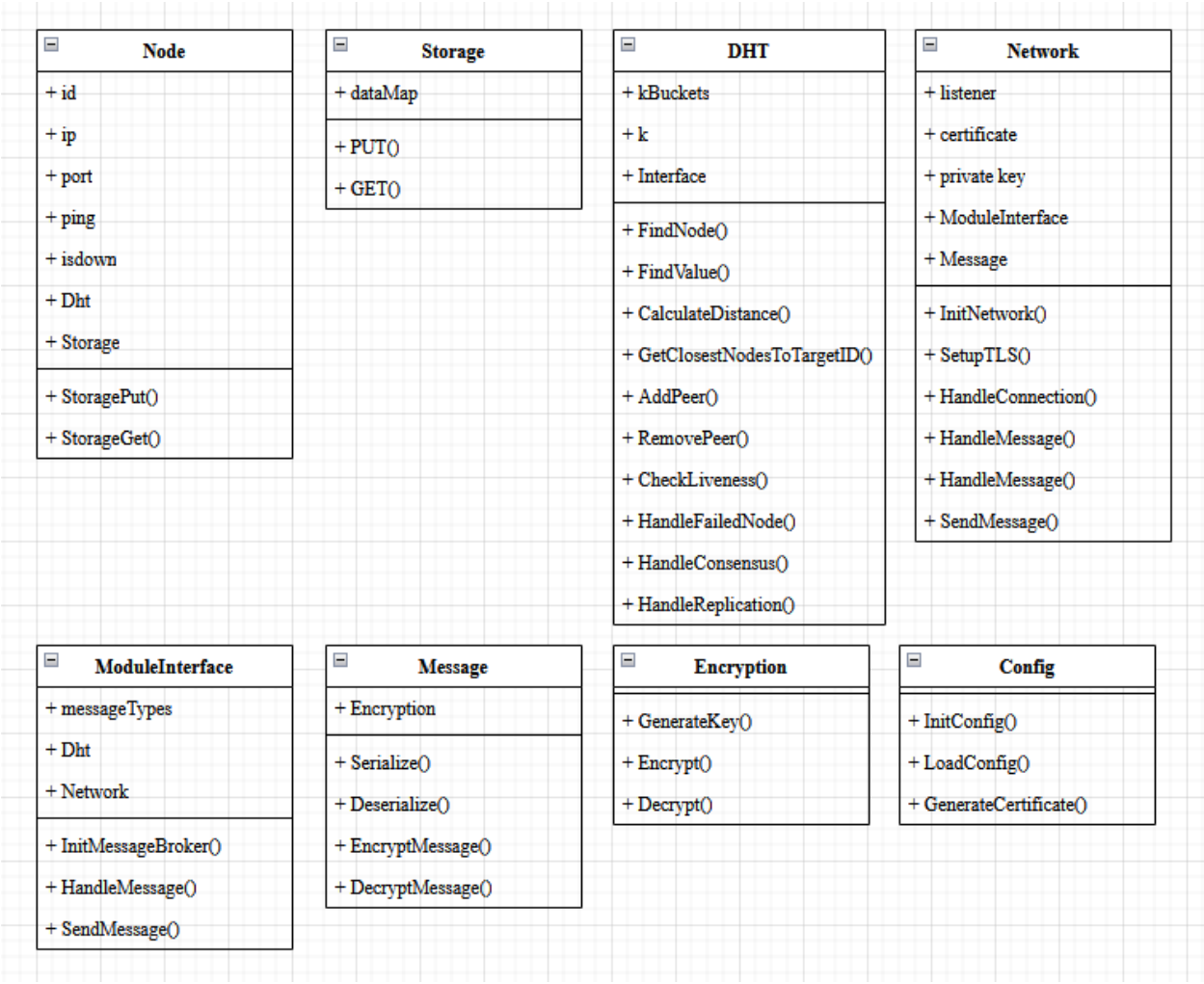
# VoidPhone Project: Midterm Report

Group 14
July 2, 2024

**Muhammed Orhun Gale**
**Kerem Kurnaz**

Group Name: 14
Number of Team Members: 2
Module: DHT
Implementation: Kademlia

## Logical Structure of the DHT Project




The Distributed Hash Table (DHT) project comprises several interrelated components designed to facilitate distributed storage and retrieval of data across a network of nodes. Below is a detailed explanation of the logical structure and its components:

**1. Node**

The Node class represents an individual unit in the network with the following attributes and methods:

- **Attributes:**
  - id: Unique identifier for the node.
  - ip: IP address of the node.
  - port: Communication port for the node.
  - ping: Status check to verify if the node is responsive.
  - isdown: Boolean status indicating if the node is down.
  - Dht: Reference to the DHT component that the node interacts with.
  - Storage: Reference to the storage component associated with the node.
- **Methods:**
  - StoragePut(): Method to store data in the node's storage.
  - StorageGet(): Method to retrieve data from the node's storage.

**2. Storage**

The Storage class manages the key-value data storage for each node:

- **Attributes:**
  - dataMap: Internal data structure (a key-value store) for managing stored data.
- **Methods:**
  - PUT(): Method to insert data into the storage.
  - GET(): Method to retrieve data from the storage.

**3. DHT (Distributed Hash Table)**

The DHT class handles the logic for the distributed hash table overlay:

- **Attributes:**
  - kBuckets: Structure for managing neighbor nodes (Kademlia's k-buckets).
  - k: The number of nodes in each bucket.
  - Interface: Reference for the Connection interface for managing DHT operations.
- **Methods:**
  - FindNode(): Method to locate a node in the DHT (according to Kademlia algorithm).
  - FindValue(): Method to find a value stored in the DHT (according to Kademlia algorithm).
  - CalculateDistance(): Computes the distance metric between a pair of node IDs (XOR distance).

    - GetClosestNodesToTargetID(): Retrieves nodes closest to a specific target ID ().
    - AddPeer(): Adds a new node to the DHT.
    - RemovePeer(): Removes a node from the DHT.
    - CheckLiveness(): Checks if nodes in the DHT are still active.
    - HandleFailedNode(): Manages nodes that have failed.
    - HandleConsensus(): Ensures data consistency and agreement among nodes.
    - HandleReplication(): Manages data replication across nodes for fault tolerance.

**4. Network**

The Network class is responsible for network communication between nodes:

- **Attributes:**
    - listener: Listens for incoming network connections.
    - certificate: Security certificate for encrypted communications.
    - privateKey: Private key used for securing network communications.
    - ModuleInterface: Reference to the Interface component for interacting with other modules.
    - Message: Reference to the Message component for serializing/deserializing network messages.
- **Methods:**
    - InitNetwork(): Initializes the network configuration.
    - SetupTLS(): Sets up Transport Layer Security for secure communications.
    - HandleConnection(): Manages incoming and outgoing network connections.
    - HandleMessage(): Processes received messages from the network.
    - SendMessage(): Sends messages to other nodes over the network.

**5. ModuleInterface**

The ModuleInterface class manages interactions between different modules within the node:

- **Attributes:**
    - messageTypes: Different types of messages that can be processed.
    - Dht: Reference to the DHT component.
    - Network: Reference to the Network component.
- **Methods:**
    - InitMessageBroker(): Initializes the message broker for handling inter-module communication.
    - HandleMessage(): Processes messages received from other modules and creates tasks/operations for other components accordingly.
    - SendMessage(): Sends messages to other modules or nodes (sends the result of tasks as a response).

**6. Message**

The Message class handles the formatting and processing of messages exchanged between nodes:

- **Attributes:**
  - Encryption: Reference to the encryption component for securing messages.
- **Methods:**
  - Serialize(): Converts a message into a format suitable for transmission.
  - Deserialize(): Converts received data into a message format.
  - EncryptMessage(): Encrypts a message for secure transmission using the Encryption class.
  - DecryptMessage(): Decrypts a received message using the Encryption class.

**7. Encryption**

The Encryption class provides cryptographic functions for securing data and communications:

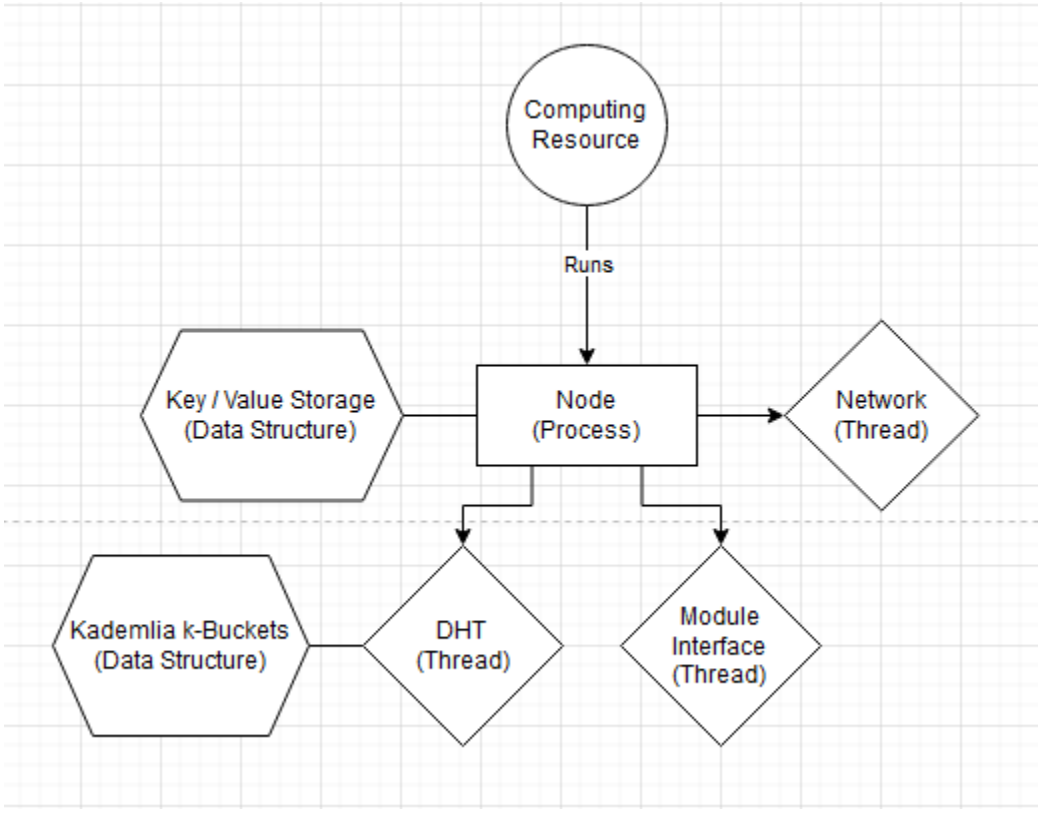
- **Methods:**
  - GenerateKey(): Generates cryptographic keys for encryption and decryption.
  - Encrypt(): Encrypts data using a specified key.
  - Decrypt(): Decrypts data using a specified key.

**8. Config**

The Config class manages configuration settings for the DHT and its components:

- **Methods:**
  - InitConfig(): Initializes configuration parameters.
  - LoadConfig(): Loads configuration from a file or another source.
  - GenerateCertificate(): Generates security certificates for encrypted communications.

**Process Architecture of the DHT Project**




The DHT system operates as a collection of a process and threads, each responsible for handling specific tasks. At the heart of the process architecture is the Node process. Each node represents an independent entity in the network, and it runs as a separate process. This design encapsulates the node's activities and allows for parallel operation with other nodes. The Node process is responsible for managing its own key-value data storage, DHT operations for peers, and network communications. It interacts with various threads to perform its functions efficiently.

**Components and Threads**

1. **Key/Value Storage (Data Structure)**
    - This component is responsible for managing the actual data that the node stores. It implements a key-value storage system where data can be put and retrieved.
    - This structure is directly accessed by the Storage class for data operations.
2. **Kademlia k-Buckets (Data Structure)**
    - K-Buckets are a part of the Kademlia protocol implemented in the DHT system. They are used to manage information about neighboring nodes in the network.
    - Each node maintains several k-buckets, which store identifying information (IDs) about other nodes, facilitating efficient routing and lookup operations in the DHT.

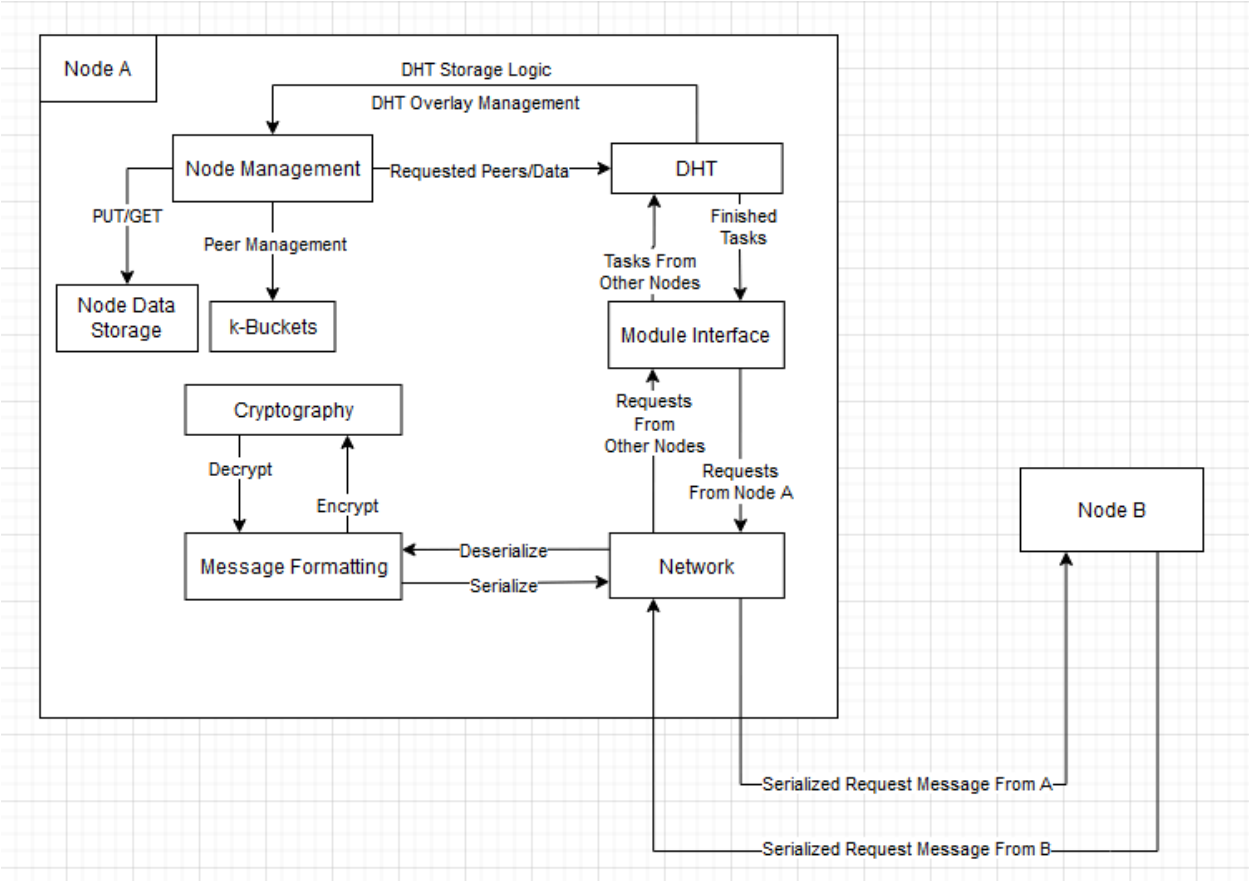
3. **DHT Thread**
    - The DHT thread is dedicated to handling DHT-specific tasks such as peer discovery, data lookup, and node management. It ensures that the node can efficiently find and interact with other nodes in the distributed network.
    - This thread manages the kBuckets and coordinates with other nodes to maintain the integrity and performance of the DHT overlay network.
4. **Network Thread**
    - The Network thread is tasked with managing all network communications, ensuring that the node can communicate with other nodes, send and receive data, and maintain network stability. Such as handling incoming and outgoing connections, message serialization and deserialization, and secure data transmission.
5. **Module Interface Thread**
    - The ModuleInterface thread serves as a bridge between the DHT and Network threads, it handles internal messaging and task coordination between different modules, ensuring that the node operates cohesively.

**Execution Flow**

1. **Initialization:**
    - The Node process is launched, initializing its storage, DHT, and network components.
    - It then starts the DHT, Network, and Module Interface threads, setting up the environment for distributed operations.
2. **Node Operations:**
    - The DHT thread continuously manages peer discovery, key-value data lookup, and maintains the integrity of the DHT network.
    - The Network thread listens for incoming connections, manages outgoing messages, and ensures secure communication through encryption.
    - The Module Interface thread coordinates the activities of other threads, ensuring seamless interaction between the DHT logic and network operations.
3. **Data Storage and Retrieval:**
    - Data storage and retrieval operations are handled by the Storage class, which interacts with the key-value storage system and uses the DHT thread for locating data across the network.
    - The DHT thread uses k-buckets to efficiently locate nodes and data, leveraging the structure of the DHT for optimal performance.

4. **Communication:**
    - Network messages are formatted, serialized, and transmitted by the Network thread. To ensure security, uses encryption and TLS.
    - The Message class handles the formatting, serialization, and encryption of messages, providing secure and reliable communication between nodes.
5. **Fault Tolerance and Replication:**
    - The DHT thread monitors node availability and handles replication of data to ensure fault tolerance.
    - Nodes regularly check the liveness of their peers and replicate data to maintain consistency and reliability in the network.

**System Architecture of the DHT Project**




The system architecture diagram for the DHT project shows the internal components of a single node (Node A) its interactions with other nodes (such as Node B) and how the internal components interact with each other. Each node in the network performs specific roles including data storage, peer management, message handling, and network communication.

**Key Components and Their Roles**

1. **Node Data Storage**
    - **Purpose:** Manages the local storage of data within each node.
    - **Functionality:** Handles PUT and GET operations, allowing data to be stored and retrieved using a key-value storage mechanism.
    - **Interaction:** Communicates directly with the Node Management component to handle data operations requested by the DHT.
2. **Node Management**
    - **Purpose:** Oversees the overall management and health of the node.
    - **Functionality:** Manages peer relationships, maintains the k-buckets for routing efficiency, and coordinates data storage requests.
    - **Interaction:** Works closely with the DHT, K-Buckets, and Storage components to manage node operations and ensure data is properly stored and accessible.
3. **K-Buckets**
    - **Purpose:** Implements the Kademlia protocol for efficient node lookup and routing.
    - **Functionality:** Maintains lists of peer nodes organized by their distance from the current node, facilitating quick lookup and routing of messages within the DHT.
    - **Interaction:** Integrated with the DHT component to locate nodes and route messages effectively.
4. **Cryptography**
    - **Purpose:** Ensures the security of data and communications within the network.
    - **Functionality:** Encrypts and decrypts messages and data to maintain confidentiality and integrity during storage and transmission.
    - **Interaction:** Interacts with the Message Formatting component to secure messages before they are serialized and sent over the network.
5. **Message Formatting**
    - **Purpose:** Manages the formatting and processing of messages for network communication.
    - **Functionality:** Serializes messages for transmission and deserializes received messages, ensuring they are in the correct format for processing by the node.
    - **Interaction:** Works with the Network component to handle incoming and outgoing messages, and uses Cryptography for securing these messages.
6. **Network**
    - **Purpose:** Facilitates communication between nodes in the network.
    - **Functionality:** Manages network connections, handles the transmission and reception of serialized messages, and ensures secure communication using TLS.
    - **Interaction:** Directly interfaces with other nodes (e.g., Node B) to send and receive messages. It uses the Message Formatting component to process these messages.

7. **Module Interface**
    - **Purpose:** Acts as a coordinator for tasks and messages within the node.
    - **Functionality:** Processes requests from both internal components and external nodes, and routes them to the appropriate handlers within the node.
    - **Interaction:** Communicates with the DHT, Network, and other internal modules to manage requests and ensure tasks are executed correctly.
8. **DHT (Distributed Hash Table)**
    - **Purpose:** Manages the DHT logic and operations.
    - **Functionality:** Handles tasks related to peer discovery, data lookup, and overall maintenance of the DHT overlay network.
    - **Interaction:** Interfaces with the Node Management to process requests for peers and data. It also coordinates with the Module Interface to handle tasks from other nodes and the K-Buckets to find the closest nodes to a target ID.

**Interaction Flow**

The interaction flow between these components can be understood through the following processes:

1. **Data Storage and Retrieval:**
    - The Node Management component receives a PUT/GET request.
    - The request is processed and routed to Node Data Storage for local operations or to the DHT for distributed operations.
    - The DHT uses K-Buckets to locate the target node if the data is stored elsewhere in the network.
    - Data is retrieved or stored, and the response is returned to the same path.
2. **Peer Management:**
    - The Node Management component continuously maintains peer relationships, using the DHT and K-Buckets to manage the list of neighbors.
    - The DHT monitors the health and status of peers, using the CheckLiveness method to ensure peers are active.
3. **Message Handling:**
    - Incoming messages are received by the Network component, deserialized by Message Formatting, and decrypted by Cryptography.
    - The Module Interface processes these messages, directing them to the appropriate components (e.g., DHT for data requests or Node Management for peer updates).
    - Outgoing messages follow the reverse path: they are serialized by Message Formatting, encrypted by Cryptography, and transmitted by the Network.

4. **Network Communication:**
    - The Network component handles all aspects of network communication, including setting up secure connections with TLS and transmitting serialized messages between nodes.
    - It ensures that messages are delivered to the correct recipient node (e.g., Node B) and manages the response back to the sender.
5. **Security:**
    - The Cryptography component secures all data and messages handled by the node, ensuring that they are encrypted before transmission and decrypted upon receipt.
    - This component is critical for maintaining the confidentiality and integrity of data in the DHT network.

**Security Measures**

**Message Encryption Algorithms (Implemented)**

- **TLS (Transport Layer Security):**
    - **Implementation:** TLS is implemented using certificates and private keys. The sender node encrypts the message using their private key then, the receiving node must be able to decrypt the certificate using their own key.
    - **Security Benefits:** TLS ensures the authenticity of the nodes involved in the communication and provides a secure channel between nodes, preventing eavesdropping and man-in-the-middle attacks.
- **AES-256-GCM Encryption:**
    - **Implementation:** We have implemented AES-256-GCM (Advanced Encryption Standard with 256-bit keys in Galois/Counter Mode) for message encryption. The messages are encrypted before sending to another node and, the data is encrypted before storing within the node.
    - **Security Benefits:** In addition to TLS, message encryption makes the data transmission end-to-end encrypted. End-to-end encryption prevents insider threats from gaining access to unauthorized data.

**Flood Prevention**

**Rate-Limiting Algorithms:**

- **Fixed Window Counter:**
    - **Implementation:** This method counts the number of requests from a node in a fixed time window and blocks requests if they exceed a predefined limit.
    - **Security Benefits:** It helps in preventing denial-of-service (DoS) attacks by limiting the number of requests a node can make in a given time frame.

- **Token Bucket:**
  - **Implementation:** This method uses tokens to regulate the rate of requests. Tokens are added to the bucket at a fixed rate, and a request is only processed if there are enough tokens available.
  - **Security Benefits:** The token bucket algorithm is more flexible and can handle burst traffic more effectively, it provides a balanced approach to rate-limiting.

**Data Redundancy and Consensus Algorithms**

1. **Consensus Algorithms:**
   - **Quorum:**
     - **Implementation:** In a quorum-based consensus, a majority of nodes must agree on any changes to the data.
     - **Security Benefits:** Quorum prevents malicious nodes from altering data without the agreement of the majority, thus ensuring data integrity.
   - **Proof of Work (PoW):**
     - **Implementation:** PoW requires nodes to perform computational work to propose a change in the network.
     - **Security Benefits:** PoW makes it computationally difficult for attackers to alter data or disrupt the network using spamming attacks, providing a higher level of security against attacks and achieving consensus.
2. **Data Replication for Redundancy:**
   - **Implementation:** Data is replicated across multiple nodes to ensure its availability and durability.
   - **Security Benefits:** Data replication ensures that even if some nodes fail or are compromised, the data remains accessible and intact across the network.

**Secure Configuration (Implemented)**

- **Implementation:** Certificates are generated securely with a Python script before the node initializes.
- **Security Benefits:** Secure configuration management ensures that the system operates as intended and reduces the risk of misconfigurations that could lead to vulnerabilities.

Currently, message encryption algorithms (TLS, AES-256) and secure configuration measures are implemented in our mock DHT module. The remaining security measures will be selected and implemented in the final project.

**Specification of the Peer-to-Peer Protocol(s)**

**1. GET Message**

Purpose: To request the value associated with a given key.
Message Format:

- size: The total size of the message.
- GET: The type of the message indicating it is a GET request.
- key: The 256-bit key for which the value is being requested.

Reasoning: The GET message is needed for retrieving stored data in the DHT. Nodes use this message to ask other nodes for specific key-value pairs.

**2. PUT Message**

Purpose: To store or update a value associated with a given key.
Message Format:

- size: The total size of the message.
- PUT: The type of the message indicating it is a PUT request.
- key: The 256-bit key for the data.
- value: The value to be stored or updated.

Reasoning: The PUT message allows nodes to add or modify data in the DHT. This is needed for maintaining and updating the distributed hash table.

**3. SUCCESS Message**

Purpose: To notify that a key/value operation (like GET or PUT) succeeded.
Message Format:

- size: The total size of the message.
- SUCCESS: The type of the message indicating a successful operation.
- key: The 256-bit key involved in the operation.
- value (optional): The value associated with the key if applicable.

Reasoning: This message provides confirmation that an operation completed successfully, helping nodes to verify that their requests have been fulfilled.

**4. FAILURE Message**

Purpose: To notify that a key/value operation (like GET or PUT) failed.
Message Format:

- size: The total size of the message.
- FAILURE: The type of the message indicating a failed operation.
- key: The 256-bit key involved in the operation.

Reasoning: This message informs nodes when an operation cannot be completed, enabling them to handle errors.

**5. PING Message**

Purpose: To check if a node is alive and responsive.
Message Format:

- size: The total size of the message.
- PING: The type of the message indicating a ping request.
- sender_id: The 256-bit identifier of the node sending the ping.

Reasoning: PING messages are used to ensure the liveness of nodes in the network. They help to maintain a healthy and a reliable network by seeing which nodes are available.

**6. PONG Message**

Purpose: To respond to a PING message and confirm that a node is alive.
Message Format:

- size: The total size of the message.
- PONG: The type of the message indicating a pong response.
- sender_id: ID of the node sending the pong.

Reasoning: PONG messages are sent in response to PING messages to confirm node availability, which supports the network's health and reliability.

**7. FIND_NODE Message**

Purpose: To request the location of a specific node in the network.
Message Format:

- size: The total size of the message.
- FIND_NODE: The type of the message indicating a request to find a node.
- target_id: ID of the target node being searched for.

Reasoning: This message helps in locating other nodes in the DHT, which is essential for routing queries and maintaining the network structure.

**8. FIND_VALUE Message**

Purpose: To request the location of a node holding a specific value.
Message Format:

- size: The total size of the message.
- FIND_VALUE: The type of the message indicating a request to find a value.
- key: The key associated with the value being searched for.

Reasoning: FIND_VALUE messages are used to locate the nodes storing a specific value, facilitating data retrieval within the DHT.

**9. FOUND_NODE Message**

Purpose: To reply with the information of a found node.
Message Format:

- size: The total size of the message.
- FOUND_NODE: The type of the message indicating a reply with node information.

- node_id: ID of the node being reported.
- node_address: The network address (e.g., IP and port) of the node.

Reasoning: This message provides the requester with the details of a node that has been located, enabling communication and further interactions.

**Future Work**

- Improve and prepare tests for the Kademlia logic.
- Update the current mock DHT module, to be a true decentralized peer-to-peer application. The current implementation's tests are done in a single DHT instance, for a more convenient testing environment.
- Implement the ping and pong logic.
- Implement the Module Interface component. In the current version of the module, the DHT component handles processing messages and acting upon the requests. In the final version, the module interface will be responsible for analyzing the deserialized messages and request operations from the necessary components (DHT, Node Management etc).
- Implement the message formats for specifying the peer-to-peer protocol.
- Implement the missing (but desired) security measures as stated in the Security Measures section.
- Dockerization of the module
- Check if Kubernetes and Terraform integrations are feasible for the project.

## 7. Effort Spent for the Project

**Both Teammates:** We collaboratively designed the logical structure, process architecture, and system architecture. Additionally, we integrated the specified security measures into the system.

**For Orhun:** Orhun focused on the development and communication of the Node, DHT, and Network modules along with their related components. He prepared unit and integration tests, established the test/module build system (makefile), and implemented CI/CD integration.

**For Kerem:** Kerem concentrated on the Kademlia logic within the DHT module and prepared unit and integration tests for this component. He was also responsible for all reporting tasks.