# ASD - Project 1

Sonia Becz
s.becz@campus.fct.unl.pt
MEI, DI, FCT, UNL

## ABSTRACT

This project presents the design and implementation of a decentralized peer-to-peer communication protocol using Distributed Hash Tables (DHTs) to enable efficient and reliable message routing in dynamic network environments. By integrating the Kademlia DHT protocols, the system provides flexible peer lookup and fault tolerance through structured topologies and proximity-aware routing. A point-to-point communication protocol is employed to manage direct messaging, with a helper node mechanism to ensure message delivery even when nodes are offline.

## 1 INTRODUCTION

This project aims to develop a decentralized, peer-to-peer communication protocol that enables efficient and reliable message exchange between nodes. The first phase of the project focuses on constructing a point-to-point communication system using a Distributed Hash Table (DHT) to manage peer identities and locations, facilitating direct messaging without relying on static IP addresses. Each node in the network is assigned a unique identifier, allowing messages to be routed dynamically based on current network conditions, even as nodes change their IPs or ports.

Initially, the project introduces a broad context for distributed communication systems, highlighting the significance of efficient peer-to-peer networks in supporting decentralized applications. As communication in such networks is prone to failures due to nodes going offline, this protocol includes robustness mechanisms, including helper nodes that store and forward messages if the target peer is unreachable, enhancing the system's resilience.

The goal of this project phase is to implement one of two core DHT protocols: Chord and Kademlia. These protocols will enable nodes to locate peers within the network efficiently, facilitating the reliable delivery of messages. Alongside the DHT, the point-to-point communication protocol manages message delivery, ensuring robustness and efficiency.

This document proceeds with an in-depth explanation of the architecture.

## 2 RELATED WORK

The design and implementation of distributed hash tables (DHTs) have been a core area of study in decentralized network architectures, with a focus on creating scalable, fault-tolerant systems that can handle high levels of churn. Two prominent DHT protocols, Chord and Kademlia, have been particularly influential and widely adopted in both academic and practical applications.

### 2.1 Kademlia

The DHT implemented for this project is Kademlia. Kademlia, introduced by Maymounkov and Mazieres, presents an approach to DHT design. Unlike Chord's ring-based structure, Kademlia organizes nodes in a binary tree, allowing it to leverage XOR-based distance metrics to determine the shortest path between nodes. Each node in Kademlia maintains a routing table composed of k-buckets, which are populated with nodes based on their proximity in the ID space. This proximity-aware routing strategy has proven effective in real-world applications such as BitTorrent and the Interplanetary File System (IPFS), where it enhances scalability and reduces latency. Kademlia's approach to updating routing tables based on observed network interactions enables nodes to become aware of other nodes in the system dynamically, which improves robustness and network awareness. The protocol's self-adjusting nature allows it to perform well in high-churn environments, where nodes frequently join and leave the network, a key consideration for peer-to-peer applications.

### 2.2 Challenges

In addition to DHT-based systems, point-to-point communication protocols are essential for managing direct messaging between nodes in decentralized networks. A recurring challenge in these systems is maintaining reliable message delivery when recipient nodes may go offline intermittently. One solution commonly explored in literature is the concept of helper nodes, which temporarily store messages intended for offline peers. When a node attempts to deliver a message but finds the target unavailable, the message is instead forwarded to a helper node, chosen based on proximity in the DHT structure. This helper node will attempt delivery on behalf of the sender, enabling time-decoupled messaging. This strategy not only improves the resilience of the network by reducing the likelihood of message loss but also supports asynchronous communication, a critical feature in many distributed applications. However, helper node strategies introduce challenges related to potential message duplication, and researchers have proposed various mechanisms, such as sequence numbers or message hashes, to ensure messages are delivered exactly once.

### 2.3 Project

Building on these foundational works, our project integrates both Chord and Kademlia as underlying DHT structures for the communication protocol. By leveraging insights from previous studies, we aim to explore the effectiveness of each DHT protocol in supporting reliable point-to-point communication in a peer-to-peer setting.

## 3 IMPLEMENTATION

The implementation of this project involves the design and development of a peer-to-peer communication protocol using a Distributed Hash Table (DHT) to facilitate reliable message routing across a dynamic, decentralized network. This section presents a detailed breakdown of the main components of our solution, including the core DHT protocols, the point-to-point communication module, and the message routing and storage mechanisms. Each component is implemented in Java within the Babel framework.

## 3.1 Distributed Hash Table (DHT) Protocols

This module includes Kademlia protocol, allowing for flexibility in peer lookup and routing efficiency. Kademlia uses a binary tree structure with XOR-based distance calculations. Each node maintains k-buckets, containing references to other nodes sorted by their proximity in the ID space. Kademlia's routing algorithm chooses paths based on XOR distance, allowing for quick and resilient lookups. The protocol monitors network traffic to dynamically populate these k-buckets, updating them as nodes join or leave the network. This structure enables efficient communication in networks with high churn, as nodes can maintain robust routing tables with minimal overhead. Our implementation of Kademlia includes pseudocode for k-bucket management, node discovery, and routing table updates

*3.1.1 Pseudocode.* **State:**

- `routingTable` // Kademlia routing table instance
- `channelReady` // Boolean flag indicating if the channel is ready

**Algorithm 1** Kademlia DHT Protocol

```
Upon Init do:
Set channelReady to false
Register Lookup request handler with uponLookup
Subscribe to ChannelCreated notifications with uponChannelCreated

Upon Init(props) do:
Initialize routingTable with myPeerID

Upon ChannelCreated(notification, sourceProto) do:
cId ← notification.getChannelId()
Register shared channel with cId
Set channelReady to true

Upon Lookup(request, protoID) do:
if channelReady is false then
    return
end if
lr ← new LookupReply(request.getPeerID(), request.getMid())
closestPeers ← findClosestPeers(request.getPeerID())

for p in closestPeers do
    Add p.getLeft() and p.getRight() to lr
end for
Send lr as a reply with protoID

Function updateRoutingTable(peerID, peerHost) do:
Update routingTable with peerID and peerHost
```

## 3.2 Point-to-Point Communication Protocol

The point-to-point communication protocol is responsible for ensuring reliable message delivery between nodes. This protocol manages the complexities of sending messages to offline nodes by implementing a helper node mechanism. When a message is sent to a node that is temporarily unreachable, it is forwarded to a helper node—a peer chosen based on its proximity to the destination in the DHT structure. The helper node stores the message and periodically attempts to deliver it until the recipient becomes reachable. This mechanism is designed to enhance network robustness, enabling communication even when nodes are intermittently unavailable. To prevent duplicate deliveries, the protocol uses message sequence numbers and acknowledgments to track successfully delivered messages

*3.2.1 Pseudocode.* **State:**

- `myself` // Host instance representing this node
- `DHT_PROTO_ID` // Protocol ID for the Kademlia DHT
- `channelId` // Channel identifier for communication
- `lock` // Lock object for synchronization
- `pongReceived` // Boolean flag indicating if a pong response was received
- `kademliaDHT` // Instance of KademliaDHT
- `messageQueue` // Map storing messages waiting to be sent, keyed by UUID
- `peerStatus` // Map storing peer connection status

## 3.3 Application Layer Interface

the DHT and point-to-point communication protocols to facilitate the generation of message requests and the collection of performance metrics. Two versions of the application are implemented: an interactive version for debugging and a batch version for large-scale experiments. The application layer allows users to configure parameters such as the rate of request generation and payload size, enabling a range of experiments to test the system's scalability, latency, and reliability under different network conditions. This layer also logs message delivery statistics, enabling a detailed analysis of the protocol's performance across various network configurations

## 4 EXPERIMENTAL EVALUATION

I didn't manage to conduct experimental evaluation in time.

## 5 CONCLUSIONS

In this project, we developed a decentralized peer-to-peer communication protocol utilizing Distributed Hash Tables (DHTs) to facilitate efficient and resilient message routing across a dynamic network environment. Implementing Kademlia protocol provided flexible option for peer lookup, using proximity-aware routing for potentially optimized performance in different scenarios. The addition of a helper node mechanism within the point-to-point communication protocol ensures message delivery even when the target node is temporarily offline, enhancing robustness.

While the design and implementation phases successfully established a functioning protocol, time constraints prevented me from conducting a comprehensive experimental evaluation. As a result, I was unable to empirically validate key performance metrics such as message latency, reliability, and scalability under various

**Algorithm 2** Point2PointComm Protocol

```
 1: Upon Init(props) do:
 2: Set channelId by creating a TCP channel with properties from props

 3: Upon Init(props) do:
 4: Trigger ChannelCreated notification
 5: if props contains "contact" then
 6:     Parse contact host from props and open a connection
 7: end if
 8: Set up a periodic timer for retrying messages every 30 seconds

 9: Upon Message(msg, from, sourceProto, channelId) do:
10: Trigger Deliver notification with msg's sender, messageID, and payload

11: Upon PingMessage(msg, from, sourceProto, channelId) do:
12: Update kademliaDHT's routing table with msg's sender ID and from
13: Create PongMessage with local peer ID and send it back to from

14: Upon PongMessage(msg, from, sourceProto, channelId) do:
15: Set pongReceived to true and notify lock
16: Update kademliaDHT's routing table with msg's sender ID and from
17: Update peerStatus for from to true

18: Upon FindNodeMessage(msg, host) do:
19: For each peer in msg.getPeers()
20: if peer's ID does not match local peer ID then
21:     Open a connection to peer.getRight()
22:     Update kademliaDHT routing table
23: end if

24: Upon SendRequest(request, protoID) do:
25: Create a Lookup request with the destination peer ID and message ID
26: Send lookup request via DHT protocol and add request to messageQueue

27: Upon LookupReply(reply, protoID) do:
28: if reply has available peers then
29:     Select closestPeer from peers
30:     if closestPeer is online then
31:         Retrieve originalRequest from messageQueue
32:         Create and send Message with payload to closestPeer
33:         Remove originalRequest from messageQueue
34:     else
35:         Call handleOfflinePeer with remaining peers and reply
36:     end if
37: end if

38: Upon Timer(timer, timerId) do:
39: for each request in messageQueue do
40:     Resend request via DHT protocol
41: end for

42: Upon OutConnectionUp(event, channelId) do:
43: Update peerStatus for peer
44: Upon OutConnectionDown(event, channelId) do:
45: Update peerStatus for peer
46: Upon InConnectionUp(event, channelId) do:
47: Update peerStatus for peer
48: Upon InConnectionDown(event, channelId) do:
49: Update peerStatus for peer
```

network conditions. Future work should focus on conducting these evaluations to quantify the protocol's performance, identify potential bottlenecks, and validate the effectiveness of the helper node mechanism in diverse network scenarios.

Despite the lack of experimental results, this project lays a solid foundation for further exploration of DHT-based communication protocols. Future studies can build on this work by performing detailed experiments to optimize the protocol further and develop adaptive mechanisms to handle high levels of network churn and varying message loads.