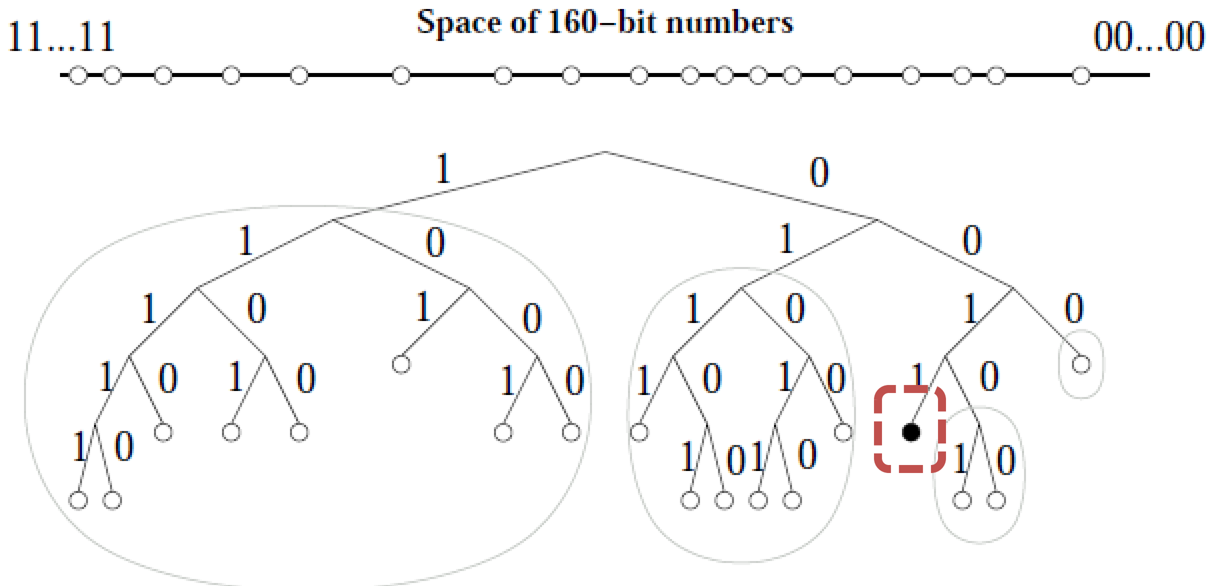
<ins>**Improving the Routing Performance of KAD through Social Network Analysis.**</ins>
Xiangtao Liu, *et al*. Institute of Computing Technology, Chinese Academy of Science. *International Symposium on Computers and Communications, 2010.*

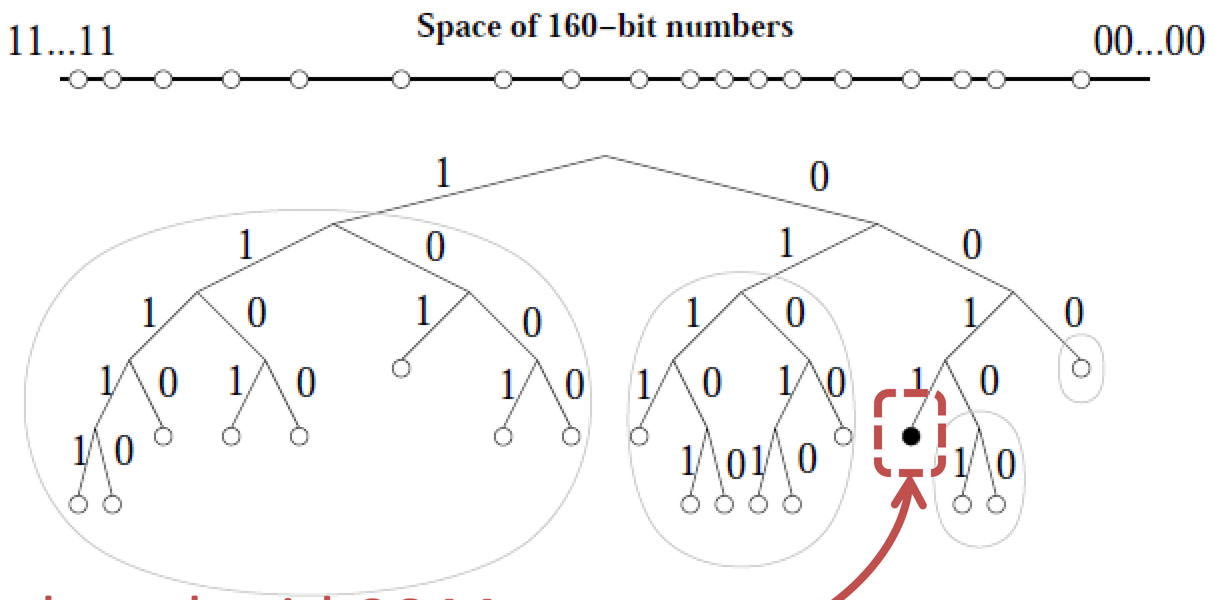
<ins>**Revisiting Why Kad Lookup Fails.**</ins> **Bingshuang Liu, *et al*. Peking University, Beijing, China. *IEEE International Conference on Peer-to-Peer Computing, 2012.***

<ins>**Kaleidoscope: Adding Colors to Kademlia.**</ins> **Gil Einziger, *et al*. Computer Science Department, Technion Haifa. *IEEE International Conference on Peer-to-Peer Computing, 2013.***

How does **Kademlia** work?

- 160-bit keys and 160-bit node identifiers
- Distances between nodes and keys are calculated using XOR
- **Each step is one bit closer to the destination**.
  - Peers have routing tables
  - Routes are grouped on 160 buckets (one for each different bit)


Space of 160−bit numbers
11...11
00...00



This node has the id **0011**

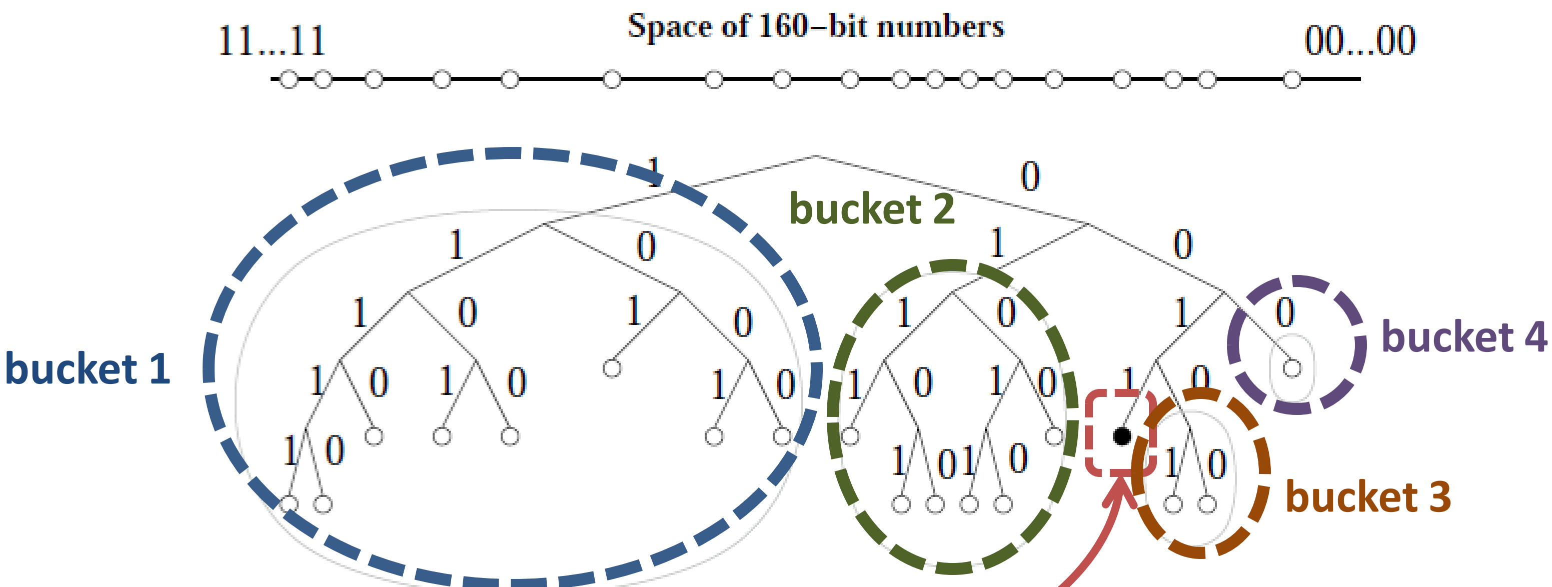
In this node the others are grouped in buckets:

1* bucket 1**

01 bucket 2**

0001 bucket 3

0000 bucket 4 (contains one node)



This node has the id **0011**

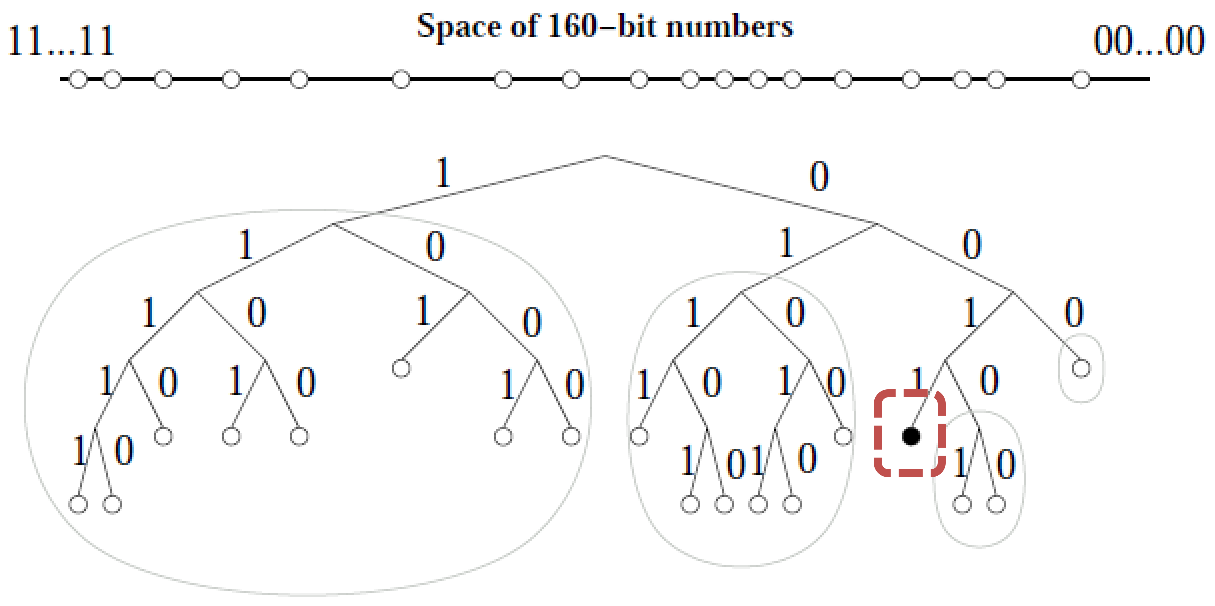
In this node the others are grouped in buckets:

1* bucket 1**

01 bucket 2**

0001 bucket 3

0000 bucket 4 (contains one node)



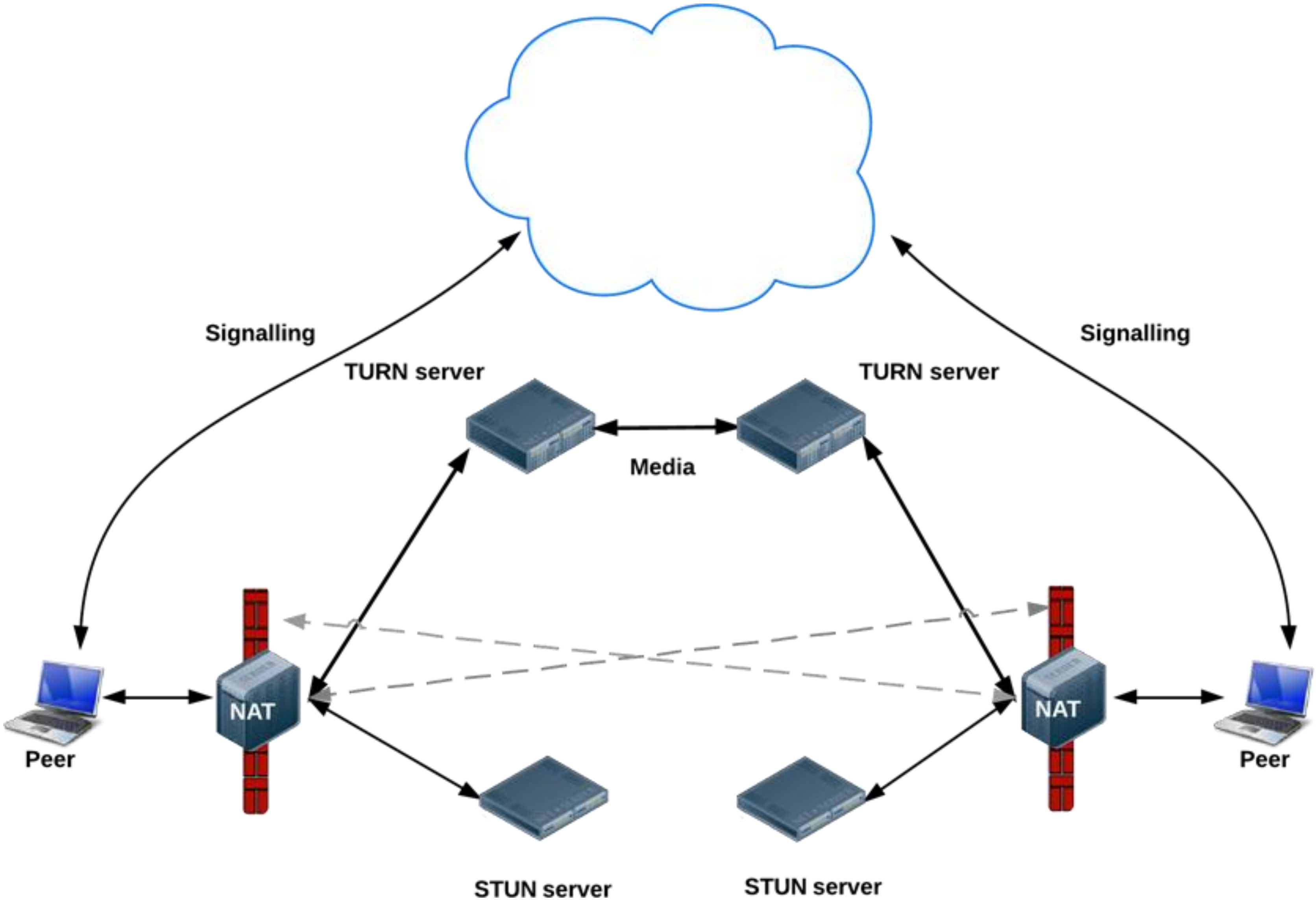
<u>Kademlia: A Peer-to-peer Information System Based on the XOR Metric.</u>
Petar Maymounkov and David Mazieres. New York University.
International Workshop on Peer-to-peer Systems, 2002.

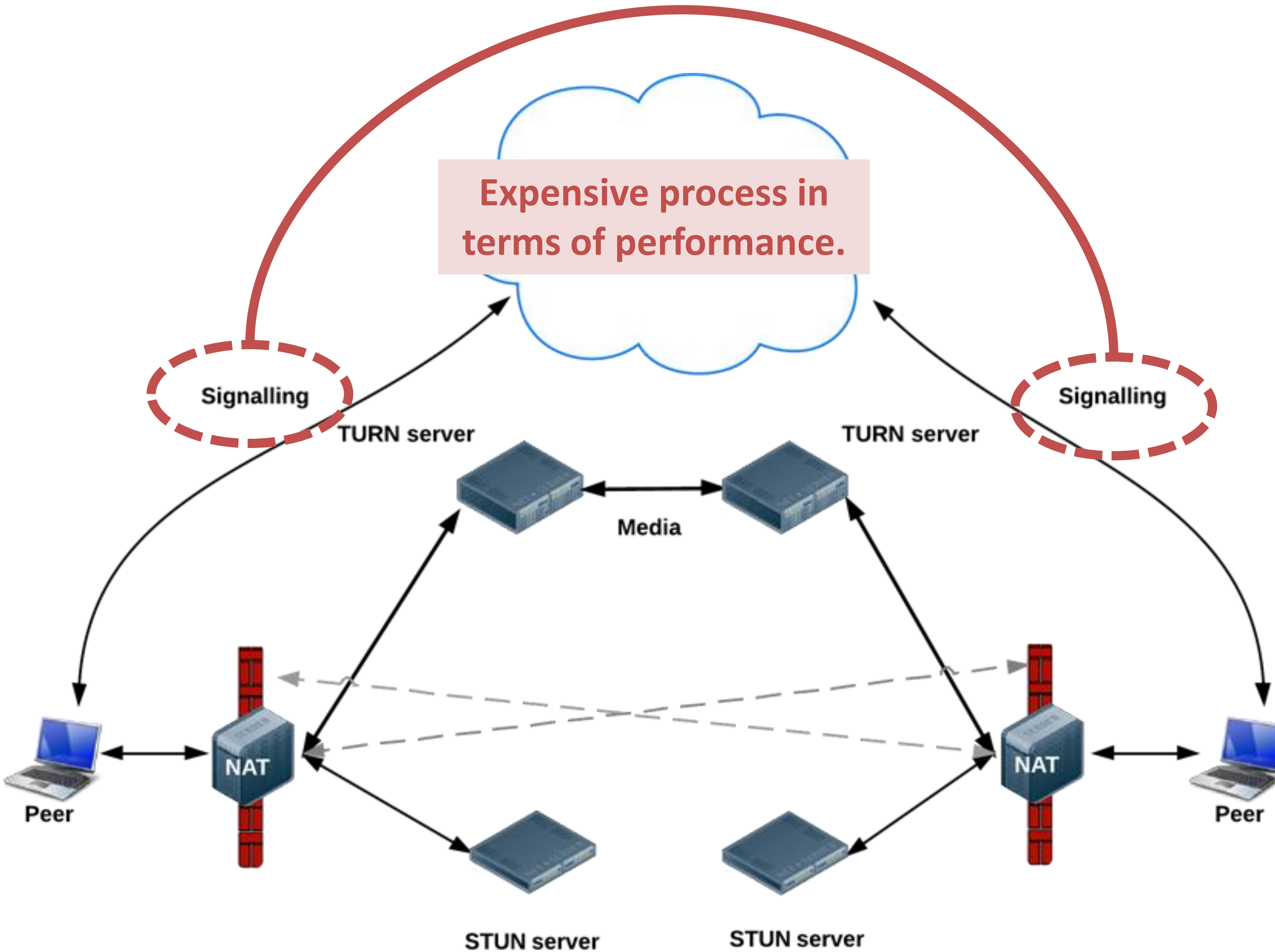
Tools used to build a WebDHT

- **Storage:** HTML5 (through providing the “localstorage” object that enables permanent storage)
- **P2P Communication :** WebRTC a W3C Standard and the only mechanism to enable browser-to-browser communication without plug-ins or third-party extensions (Java, ActiveX, Flash, etc.).

WebRTC Considerations

- For peers to contact each other, they need a signal server for the initial connection.
  - **ICE** (Interactive Connectivity Establishment)
  - **STUN** (Session Traversal Utility for NAT)
  - **TURN** (Traversal Using Relays around NAT)


Signalling
Signalling
TURN server
TURN server
Media
NAT
NAT
Peer
Peer
STUN server
STUN server



Expensive process in terms of performance.
Signalling
Signalling
TURN server
TURN server
Media
NAT
NAT
Peer
Peer
STUN server
STUN server

Memory

- There is a limited amount of RAM the OS assigns to the browser (between 2 to 4 Gigabytes of RAM).
- Each connection demands a chunk of this memory.

...Aw, Snap...!

- Kademlia requires communication with 160 buckets. There are two ways of doing this:
  - Connect and disconnect
    - OR
  - Keep connections open

- Kademlia requires communication with 160 buckets. There are two ways of doing this:
  - Connect and disconnect
    - OR
  - Keep connections open

- **This will trigger too many hits on the signal server.**
- **For every request to the signal server, the performance is affected.**

- Kademlia requires communication with 160 buckets. There are two ways of doing this:
  - Connect and disconnect
    - OR
  - Keep connections open

- **Memory limitations of the browser.**
- **Consider mobile devices…**
- **Consider peers leaving…**

Can we “extend” Kademlia?

=Memory=

- Keep as few open connections as possible

=Performance=

- Try to avoid opening and closing connections

=Distribution=

- Use XOR to compute distances

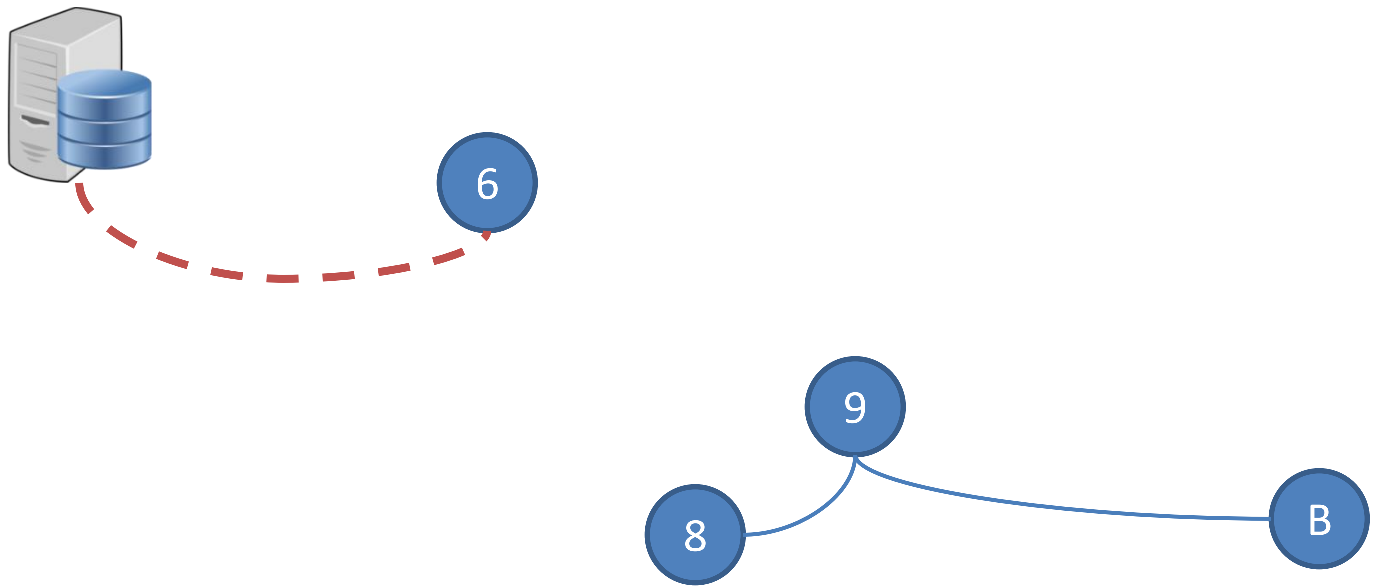
This is how we “extended” Kademlia…

- Node aggregation and find/store key operations are done via **routing** instead of **opening new connections**.
- **XOR-wise**
  - Forward OR
  - Store/Connect

(-)Disadvantages:

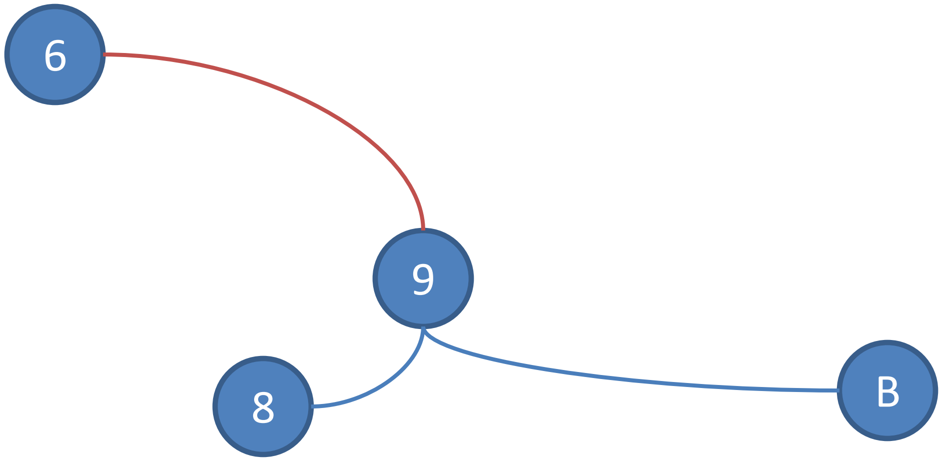
- Lookups will not perform as fast as opening and keeping the connections open.
- TTL is needed when packages cannot be delivered.

Node Aggregation



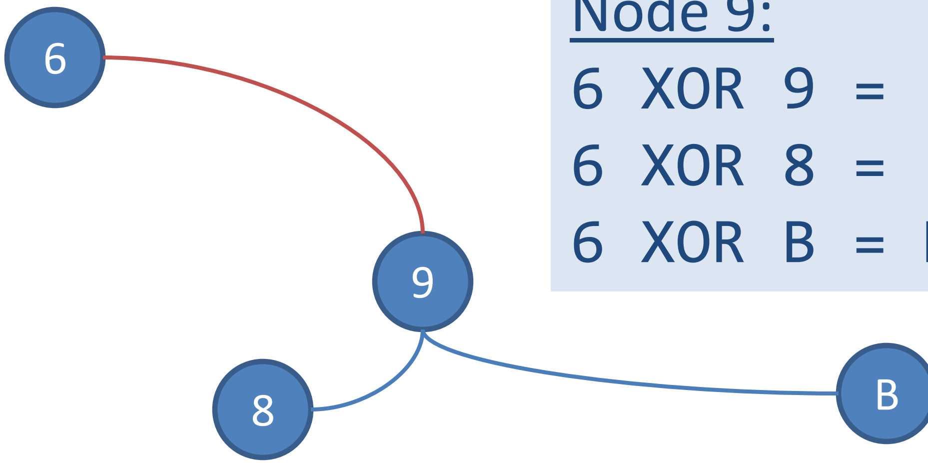
A new node contacts the signal server to receive an identifier and a list of peers. The identifier is a string such as “xpqr34trs”. The node uses SHA1 to transform it into a 160-bit identifier.

Node Aggregation

A new node contacts any other peer in the network.

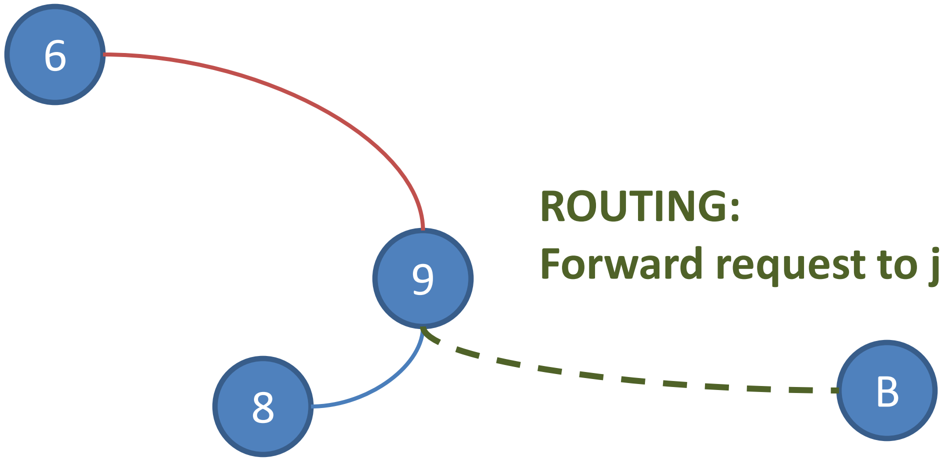
Node Aggregation



Node 9:
6 XOR 9 = F
6 XOR 8 = E
6 XOR B = D

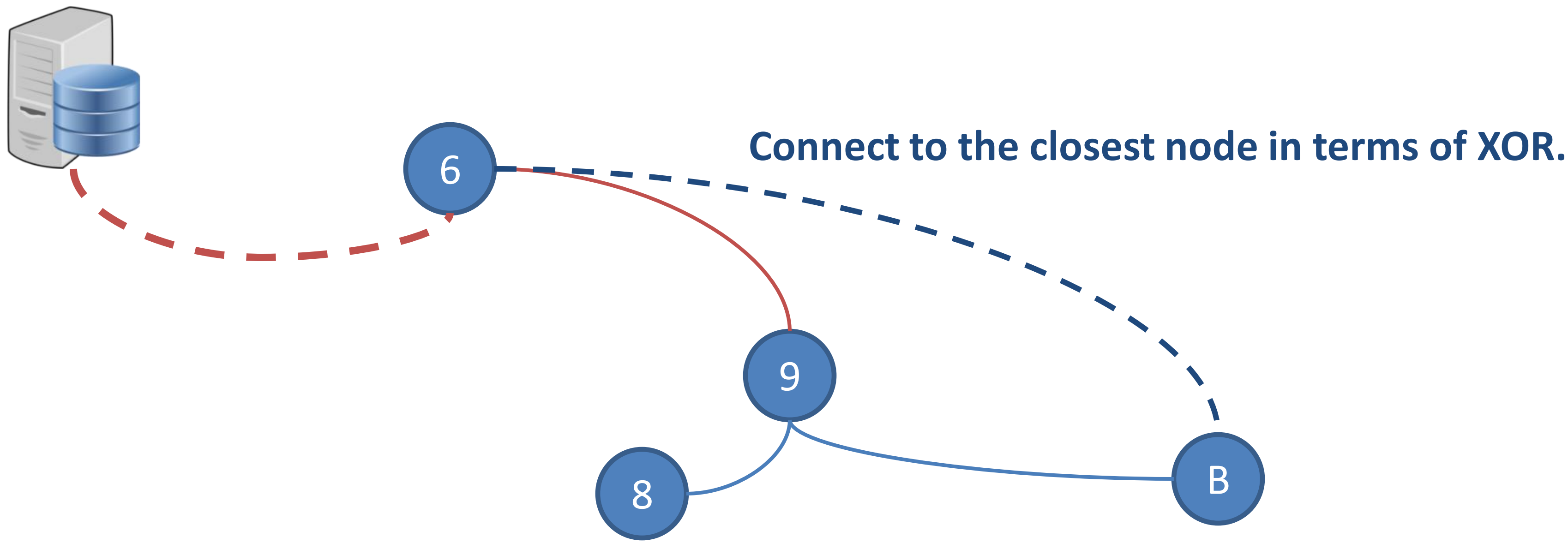
The contacted peer finds among itself and its open connections the closest node in terms of XOR.

Node Aggregation



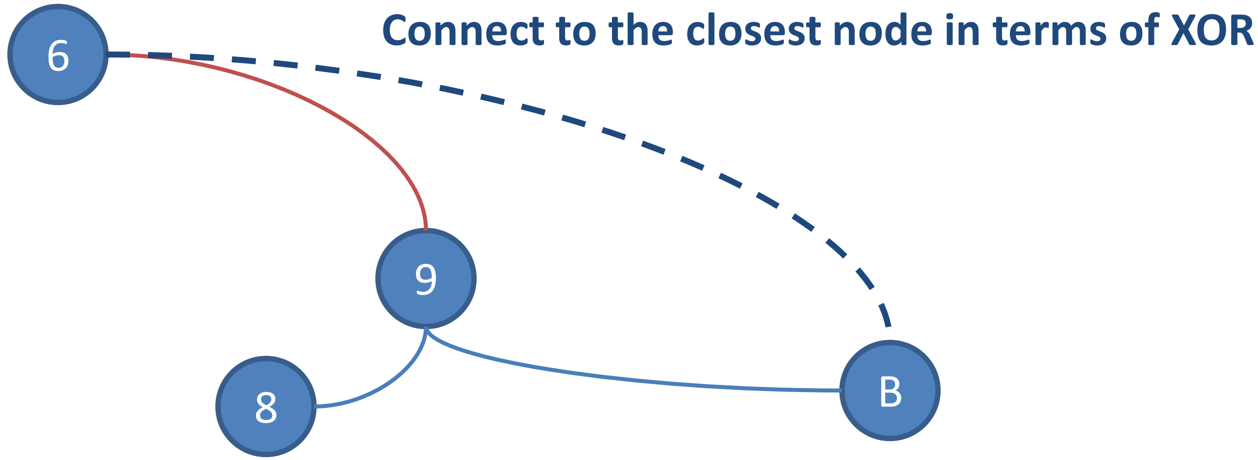
ROUTING:
Forward request to join the network.

Node Aggregation



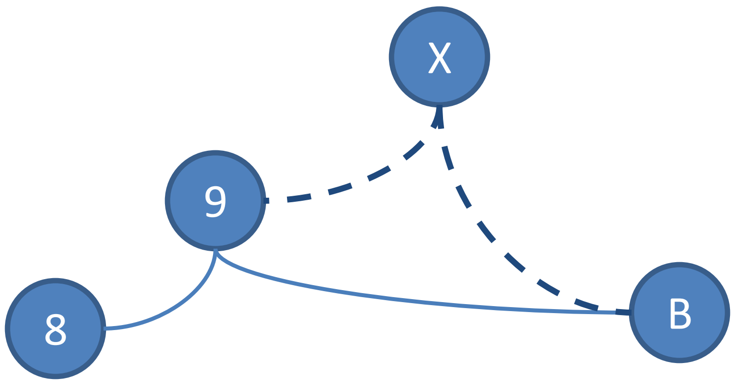
Node 9 forward the request to node B. Node B would compute in the same way and ultimately connect to the new node, node 6.

Node Aggregation



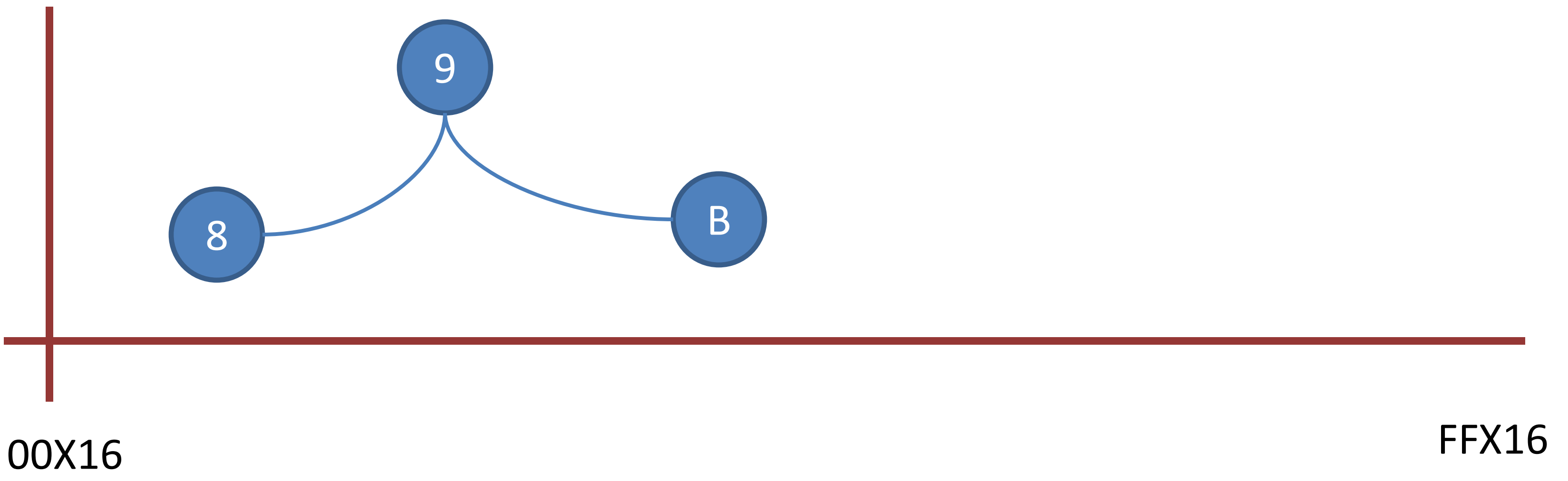
Node 9 forward the request to node B. Node B would compute in the same way and ultimately connect to the new node, node 6.

Node Aggregation

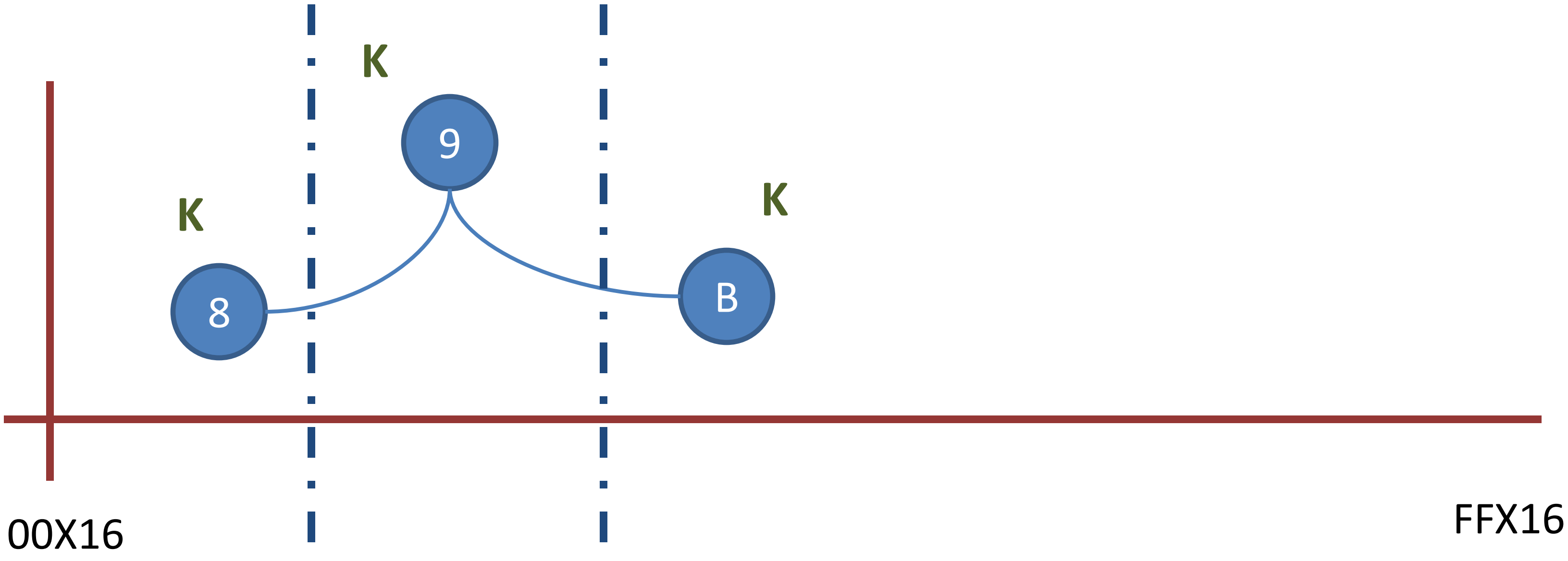


If a new node is inserted in between two other nodes, a connection from both ends is opened.

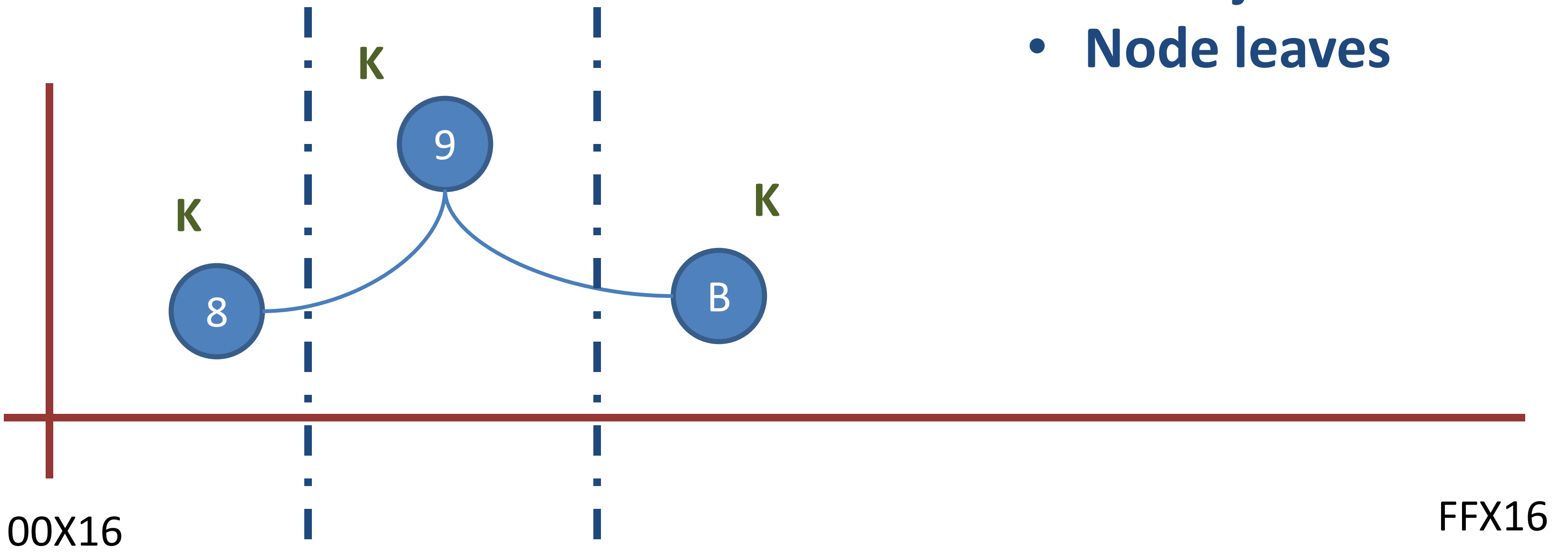
Key Space



Key Space

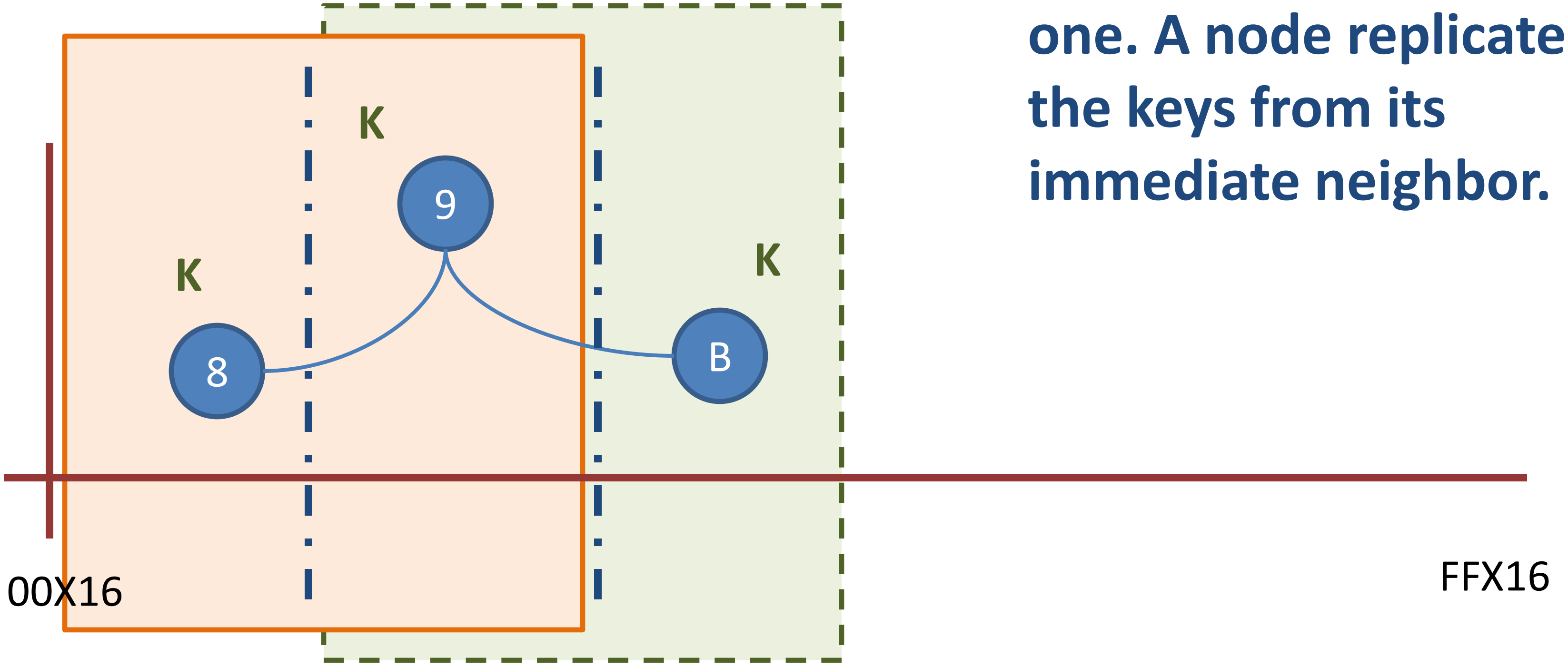


Key Space



Key redistribution:
- **Node joins**
- **Node leaves**

Key Space



Replication of degree one. A node replicates the keys from its immediate neighbor.

Find / Store Key

- Serialize data (JSON)
- Hash with SHA1 (node identifier as salt)
  - The result is a 160-bit key
- Find the closest XOR distance among the open connections and the peer itself.
  - Forward OR
  - Store
- On each peer "stamps" its id to the message => create a **trace route**

Find / Store Key

- Serialize data (JSON)
- Hash with SHA1 (node identifier as salt)
  - The result is a 160-bit key
- Find the closest XOR distance among the open connections and the peer itself.
  - Forward OR
  - Store
- On each peer "stamps" its id to the message => create a **trace route**

Just like node aggregation

Example : Serialization and Key Construction

```
nodeId = “rqpws49p321”;
data = [
  {
    hello: “world:,
    name: “andres”,
    project: “iSocial”
  }
];

var value = JSON.Stringify(data);
var key = SHA1(nodeId + value);

console.log(“key: ” + key);
console.log(“value: ” + value);
```

```
> key: 97295d659d44340d72f084553239428de4b4f094
> value: [{hello: “world:, name: “andres”, project: “iSocial”}]
```

Follow us!

github.com / landreus / isocial

- Read me
- Commit logs
- Latest version

Follow us!

github.com / landreus / isocial

alias project

- Read me
- Commit logs
- Latest version

What do we have so far?

- 160-bit key management
- XOR-wise distance calculation
- Node aggregation
- Find node and key
- Store
  - put(key, value)
- Retrieve
  - get(key)
- Trace communication route
  - Avoid deadlocks
- TTL

Finalize the first step…

- Implement look-ahead to speed up the finding process
- Testing:
  - Simulate 100 000+ nodes
  - Aspects to measure
    - Performance – response time of a find key operation
    - Reliability – how many nodes can fail while still maintaining an operation network?
  - More testing along with Peerialism…
- Come up with improvements
  - Allow new connections under certain conditions to reduce the number of hops => **subject to testing**
- Build demo applications
  - Micro-blogging
  - Social network
- Put it on paper ☺

What comes after the first step?

- Investigate DHT overlays
  - Design ***adaptable*** or ***multiple-criteria*** overlays
    - bandwidth
    - network proximity
    - reputation
    - social links
    - etc…
  - Implementation of a prototype
  - Allow applications to choose overlays based on requirements
- Experiment with key space
  - Faster recovery when nodes go offline
  - Faster lookups in large systems
  - Improve replication strategy (performance and reliability)

Thank you for your attention! ☺

Questions? Feedback?