26 July 2023

Peer-to-Peer Systems and Security (IN2194)
Midterm Project Report

Doğan Can Hasanoğlu, Emin Sadikhov

1. Changes to Initial Report

Without making any modifications to our initial report, we opted to maintain our initial findings and strategies in the report. Using the initial document as a solid foundation, we continued to build upon our preliminary conclusions, ensuring a consistent advancement and thorough examination of our project.

2. Architecture

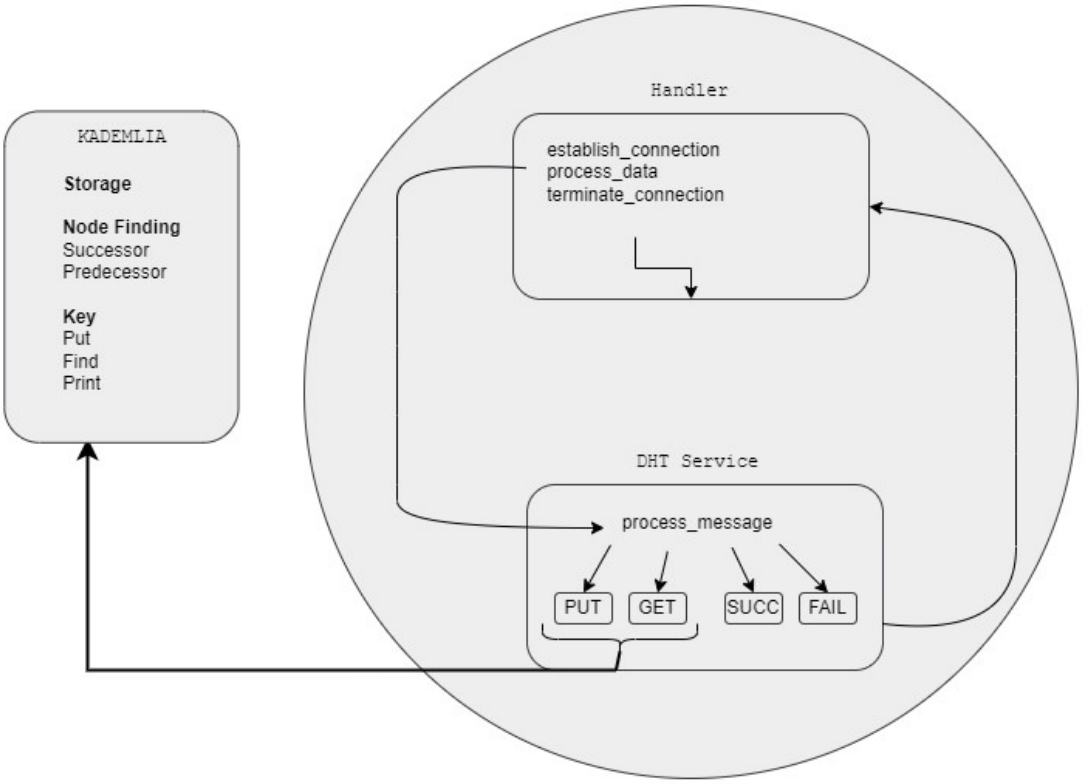
We have decided to architect a Distributed Hash Table (DHT) utilizing the principles of the Kademlia protocol. Our system design consists of two primary components: the API Handler and the DHT Service. The API Handler is tasked with initiating connections, first with the bootstrap node, and subsequently with additional peers. It is also in charge of processing incoming requests and forwarding them to the DHT Service. The DHT Service, conversely, takes on the responsibility of performing PUT and GET operations, thereby managing the data storage and retrieval within the distributed network. This bifurcated structure of our system ensures an efficient and responsive design, capable of handling a high throughput of data transactions.

2.1. Logical Structure

1. **API Handler**: This component initiates the server and manages its fundamental functions. It further creates an instance of the DHT Service class, acting as a server object, which allows for interaction with the DHT network.

2. **DHT Service**: This component manages the Distributed Hash Table (DHT) service. It maintains communication with the API Handler, processing incoming messages in the

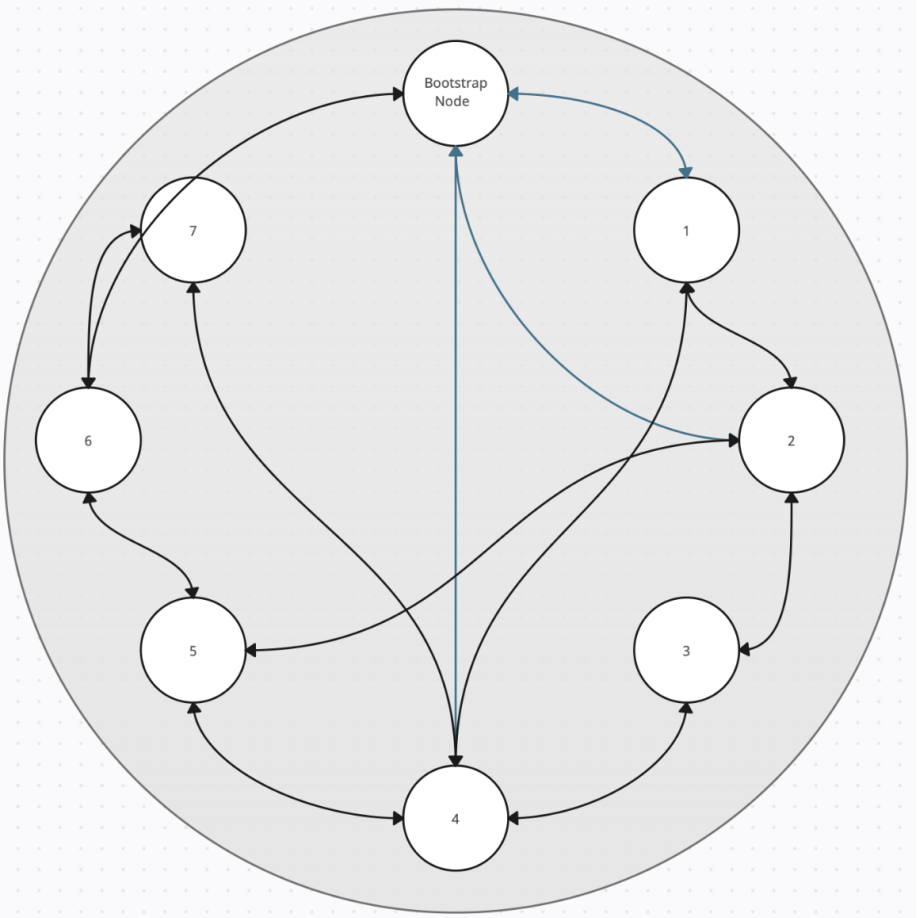
context of the DHT. Furthermore, it utilizes the Kademlia Service for operations within the DHT, aligning with the established protocol rules.



2.2. Process Architecture & Networking

We plan to employ the Kademlia protocol, basing our logic on the depicted structure. We have utilized the Asyncio library to define our node structure, enabling multiple clients to connect to our node and initiate PUT and GET requests. Upon completion of the Kademlia implementation, we will proceed to adapt all required mechanisms to support multi-threading operations. This approach will allow for simultaneous processing, improving the efficiency and responsiveness of our system.

```
Server started at 127.0.0.1 : 7401
51 1 1 0
key length 32
DHT_PUT: (b'emincanemincanemincanemincanemin':b'hello_world')
```



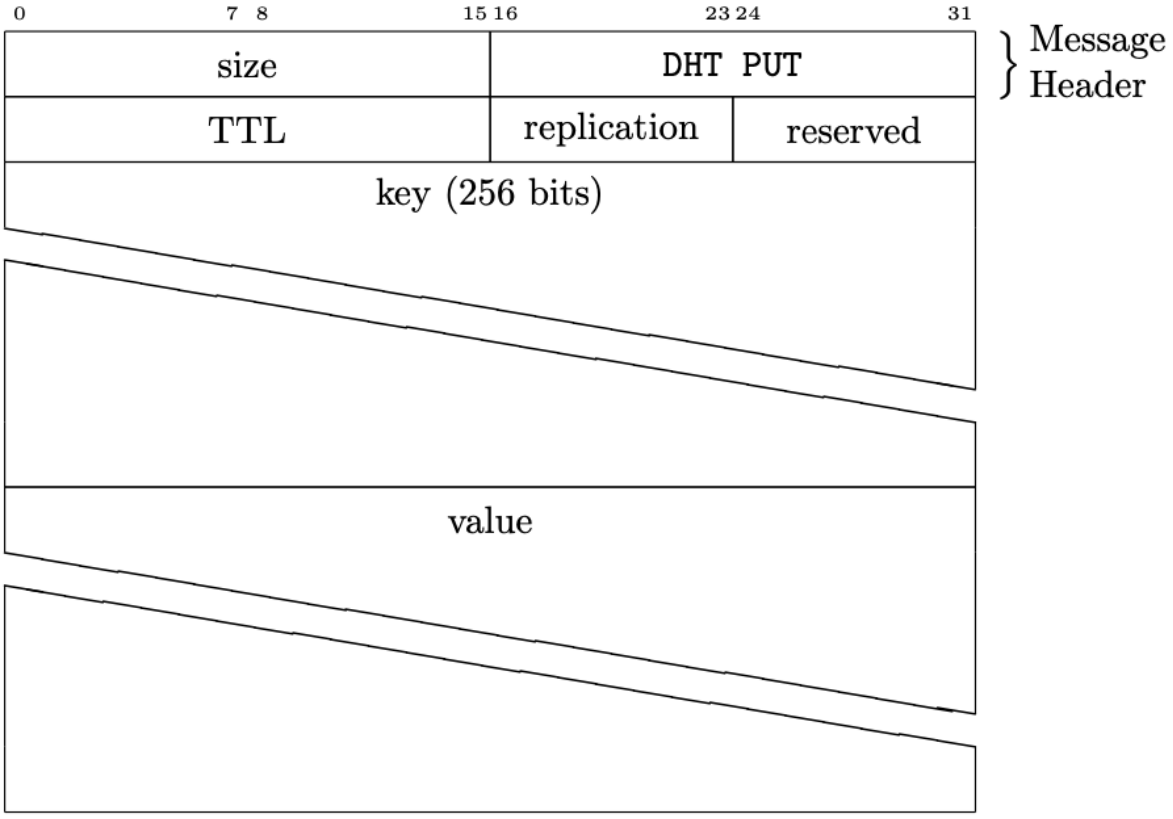
2.3. Security Measures

We plan to employ the SHA-256 for hashing operations, providing a high level of data integrity within our system. Additionally, we'll implement the RSA to facilitate encryption and decryption procedures, ensuring the confidentiality of our data transmissions. Keys required for these cryptographic operations will be generated and shared in advance. As we move towards containerizing our project using Docker, we'll implement an additional security layer. We will generate and incorporate cryptographic keys and certificates within our Docker containers. This will allow us to establish trusted communication channels and validate the authenticity of our services, and strengthen the overall security posture of our project.

3. Peer-to-Peer Protocol

3.1. DHT PUT

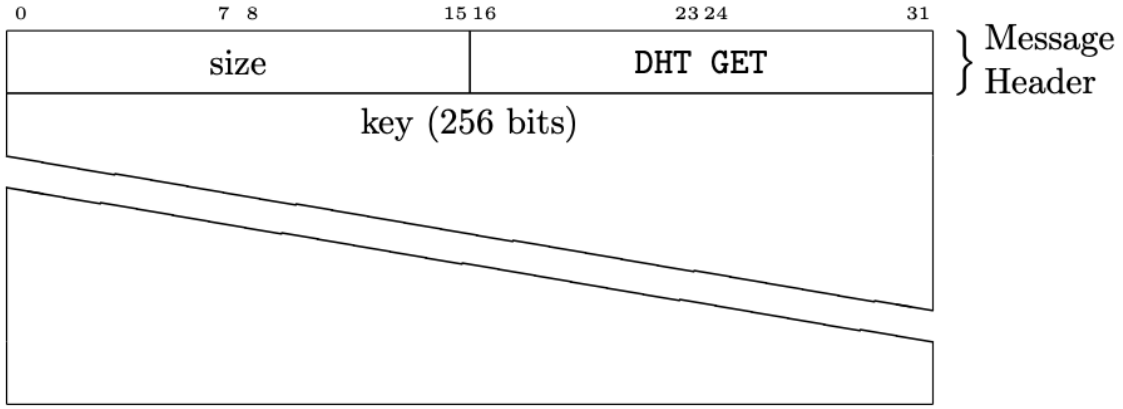
This message is used to store the provided key-value pair in the DHT.



- Size: Size of the body of the of the DHT PUT with the message headers
- Type: Type of the message (DHT PUT)
- TTL: Time to live in seconds given key value pair should be stored in the network. Peer responsible for the storing following key value pair will try to store it for the provided TTL if possible
- Replication: Number of copies should be done for the following key-value (over different peers)
- Reserved: Reserved space for extra messages

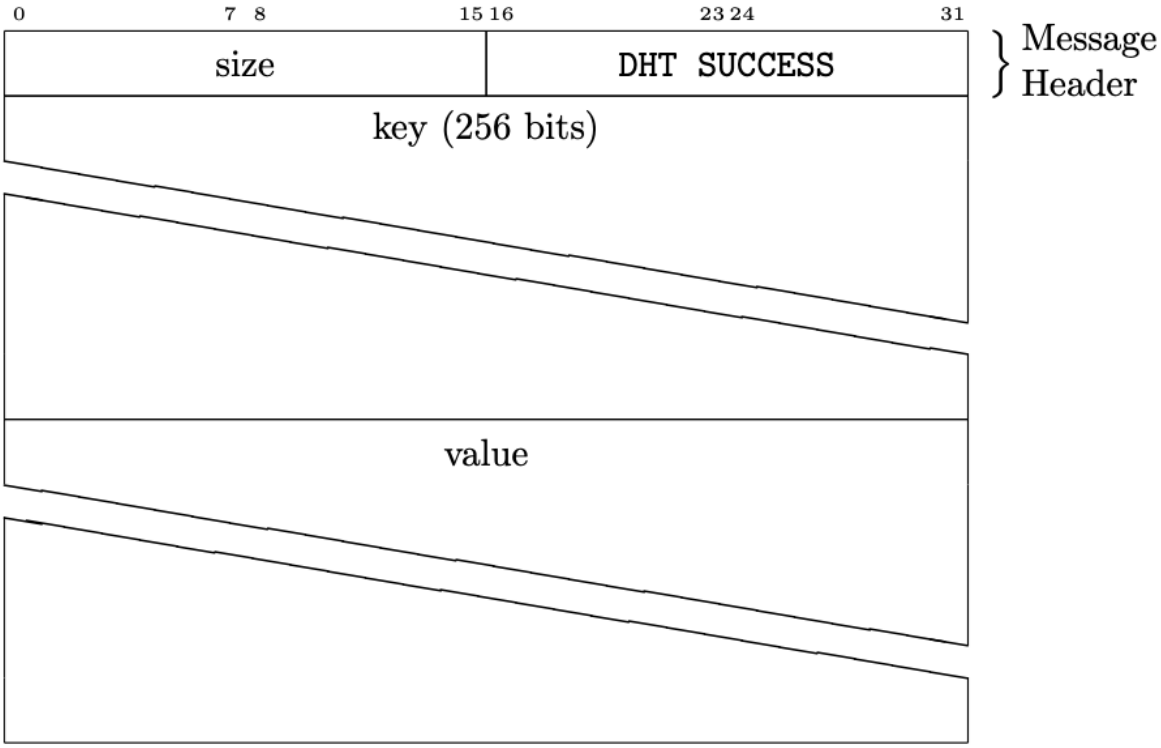
3.2. DHT GET

This message is used to ask the DHT method to search for a given key and provide the corresponding value, if it can be found in the network.



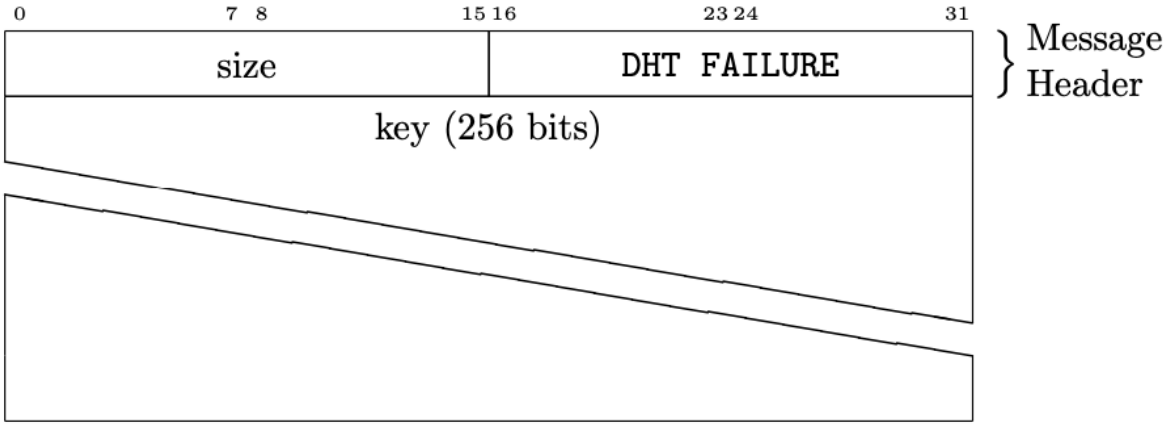
- Size: Size of the body of the of the DHT GET with the message headers
- Type: Type of the message (DHT GET)
- Body: Key of the key-value pair that should be retrieved from the network

3.3. DHT SUCCESS



- Size: Size of the body of the of the DHT SUCCESS with the message headers
- Type: Type of the message (DHT SUCCESS)
- Body: Body of the success message including key-value pair

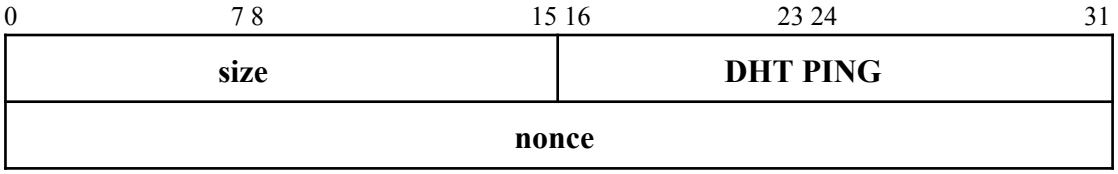
3.4. DHT FAILURE



- Size: Size of the body of the of the DHT FAILURE with the message headers
- Type: Type of the message (DHT FAILURE)
- Body: Body of the error message

3.5. DHT PING

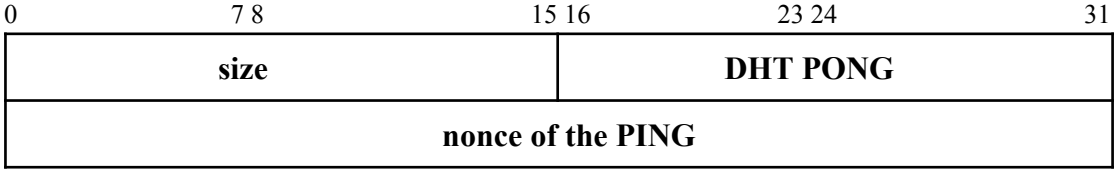
Ping is used to check if the address and the port is running and whether the peer is reachable.



- Size: Size of the body of the of the DHT PING with the message headers
- Type: Type of the message (DHT PING)
- Nonce: Random value

3.6. DHT PONG

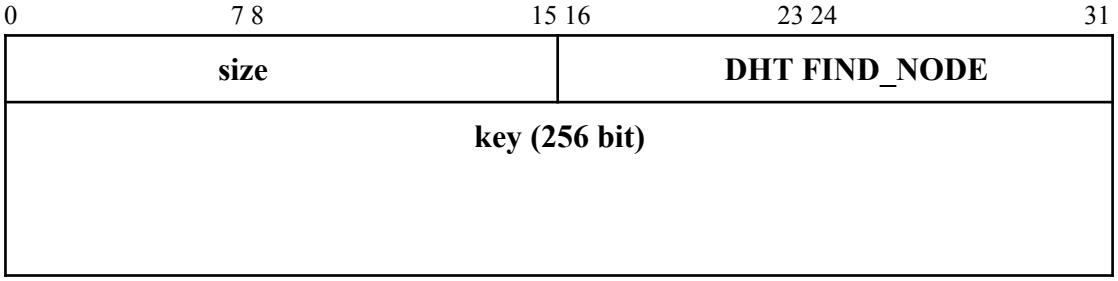
Response to the “PING” message.



- Size: Size of the body of the of the DHT PONG with the message headers
- Type: Type of the message (DHT PONG)
- Nonce: Nonce received from the PING message

3.7. DHT FIND_NODE

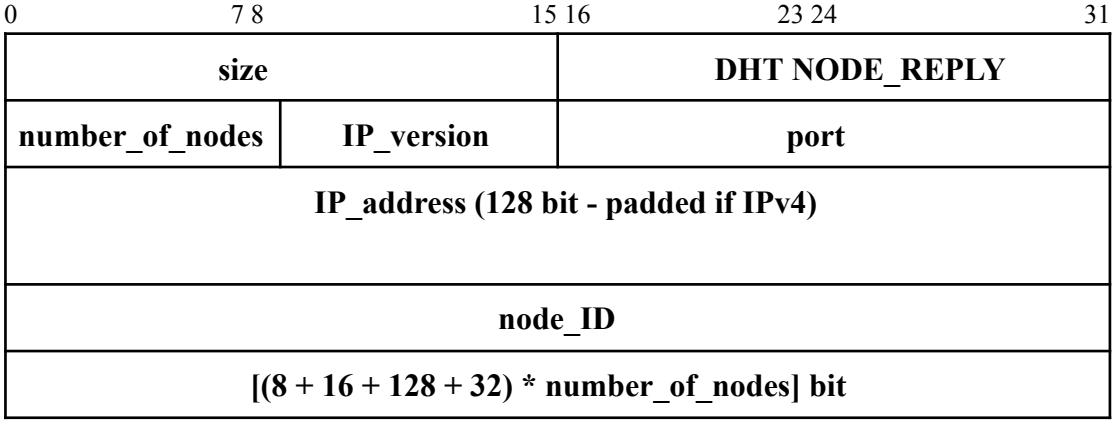
Find node message is used for the node lookups in the network. Sends requests to its non-empty k buckets.



- Size: Size of the body of the of the DHT FIND NODE with the message headers
- Type: Type of the message (DHT FIND NODE)
- Key: ID of the peer that responds with a PONG message

3.8. DHT NODE_REPLY

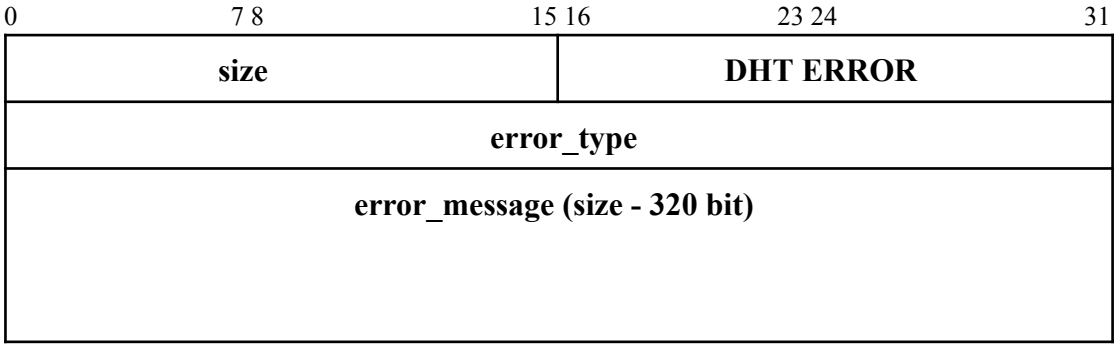
Response for the “FIND_NODE”.



- Size: Size of the body of the of the DHT NODE REPLY with the message headers
- Type: Type of the message (DHT NODE REPLY)
- IP Version: Version of the IP
- Number of Nodes: Count of nodes that Node Reply responded with. It contains information about nodes that matches Find Node message
- Port: Port of the node
- IP Address: IP address of the node
- Node ID: ID of the Node

3.9. DHT ERROR

This message is returned when an error occurs.



- Size: Size of the body of the of the DHT NODE REPLY with the message headers
- Type: Type of the message (DHT NODE REPLY)
- IP Version: Version of the IP
- Port: Port of the node
- IP Address: IP address of the node
- Node ID: ID of the Node

4. Future Work

As we move forward with our project, we have several key areas of focus planned for the next phase of our work.

4.1. Kademlia Implementation

Having conducted extensive research and gained a comprehensive understanding of the Kademlia Distributed Hash Table (DHT) scheme, we are ready to implement it in the next stage of our project.

4.2. Error Handling

Our focus will also extend to strengthening our system's robustness through enhanced error handling. Specifically, we will address issues that could arise from irregularities in messages, such as incorrect sizes or corrupted data. By preempting these potential problems, we aim to build a system that maintains its integrity and performance under varied conditions.

4.3. Extended Testing

While our current implementation has been successful in handling multiple peers and establishing connections, we acknowledge the need for comprehensive testing. In the next phase, we plan to develop and run a wider

array of test cases, enabling us to verify our system's performance under different scenarios and loads.

4.4. Security Measures

The safety of our system and the data it handles is paramount. With this in mind, we intend to enhance our security measures. We will employ SSL certificates and the RSA encryption method, securing our data transmissions and providing a more trustworthy and reliable system.

4.5. Containerization

To improve the portability and scalability of our application, we plan to use Docker for containerization. This will streamline deployment, allow for easier version control, and enhance our ability to run our application in varied environments.

In essence, these upcoming phases of our project will focus on enhancing our system's robustness, security, and adaptability.

5. Workload Distribution

Working together was crucial for us to establish the foundation of the project correctly. We didn't choose to work individually, as there were only two of us. Given that we shared many common courses, it simplified the task of scheduling our collaborative sessions. During our meetings, we dedicated considerable time to brainstorming and understanding the project requirements thoroughly. This helped us to develop a shared vision and a well-defined roadmap for our work. As a team, we divided our responsibilities based on individual strengths, resulting in a more efficient workflow. We believe that this collaborative yet segmented approach substantially improved our productivity and accelerated our progress.

6. Effort Spent

On average, we conducted two Scrum-style meetings per week, each lasting between 2 to 3 hours. These agile-focused meetings facilitated productive discussions and collaborative decision-making. We favored teamwork over individual efforts, often resorting to pair programming. All of our work was consistently committed and pushed to the main branch. We held extensive analysis and brainstorming sessions to comprehend our implementation needs and to decide on an appropriate structure. To maintain progress, we took care to commit changes regularly and document them

comprehensively. Over a span of five weeks since the initial report, we invested a total of 20-25 hours into the project.

7. References

Kademlia: A Peer-to-Peer Information System Based on the XOR Metric, www.scs.stanford.edu/~dm/home/papers/kpos.pdf. Accessed 2 July 2023.

"Kademlia." *Wikipedia*, 20 Apr. 2023, en.wikipedia.org/wiki/Kademlia.