TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

A

PROJECT REPORT

ON

P2POCKET

SUBMITTED BY:

MANDIP THAPA (PUL077BCT044)

MANISH KUNWAR (PUL077BCT045)

SUBMITTED TO:

DEPARTMENT OF ELECTRONICS & COMPUTER ENGINEERING

March, 2024

Copyright

Acknowledgments

This project is being undertaken as a course requirement of the Minor Project, as a part of the third year curriculum of Bachelor in Computer Engineering (BCT), IOE.

Our sincere thanks to Department of Electronics and Computer Engineering(DoECE) for providing us the inspiration and guidance for this project.

We would like to express our very great appreciation to Assistant Professor Dr. Babu R. Dawadi for his valuable suggestions during the feasibility study of the project. His willingness to give his time so generously has been very much appreciated.

We'd like to express our deep gratitude to our cluster leader, Associate Professor Daya Sagar Baral for helping us in idea brainstorming and initial draft.

We would also like to extend our thanks to Assistant Professor Santosh Giri for his guidance and support in preparing this proposal.

We would also like to express our deep appreciation and gratitude to our friends for providing feedback and suggestions while selecting the project topic.

Abstract

In a landscape where traditional storage options are constrained and cloud storage solutions are not always desired, P2Pocket is an attempt to develop a system on top of Kademlia and P2P networking where users in the network can collaboratively share their available storage resources and tap into the collective capacities of the network. Inspired heavily by BitTorrent, this project seeks to be an alternative to the way individuals manage digital storage. This project marks the culmination of three years of our engineering study deriving knowledge from subjects like distributed system and computer networks. Therefore, this project serves both facets of academia and practical implementation. This project introduces a command line interface for file storage and retrieval operations within the network. therefore, it facilitates collaborative sharing while utilizing collective network capacities.

Keywords: *Networking, Peer-to-Peer System, Overlay Networking, Decentralized, Distributed Hash Table, Kademlia, Bootstrap Server, Storage Sharing*

Contents

List of Figures

List of Tables

List of Abbreviations

P2P	Peer to Peer
DHT	Distributed Hash Table
IPFS	InterPlanetary File System
NAT	Network Address Translation
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
IP	Internet Protocol

1. Introduction

1.1 Background

In today's increasingly digital world, the demand for storage space has surged dramatically. Whether it's storing photos, videos, documents, or presentations, almost every aspect of our lives requires a significant amount of digital storage. However, the storage capacity of our personal devices is limited, leading many individuals and businesses to seek additional storage solutions.

Cloud storage has emerged as a popular option to address this need. However, it often comes with drawbacks such as subscription costs and concerns about privacy and security. As a result, there's a growing interest in alternative approaches to storage.

P2Pocket is a conceptualized P2P storage sharing system, where users share their available storage to the network and uses storage resources of the network in need. The project is heavily inspired from BitTorrent, a P2P file sharing system. The proposed system comes with its own set of challenges and may not be an appropriate alternative to the existing solutions. Therefore, through this project, we only aim to explore the practicality of P2P system to address the ever growing need for supplementary storage.

1.2 Problem statements

The prevalent file storage systems in use today rely heavily on centralized architectures, wherein data is stored on a single server. However, this approach poses several limitations. Firstly, scalability becomes an issue as centralized systems struggle to handle increasing data volumes and user demands. Maintenance also becomes more complex, with any issues affecting the central server potentially impacting the entire system. Furthermore, the system's dependence on a single point of storage leaves it open to several kinds of failures, including those brought on by hardware issues, cyberattacks, and other unanticipated circumstances.

Additionally, centralized systems can become bottlenecks under heavy loads, leading to delays and reduced performance. Their inability to adapt quickly to changing user demands exacerbates these issues, as they may not efficiently allocate resources where needed most.

Privacy and security are significant concerns in centralized systems as well. With all data stored in one location, there's an increased risk of unauthorized access or data breaches. This becomes especially worrisome considering the potential misuse of personal data, such as images, which could be utilized without consent for training artificial intelligence systems, raising ethical and privacy concerns.

1.3 Objectives

1. To develop a peer-to-peer (P2P) system to maximize network storage resource utilization.

2. To establish a robust framework emphasizing security and privacy for storage sharing among network nodes.

3. To implement decentralized storage distribution to reduce dependency on centralized servers while ensuring data integrity and confidentiality.

1.4 Scope

The system can serve as a practical alternative to traditional cloud and secondary storage solutions, particularly for individuals or organizations requiring additional storage capacity. By leveraging decentralized architecture, it provides a reliable and scalable platform to meet diverse storage needs efficiently. This project endeavors to evaluate the feasibility and effectiveness of decentralized solutions across various domains, including but not limited to file sharing, content delivery, and communication protocols.

Furthermore, beyond its immediate application in addressing practical storage requirements, this project also aims to contribute to academic research by deepening our comprehension of peer-to-peer (P2P) and decentralized systems. Through rigorous analysis and experimentation, it aims to uncover insights into the underlying principles, challenges, and potentials of decentralized technologies. By advancing our understanding in this field, the project endeavors to pave the way for future innovations and developments in decentralized systems, ultimately contributing to the broader body of knowledge within the academic community.

2. Literature Review

2.1 Related work

BitTorrent[1], also referred to as simply torrent, is a communication protocol for P2P file sharing, which enables users to distribute data and files over the Internet in a decentralized manner. To download a file, users download a .torrent file that contains file metadata, hash and a list of network locations of trackers, that helps participants in the system form efficient distribution groups, called swarms. The hash is used to identify the torrents to other peers and to the trackers. Normally, a BitTorrent client is used to facilitate the download and sharing of files between peers.

Kademlia[2], a distributed hash table for decentralized P2P computer networks.It specifies the structure of the network and the exchange of information through node lookups. Kademlia nodes communicate among themselves using UDP. A virtual or overlay network is formed by the participant nodes. Each node is identified by a number or node ID. The node ID serves not only as identification, but the Kademlia algorithm uses the node ID to locate values.

IPFS[3], is a decentralized protocol revolutionizing data storage and sharing by employing a peer-to-peer network model where files are distributed across nodes rather than centralized servers, each file uniquely identified by a cryptographic hash. This content-addressing system ensures efficient retrieval and resilience against censorship and network failures. IPFS offers advantages such as improved scalability and potentially lower costs for data storage and distribution, with applications spanning decentralized web infrastructure, content delivery networks, and distributed file sharing

Filecoin[4], is an open-source, public cryptocurrency and digital payment system intended to be a blockchain-based cooperative digital storage and data retrieval method. It is made by Protocol Labs and shares some ideas from InterPlanetary File System allowing users to rent unused hard drive space. A blockchain mechanism is used to register the deals. Filecoin is an open protocol and backed by a blockchain that records commitments made by the network's participants, with transactions made using FIL, the blockchain's native currency. The blockchain is based on both proof-of-replication and proof-of-spacetime.

2.2 Related theory

2.2.1 Peer to Peer System

Peer-to-peer system is a distributed system architecture that partitions resources and workloads between peers. In such systems, peers make a portion of their resources, such as processing power, disk storage or network bandwidth, directly available to other network participants.[5]

Peers are both consumers and producers of resources, in contrast to the traditional client-server architecture in which the consumption and supply of resources are divided. Peer-to-peer networks typically establish a virtual overlay network that operates atop the physical network topology, with the overlay nodes comprising a subset of those in the physical network. While data transmission occurs directly over the underlying TCP/IP network, communication at the application layer allows peers to interact directly through logical overlay links, each aligning with a path within the underlying physical network.

2.2.2 Distributed Hash Table

A Distributed Hash Table (DHT) is a distributed data structure that provides a scalable and decentralized mechanism for mapping keys to values across a network of nodes. It operates by partitioning the keyspace into small, contiguous ranges, and each node in the network is responsible for a specific range of keys. The assignment of ranges to nodes is typically determined using a consistent hashing algorithm, ensuring a relatively uniform distribution of keys among nodes.[6]
DHTs support two fundamental operations: (1) storing a key-value pair in the network and (2) looking up the value associated with a given key. To achieve this, DHTs employ efficient routing algorithms that allow nodes to locate the responsible node for a specific key in a logarithmic number of hops, regardless of the size of the network. Common DHT features include fault tolerance and resilience to node joins and departures. Nodes periodically exchange routing information to adapt to changes in the network topology and ensure accurate key-to-node mappings.

2.2.3 Kademlia

Kademlia is a distributed hash table (DHT) algorithm designed for decentralized peer-to-peer networks. Its core mechanism revolves around organizing participating nodes into a binary tree-like structure, known as k-buckets or routing tables. Each node in the network is identified by a unique identifier, typically a large random number. The key feature of Kademlia is its use of the XOR metric to measure the distance between node identifiers.

This metric enables efficient routing and lookup operations by ensuring that nodes with similar IDs are likely to be neighbors in the network.[2]
When a node wants to find a value associated with a specific key, it initiates an iterative lookup process based on the XOR metric. During each iteration, the node contacts other nodes in the network that are closer to the target key, gradually moving towards the desired value. Data storage in Kademlia is achieved through replication, where each key-value pair is stored on multiple nodes determined by their XOR distance from the key. This redundancy ensures fault tolerance and resilience to node failures, making Kademlia suitable for decentralized applications requiring scalable and fault-tolerant data storage and retrieval.

2.2.4 Overlay Network

An overlay network is a virtual network that is built on top of an existing network infrastructure, providing an additional layer of abstraction and functionality. Overlay networks are typically created to enable specific communication patterns, services, or functionalities that may not be directly supported or efficiently implemented within the underlying network. These overlays are often used in distributed systems, peer-to-peer networks, and cloud computing environments.[7]
In an overlay network, nodes (devices or software instances) establish connections or links with each other independently of the physical network topology. These connections form a logical structure that overlays the physical network, allowing nodes to communicate and collaborate based on the rules and protocols defined by the overlay. Overlay networks can be structured in various ways, such as tree structures, meshes, or distributed hash tables (DHTs), depending on the specific requirements of the application or system.

2.2.5 Hole Punching

Hole punching is a technique used in P2P networking to establish direct communication between two peers (computers or devices) located behind network address translation (NAT) devices or firewalls. In a P2P network, when two peers want to communicate directly, the presence of NATs and firewalls can pose a challenge because they typically prevent incoming connections.[8]

In the hole punching process of peer-to-peer networking, an initiator sends a connection request to a recipient. The recipient's NAT creates a temporary mapping but doesn't forward the request due to firewall or NAT restrictions. Both peers independently send packets to a third-party server, often a rendezvous server, which helps them discover external IP addresses and ports. The server then communicates this information back to the peers, enabling direct communication. By utilizing temporary openings created by NATs or firewalls, hole punching allows for peer-to-peer communication in situations where direct connections would normally be hindered.

3. Methodology

3.1 Feasibility Study

This feasibility study explores the prospects of developing a P2P storage sharing system with mutual involvement. This study provides a comprehensive understanding of the project's potential, addressing challenges, and providing insights crucial for informed decision-making and successful implementation.

3.1.1 Technical Feasibility

Network protocols(the major hindrances in many projects) that shape the connection of the multiple devices support the algorithms that are needed to be implemented in order to achieve the goal of this project. Thus, the project is technically feasible.

3.1.2 Time Feasibility

The time frame of nearly 2 month is presented to us for the completion of project which is plenty. Thus, the project is completely time feasible.

3.1.3 Economic Feasibility

Our own personal computers are to be used to implement and examine the intended outcome of the project, with the help of some friends, resulting in almost no financial outlay. This shows that the project is economically feasible.

3.2 Requirement Specification

3.2.1 Functional Requirements

- A node should be able to join the network by providing its address.
- A user should be able to store the files in the network.
- A user should be able to retrieve the stored file in the network.
- A user should be able to view the list of files and folders stored in the network as a separate filesystem.
- A user should be able to view the status of the file pieces stored in other nodes.

3.2.2 Non-functional Requirements

- The system should be scalable in terms of number of nodes in the network.
- The data should be encrypted before storing in other nodes.
- Only optimum P2P connections should be made to prevent TCP congestion.
- The software should target all available operating system available in the market.
- The user interface should be responsive, user-friendly and facilitate ease of navigation and efficient interaction for all users.

3.3 Description of the working principles

3.3.1 Network Message format

A specified message format is used for communication between nodes. First, four bytes of the message include the length of the total message. This allows the receiving node to read a fixed amount of bytes from the UDP connection object. The next 4 bytes of the message are the type of message that is transmitted. The next 20 bytes contain the node ID of sender. Then, the remaining bytes contain the content of message.

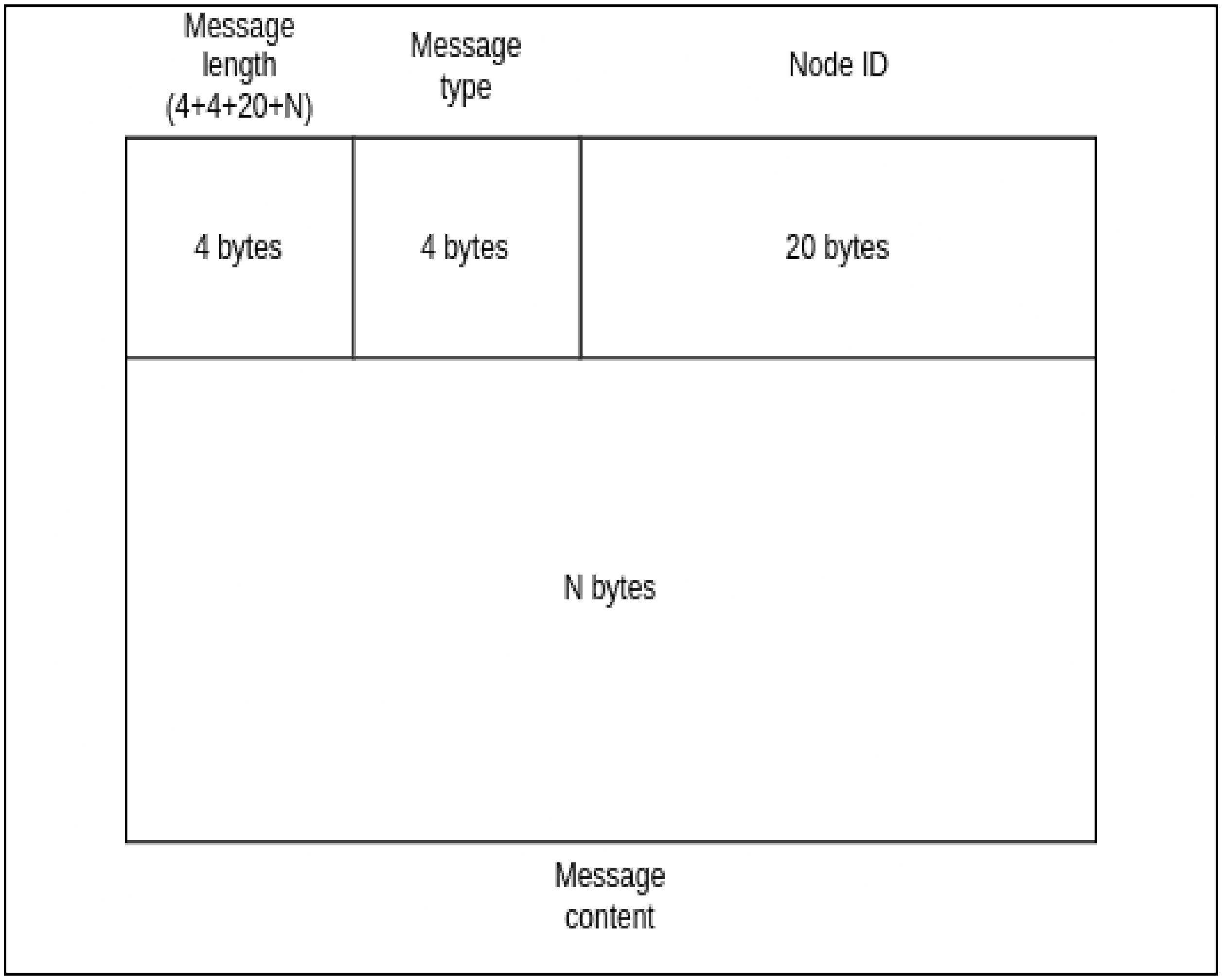


Figure 3.1: Network message format

PING	Simple data packet
STORE	The <key,value>pair
STORE_RESPONSE	Success if it stored the value else failure
FIND_NODE	ID of node to search
FIND_NODE_RESPONSE	<ID,IP,PORT>of search ID or k-bucket with shortest path
FIND_VALUE	Key of value to search
FIND_VALUE_RESPONSE	Value of search Key or k-bucket with shortest path

Table 3.1: Message header: Type field and their corresponding content

3.3.2 Joining the network

Whenever a new node wants to join the p2p network, it contacts with a bootstrap node that provides initial configuration to successfully to the new node. In our case, a bootstrap node is identified known static IP address of the bootstrap node.

The joining node inserts the bootstrap node into one of its k-buckets. The joining node then performs a node lookup of its own ID against the bootstrap node (the only other node it knows). The "self-lookup" will populate other nodes' k-buckets with the new node ID, and will populate the joining node's k-buckets with the nodes in the path between it and the bootstrap node.

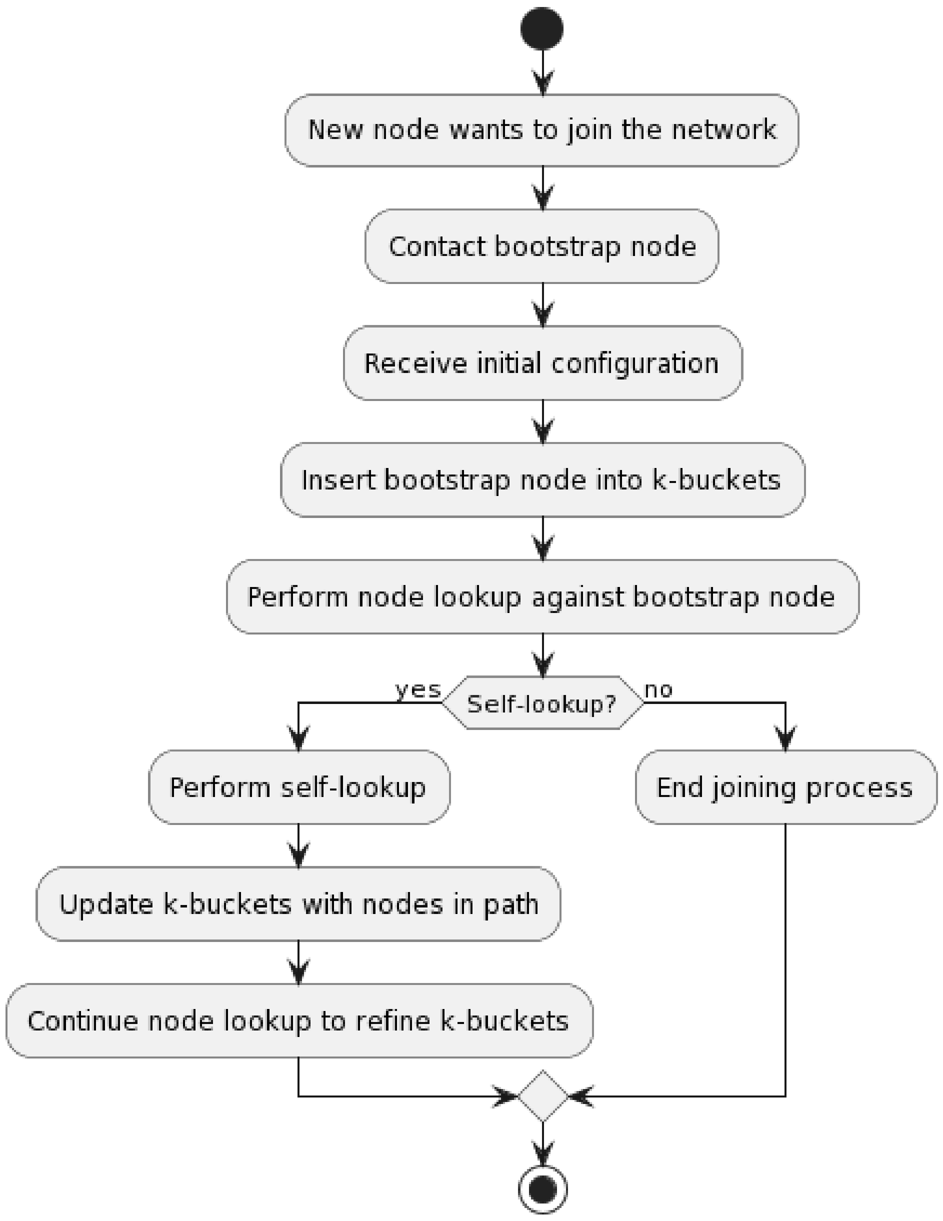

New node wants to join the network
Contact bootstrap node
Receive initial configuration
Insert bootstrap node into k-buckets
Perform node lookup against bootstrap node
Self-lookup?
yes
no
Perform self-lookup
End joining process
Update k-buckets with nodes in path
Continue node lookup to refine k-buckets

Figure 3.2: Joining the network

3.3.3 Locating Nodes

For node lookup, a node initiates a FIND_NODE request to α nodes simultaneously. The α nodes are closest ones to the desired one present in the requesting node's k-buckets. The requested nodes returns response with the k nearest nodes to the desired node that they know. The requesting node then selects the best k results and the process is repeated until the desired node is found. Because every node has a better knowledge of its own surroundings than any other node has, the received results in each iteration will be other nodes that are closer and closer to the searched key.

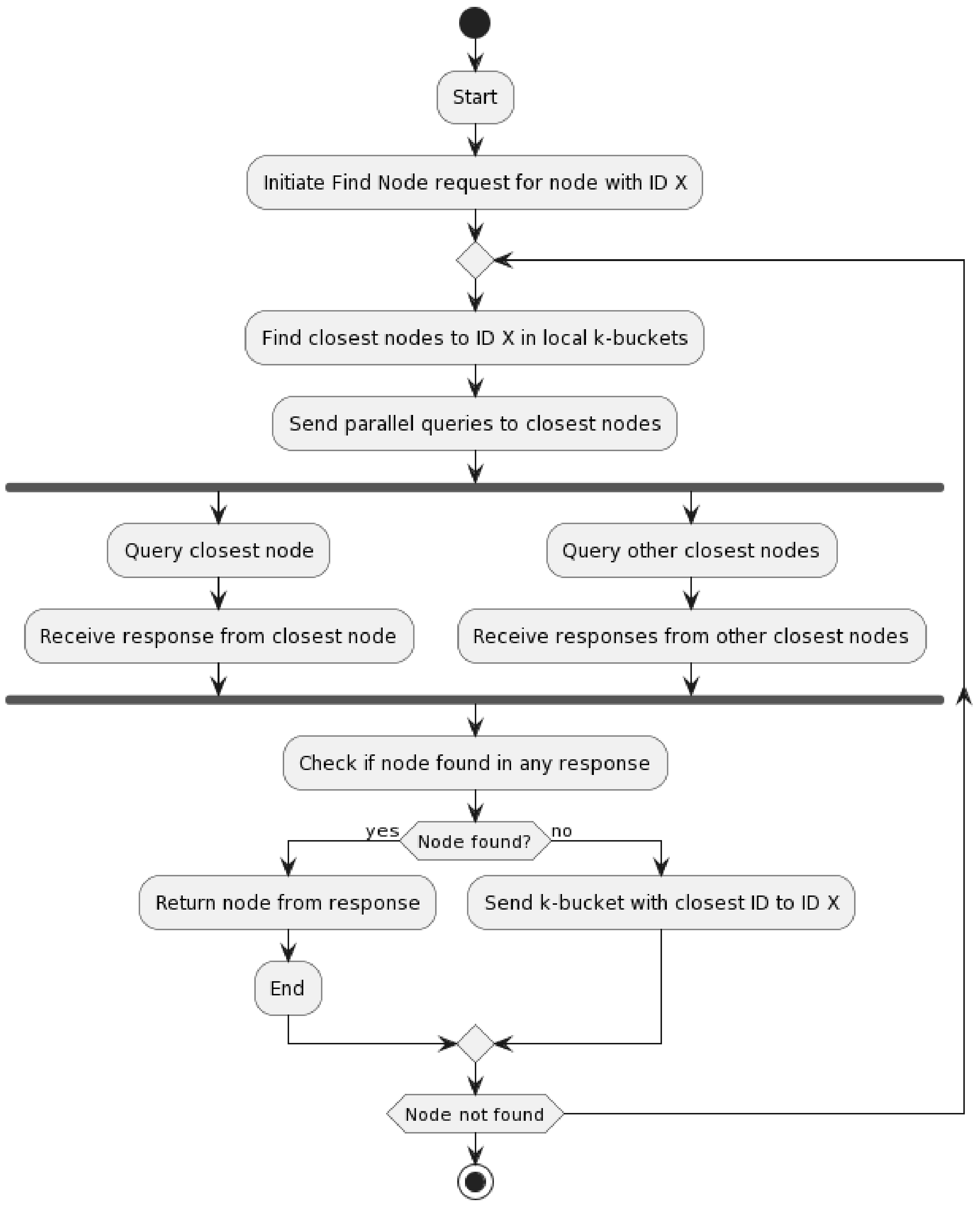

Start
Initiate Find Node request for node with ID X
Find closest nodes to ID X in local k-buckets
Send parallel queries to closest nodes
Query closest node
Query other closest nodes
Receive response from closest node
Receive responses from other closest nodes
Check if node found in any response
yes
Node found?
no
Return node from response
Send k-bucket with closest ID to ID X
End
Node not found

Figure 3.3: Locating Nodes

3.3.4 Locating resources

Information is located by mapping it to a key. A hash is typically used for the map. The storer nodes will have information due to a previous STORE message. Locating a value follows the same procedure as locating the closest nodes to a key, except the search terminates when a node has the requested value in its store and returns this value. Periodically, a node that stores a value will explore the network to find the k nodes that are close to the key value and replicate the value onto them. This compensates for disappeared nodes.

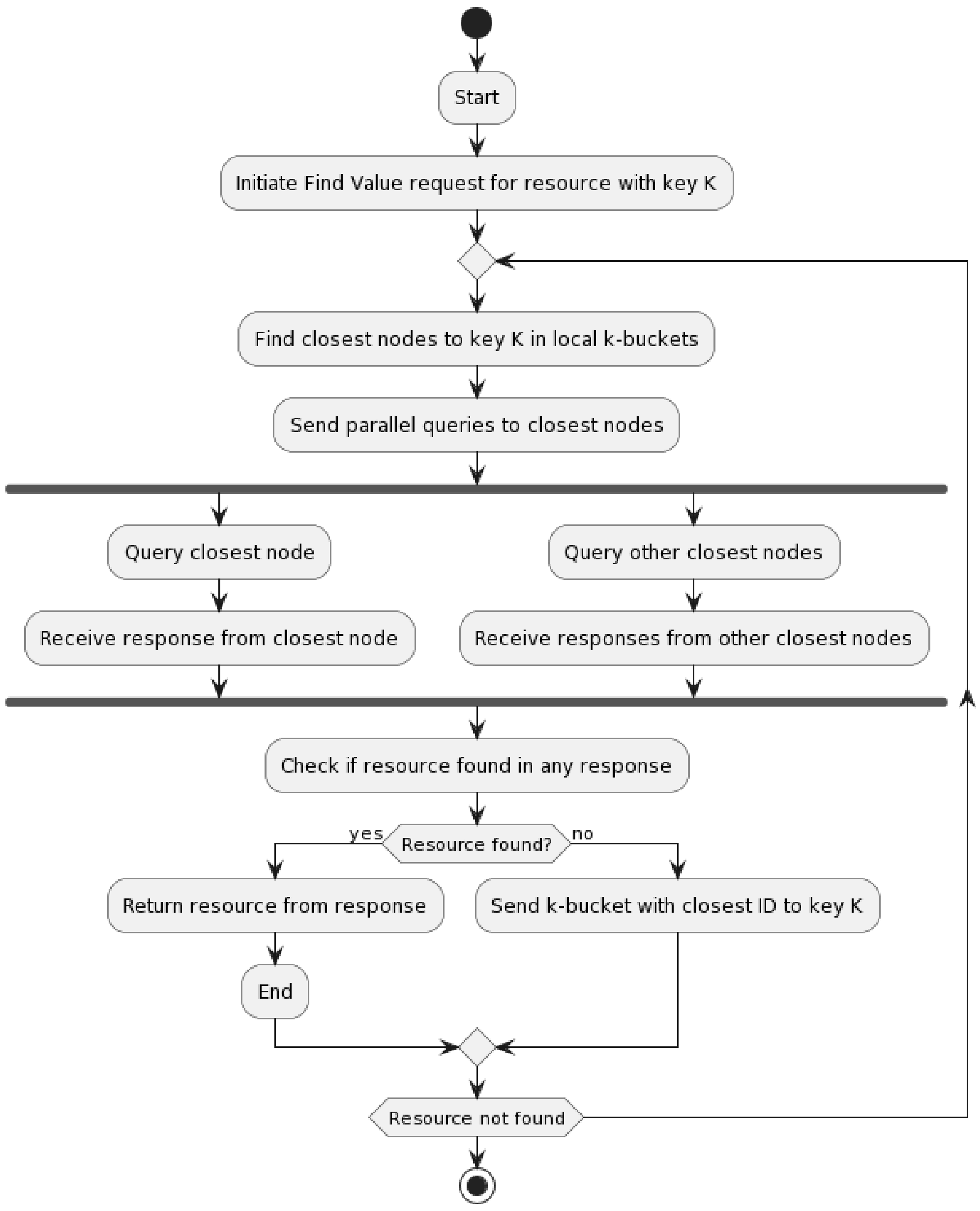

Start
Initiate Find Value request for resource with key K
Find closest nodes to key K in local k-buckets
Send parallel queries to closest nodes
Query closest node
Query other closest nodes
Receive response from closest node
Receive responses from other closest nodes
Check if resource found in any response
yes
Resource found?
no
Return resource from response
Send k-bucket with closest ID to key K
End
Resource not found

Figure 3.4: Locating resources

3.3.5 Storing Files in Network

The process described outlines a method for storing data within a network utilizing a Distributed Hash Table (DHT) mechanism. It commences with the initiation of a store request for data, specifying a key and its corresponding value. Subsequently, the system identifies the nearest nodes to the key within local k-buckets and concurrently dispatches queries to these nodes. Following the reception of responses, it verifies the presence of the responsible node for data storage. If such a node is identified, the data is stored therein; otherwise, the local k-buckets are updated with the acquired information. This iterative procedure persists until the responsible node is determined, ensuring effective data storage while dynamically accommodating alterations in the network topology.

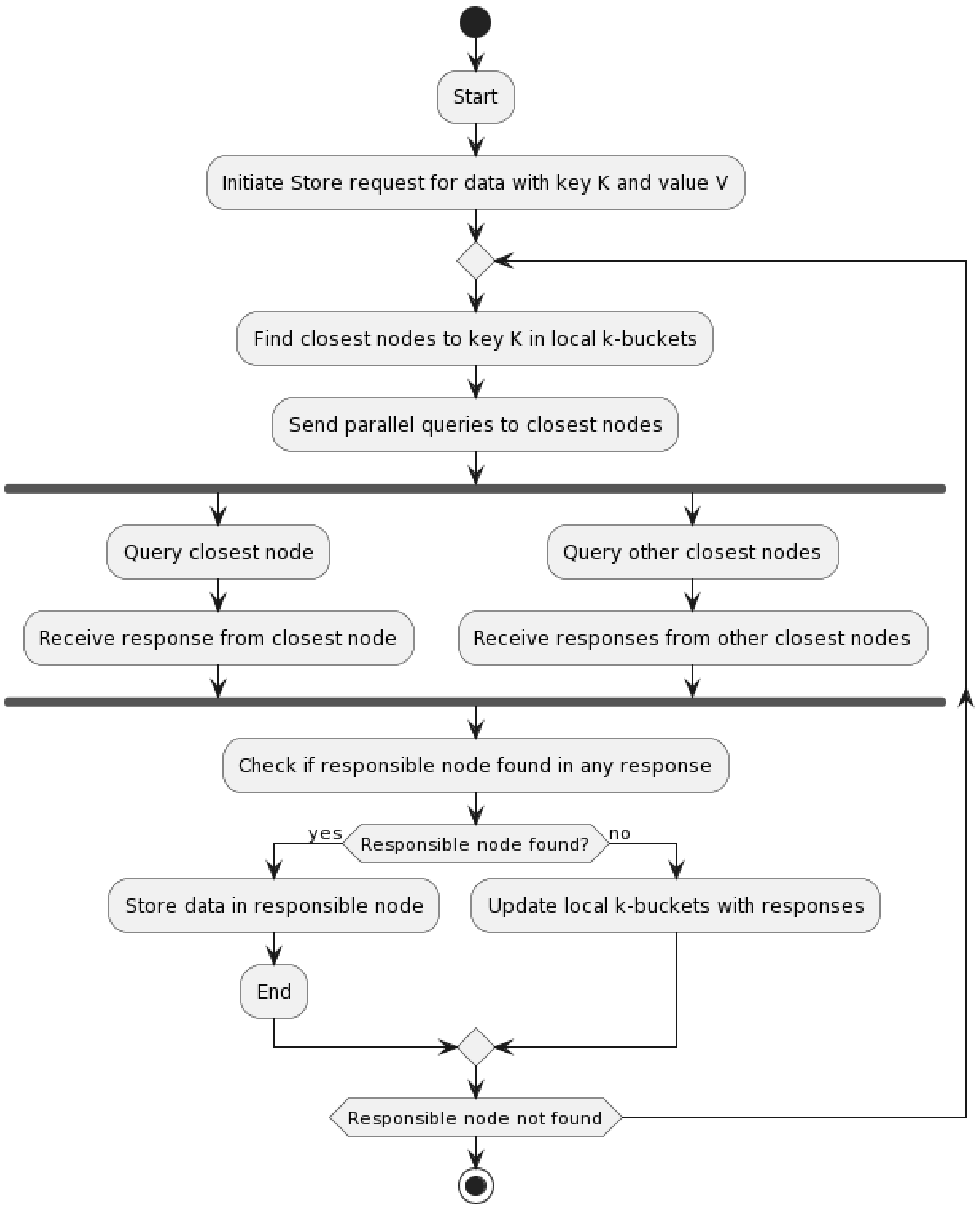

Start
Initiate Store request for data with key K and value V
Find closest nodes to key K in local k-buckets
Send parallel queries to closest nodes
Query closest node
Query other closest nodes
Receive response from closest node
Receive responses from other closest nodes
Check if responsible node found in any response
Responsible node found?
yes
no
Store data in responsible node
Update local k-buckets with responses
End
Responsible node not found

Figure 3.5: Storing Files in network

4. Experimental Setup

4.1 Bootstrapping Server

We utilized a bootstrapping node to initialize new nodes into the Kademlia network. For this purpose, we will deploy a bootstrap server during development and testing.

4.2 P2P Network Testing

This type of the project naturally requires multiple devices to test the app during development phases. Instead of using multiple physical devices, we will utilize existing technologies like Kathara, Mininet and Skydive. We wrote a simple bash script to run the multiple instances of program binding to different ports too.

4.3 Hole punching

For the hole punching procedure we utilized azure vm to host the stun server to act as intermediary between two nodes. Although we couldn't successfully implement it in the project due to NAT issuses and time frame of project.

5. System design

5.1 Components and Interactions

5.1.1 Kademlia

Kademlia uses a distance calculation between two nodes. This distance is computed as the exclusive or (XOR) of the two node IDs, taking the result as an unsigned integer number. Keys and node IDs have the same format and length, so distance can be calculated among them in exactly the same way. The node ID is typically a large random number that is chosen with the goal of being unique for a particular node.Each Kademlia search iteration comes one bit closer to the target. A basic Kademlia search algorithm has complexity of O(log2(n)).[2]

Fixed-size routing tables

Kademlia routing tables consist of a list for each bit of the node ID (e.g. if a node ID consists of 128 bits, a node will keep 128 such lists.) Every entry in a list holds the necessary data to locate another node. The data in each list entry is typically the IP address, port, and node ID of another node. Every list corresponds to a specific distance from the node. Nodes that can go in the nth list must have a differing nth bit from the node's ID; the first n-1 bits of the candidate ID must match those of the node's ID. This means that it is very easy to populate the first list as 1/2 of the nodes in the network are far away candidates. The next list can use only 1/4 of the nodes in the network (one bit closer than the first), etc.[2]

As nodes are encountered on the network, they are added to the lists. This includes store and retrieval operations and even helping other nodes to find a key. Every node encountered will be considered for inclusion in the lists. Therefore, the knowledge that a node has of the network is very dynamic. This keeps the network constantly updated and adds resilience to failures or attacks.In the Kademlia literature, the lists are referred to as k-buckets. k is a system wide number.[2]

It is known that nodes that have been connected for a long time in a network will probably remain connected for a long time in the future. Because of this statistical distribution, Kademlia selects long connected nodes to remain stored in the k-buckets. This increases the number of known valid nodes at some time in the future and provides for a more stable network.

When a k-bucket is full and a new node is discovered for that k-bucket, the least recently seen node in the k-bucket is PINGed. If the node is found to be still alive, the new node is placed in a secondary list, a replacement cache. The replacement cache is used only if a node in the k-bucket stops responding. In other words: new nodes are used only when older nodes disappear.

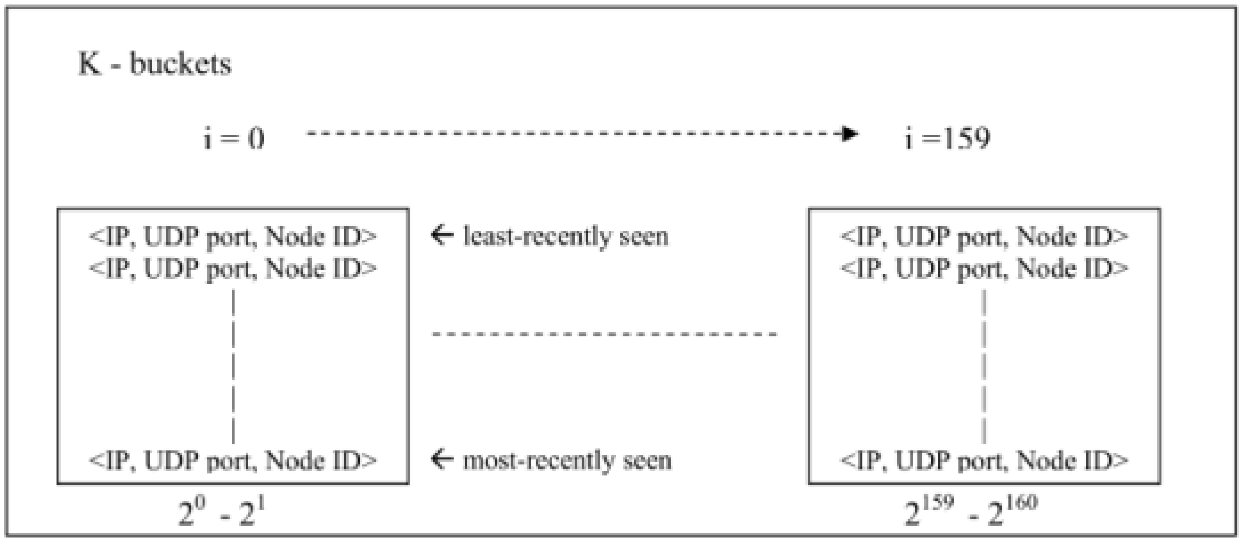


Figure 5.1: K-Bucket

Protocol messages

- PING — Used to verify that a node is still alive.
- STORE — Stores a (key, value) pair in one node.
- FIND_NODE — The recipient of the request will return the k nodes in its own buckets that are the closest ones to the requested key.
- FIND_VALUE — Same as FIND_NODE, but if the recipient of the request has the requested key in its store, it will return the corresponding value.
- GET_STORAGE_INFO - Used to get the available storage of the particular node.

5.1.2 Monitoring

”Monitoring” in the context of this project refers to a crucial component responsible for ensuring the persistence of stored files within the network infrastructure. In technical terms, it involves the implementation of a file system with metadata that tracks the whereabouts and status of files stored across the network.

This monitoring component operates by periodically querying the nodes that hold fragments or segments of the files. The purpose is to verify the continued presence and integrity of these file fragments within the network. This querying process occurs at least once every 24 hours to maintain a consistent check on the availability and completeness of the stored files.

5.1.3 Asynchronous I/O (ASIO)

Asynchronous I/O (Input/Output), often referred to as ASIO (Asynchronous I/O), represents a programming paradigm and accompanying library meticulously crafted to manage I/O operations with exceptional efficiency, particularly in scenarios where responsiveness and scalability hold paramount importance, such as in network programming and high-performance computing environments. Unlike traditional synchronous I/O operations that halt program execution until completion, ASIO empowers developers to initiate I/O tasks—like file or socket operations—and seamlessly proceed with other tasks while awaiting their conclusion. This non-blocking approach revolutionizes resource utilization and slashes latency, thereby enhancing overall application performance and responsiveness.

At the heart of ASIO lies its ability to liberate programs from the shackles of synchronous operation by allowing them to execute I/O tasks concurrently with other operations. Traditionally, programs would halt execution until I/O tasks, such as reading from or writing to a file or socket, were finished, resulting in potential inefficiencies and increased latency. However, ASIO introduces a paradigm shift by decoupling I/O operations from program execution, enabling the program to continue functioning while the I/O operation progresses asynchronously. This architectural innovation not only optimizes CPU utilization but also ensures that applications remain highly responsive even when dealing with intensive I/O tasks.

5.1.4 User interface

A Command Line User Interface (CLI) acts as a text-based interface for users to interact with computer systems and applications, offering essential commands "help," "store," "retrieve," "ls," and "status."

The "help" command serves as a beacon of guidance, providing users with assistance in navigating through available commands and understanding their functionalities. Typing "help" followed by a specific command or topic grants users access to concise explanations and usage instructions, empowering them to wield the CLI effectively.

For data management tasks, "store" and "retrieve" commands form the backbone of functionality. With "store," users securely save data, configurations, or files to network, ensuring easy access and organization. Conversely, "retrieve" enables swift access to stored data, facilitating seamless data retrieval and utilization within the CLI ecosystem.

Navigating the file system is simplified with the "ls" command, allowing users to list the files and directories stored in network effortlessly. Upon execution, "ls" displays files and directories owned by the user which are stored in the network, offering a comprehensive view of the file system structure.

Additionally, the "status" command provides users with status of files or folders shared in the network. This command offers users real-time visibility into the state of their data in the network.

Together, these foundational commands streamline user interactions within the CLI environment, fostering productivity and efficiency. Whether seeking guidance, managing data, navigating the file system, or monitoring data status in the network, users can rely on these essential commands to navigate the CLI landscape with confidence and ease.

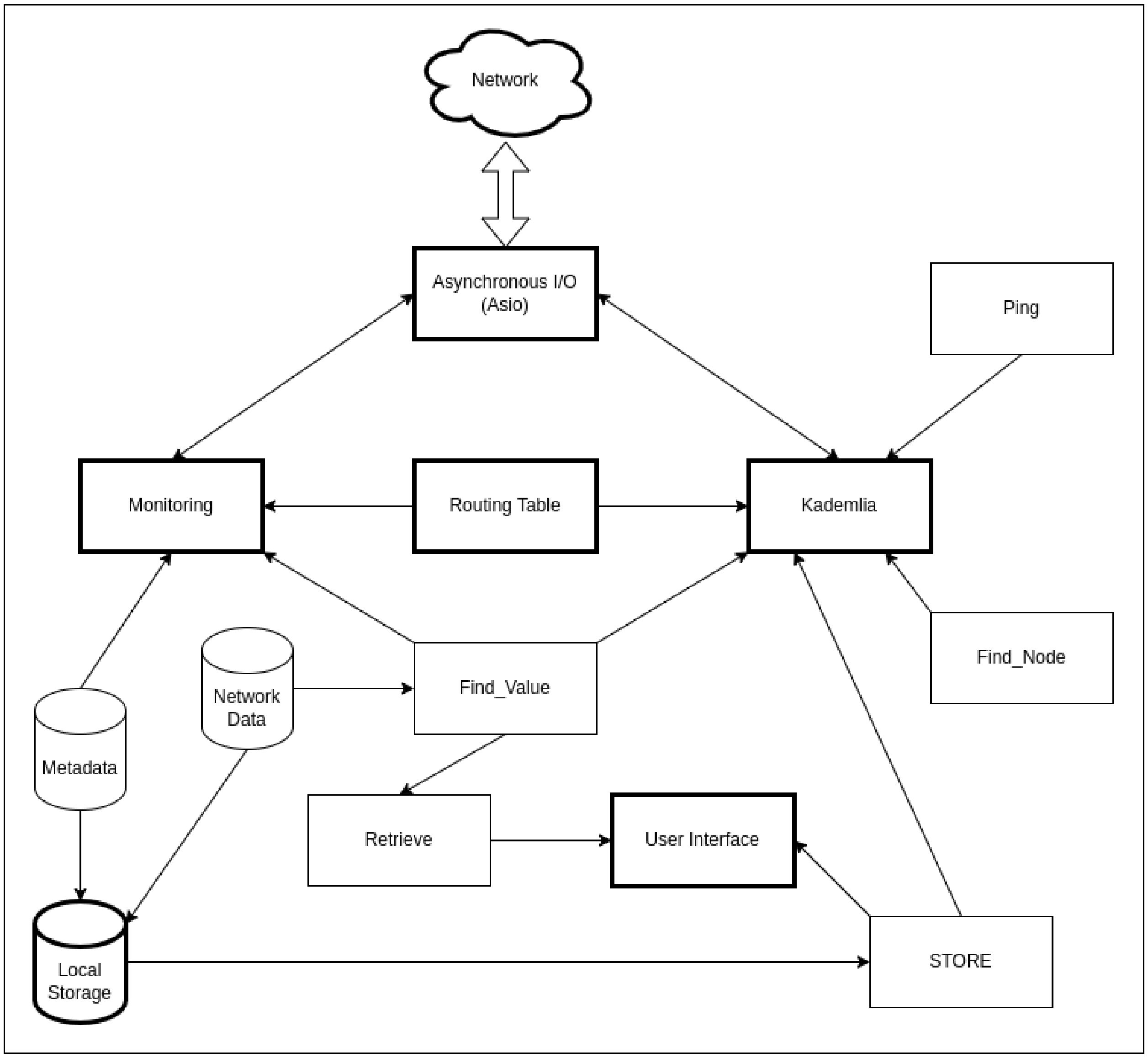


Figure 5.2: Component interaction (Single Node View)

5.2 UML Diagrams

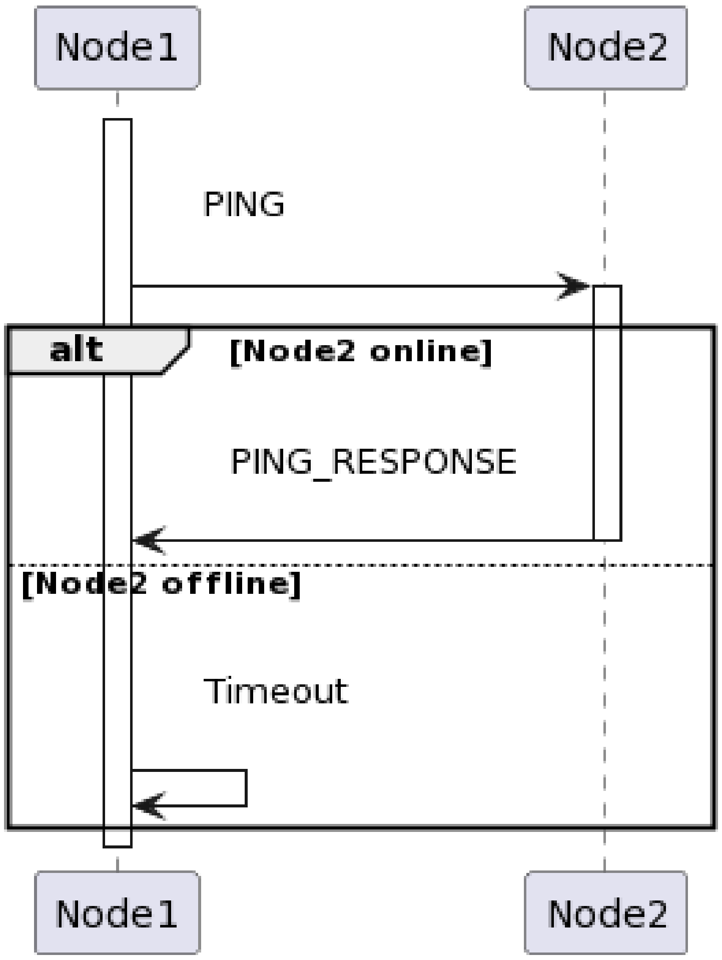


Figure 5.3: PING message interaction between two node

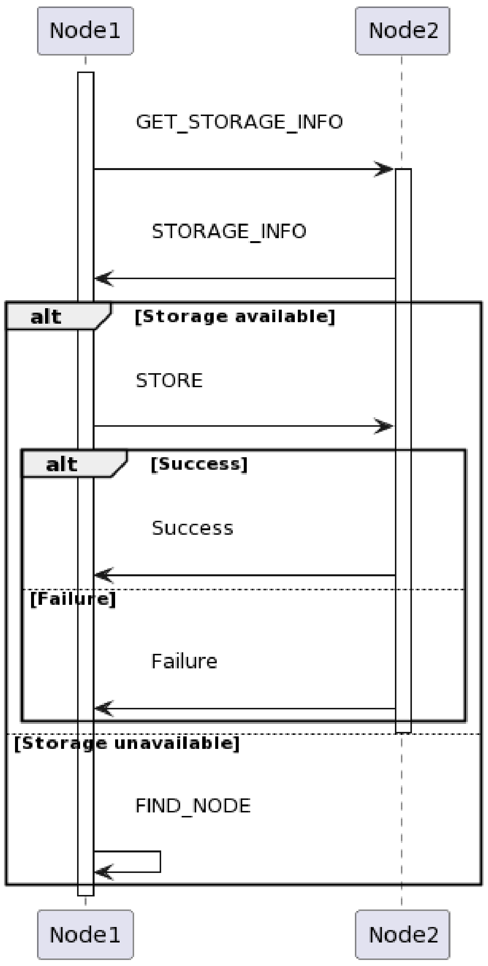


Figure 5.4: STORE message interaction between two node

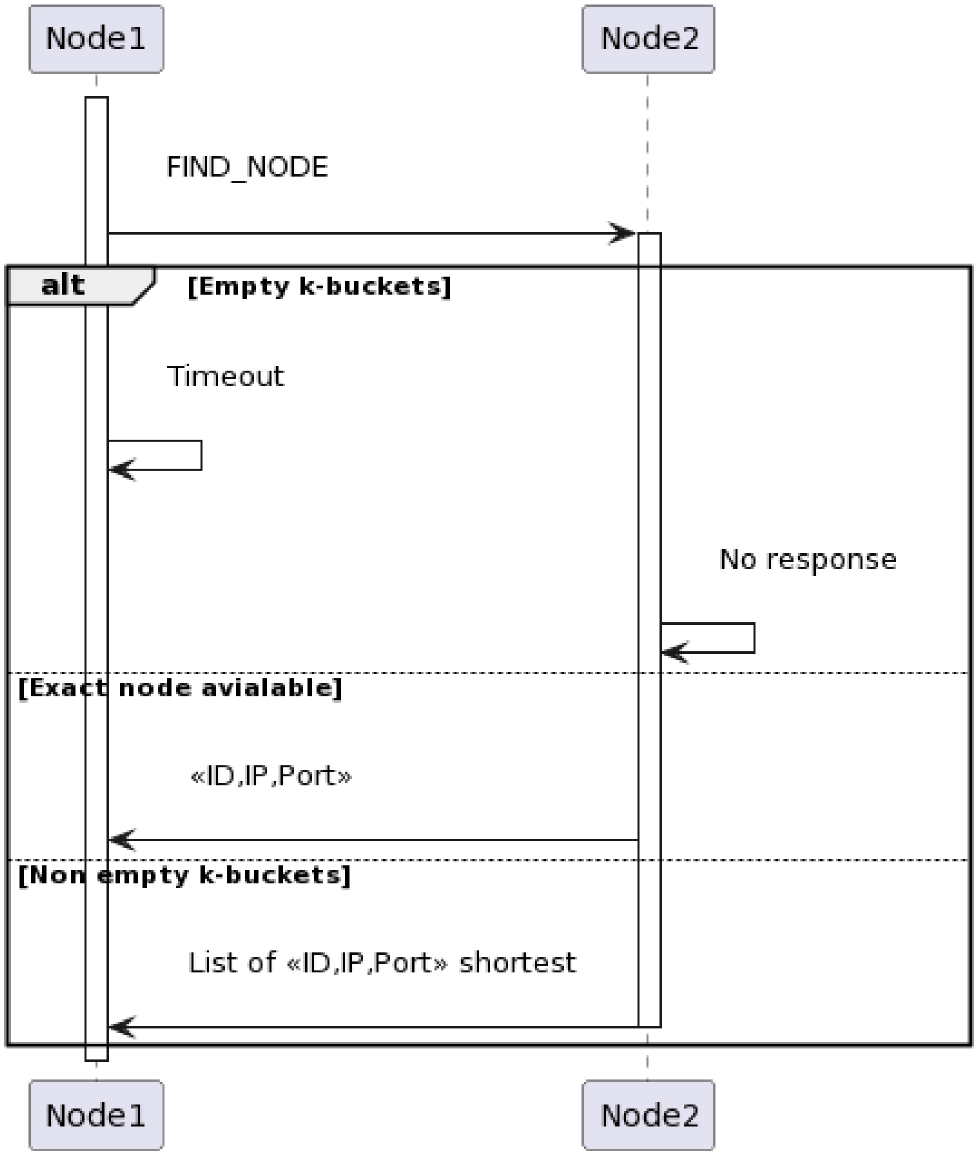


Figure 5.5: FIND_NODE message interaction between two node

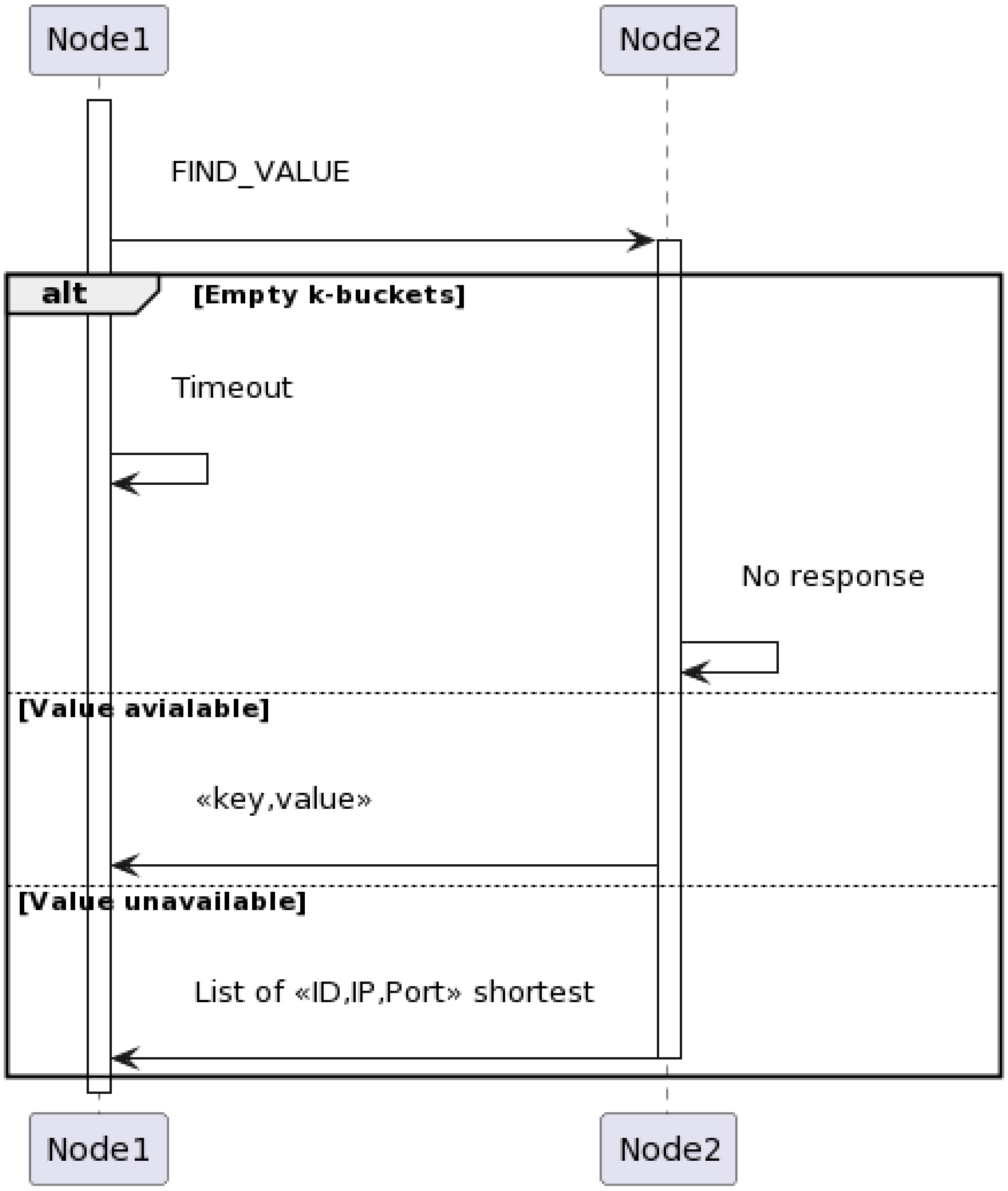


Figure 5.6: FIND_VALUE message interaction between two node

5.3 Language and Tools

5.3.1 C++

C++ stands as a cornerstone of modern software development, renowned for its efficiency, versatility, and performance. This high-level, statically typed programming language offers a rich set of features, including object-oriented and generic programming paradigms, memory management capabilities, and a vast standard library, empowering developers to tackle diverse and complex tasks with ease. From system programming to game development, C++ finds application in a wide array of domains, owing to its ability to produce highly optimized and portable code. With its strong emphasis on both low-level hardware interactions and high-level abstractions, C++ remains a preferred choice for building robust, scalable, and efficient software solutions that meet the demands of today's computing landscape.

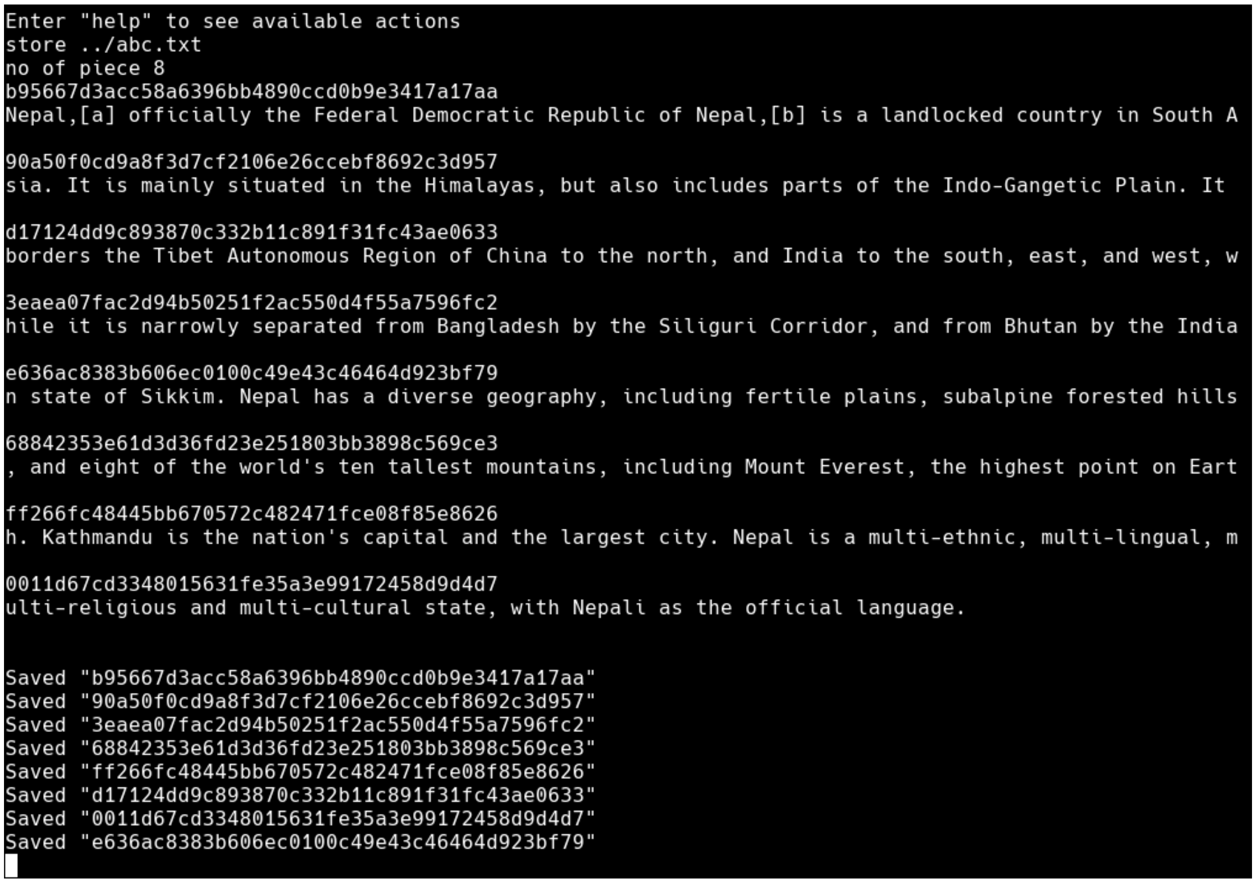
5.3.2 ASIO

ASIO, short for Asynchronous I/O, emerges as a pivotal library and programming paradigm within the realm of network programming and high-performance computing. This asynchronous I/O framework, originally part of the Boost C++ Libraries and later integrated into the C++ Standard Library, revolutionizes the handling of I/O operations by enabling non-blocking execution. At its core, ASIO facilitates asynchronous communication by allowing programs to initiate I/O tasks, such as reading from or writing to files or sockets, and continue executing other operations while awaiting completion. Its event-driven architecture simplifies the design of asynchronous applications, triggering callbacks or completion handlers upon I/O events. ASIO's platform independence and support for various protocols, including TCP/IP, UDP, and SSL/TLS, make it a versatile choice for developing cross-platform networking applications. With ASIO, developers can harness the power of asynchronous I/O to enhance responsiveness, scalability, and performance in their C++ applications, making it a cornerstone of modern network programming.

6. Results & Discussion

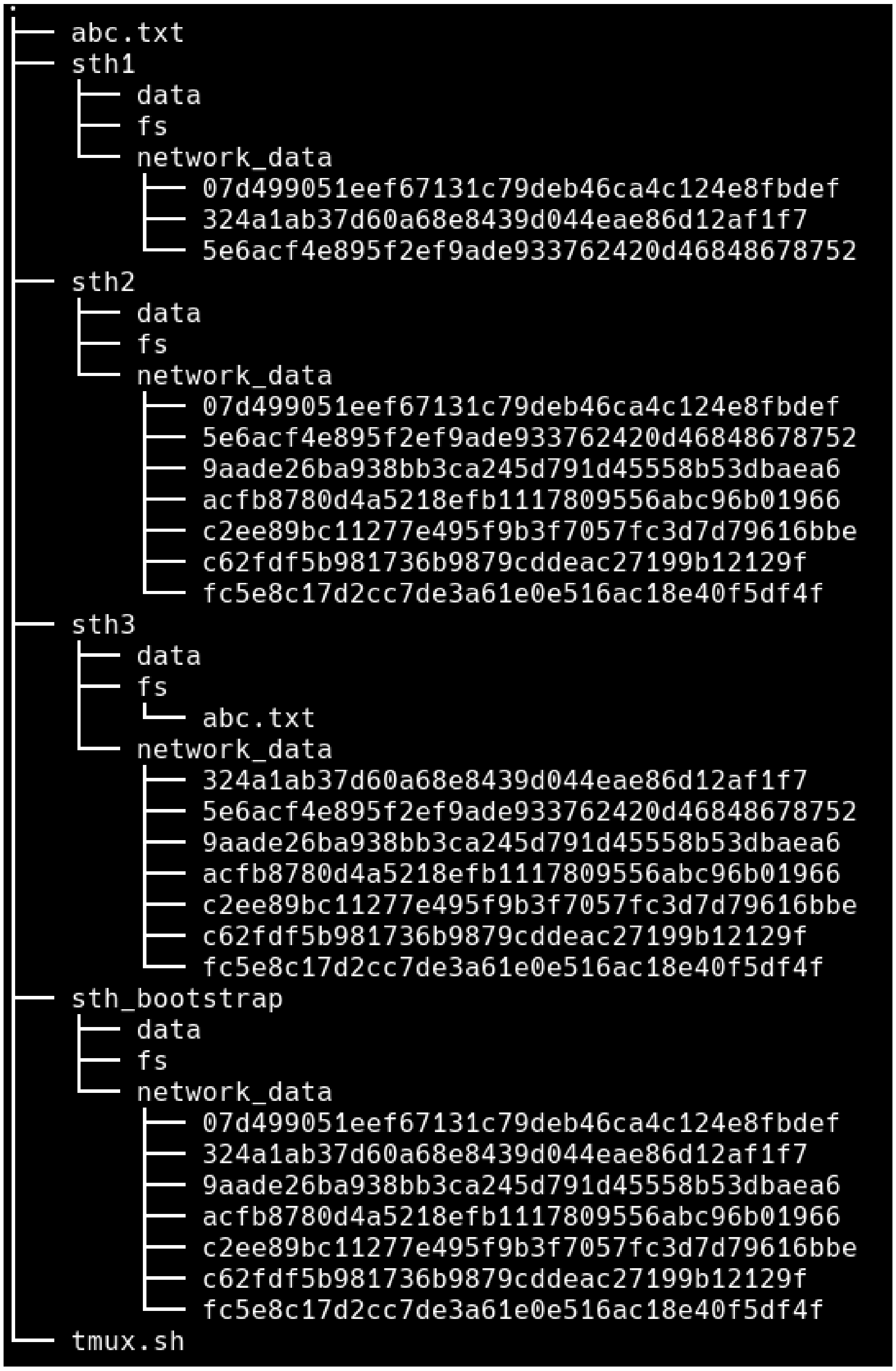
6.1 Store

The file is stored in the multiple nodes in the network after it has been separated into smaller chucks of piece (100B) for the test purpose. The following images verify the above statement where the 4 different nodes are created aslong with bootstrap node and file named abc.txt is stored in the network.



```
Enter "help" to see available actions
store ../abc.txt
no of piece 8
b95667d3acc58a6396bb4890ccd0b9e3417a17aa
Nepal,[a] officially the Federal Democratic Republic of Nepal,[b] is a landlocked country in South A

90a50f0cd9a8f3d7cf2106e26ccebf8692c3d957
sia. It is mainly situated in the Himalayas, but also includes parts of the Indo-Gangetic Plain. It

d17124dd9c893870c332b11c891f31fc43ae0633
borders the Tibet Autonomous Region of China to the north, and India to the south, east, and west, w

3eaea07fac2d94b50251f2ac550d4f55a7596fc2
hile it is narrowly separated from Bangladesh by the Siliguri Corridor, and from Bhutan by the India

e636ac8383b606ec0100c49e43c46464d923bf79
n state of Sikkim. Nepal has a diverse geography, including fertile plains, subalpine forested hills

68842353e61d3d36fd23e251803bb3898c569ce3
, and eight of the world's ten tallest mountains, including Mount Everest, the highest point on Eart

ff266fc48445bb670572c482471fce08f85e8626
h. Kathmandu is the nation's capital and the largest city. Nepal is a multi-ethnic, multi-lingual, m

0011d67cd3348015631fe35a3e99172458d9d4d7
ulti-religious and multi-cultural state, with Nepali as the official language.


Saved "b95667d3acc58a6396bb4890ccd0b9e3417a17aa"
Saved "90a50f0cd9a8f3d7cf2106e26ccebf8692c3d957"
Saved "3eaea07fac2d94b50251f2ac550d4f55a7596fc2"
Saved "68842353e61d3d36fd23e251803bb3898c569ce3"
Saved "ff266fc48445bb670572c482471fce08f85e8626"
Saved "d17124dd9c893870c332b11c891f31fc43ae0633"
Saved "0011d67cd3348015631fe35a3e99172458d9d4d7"
Saved "e636ac8383b606ec0100c49e43c46464d923bf79"
```

Figure 6.1: Store function output

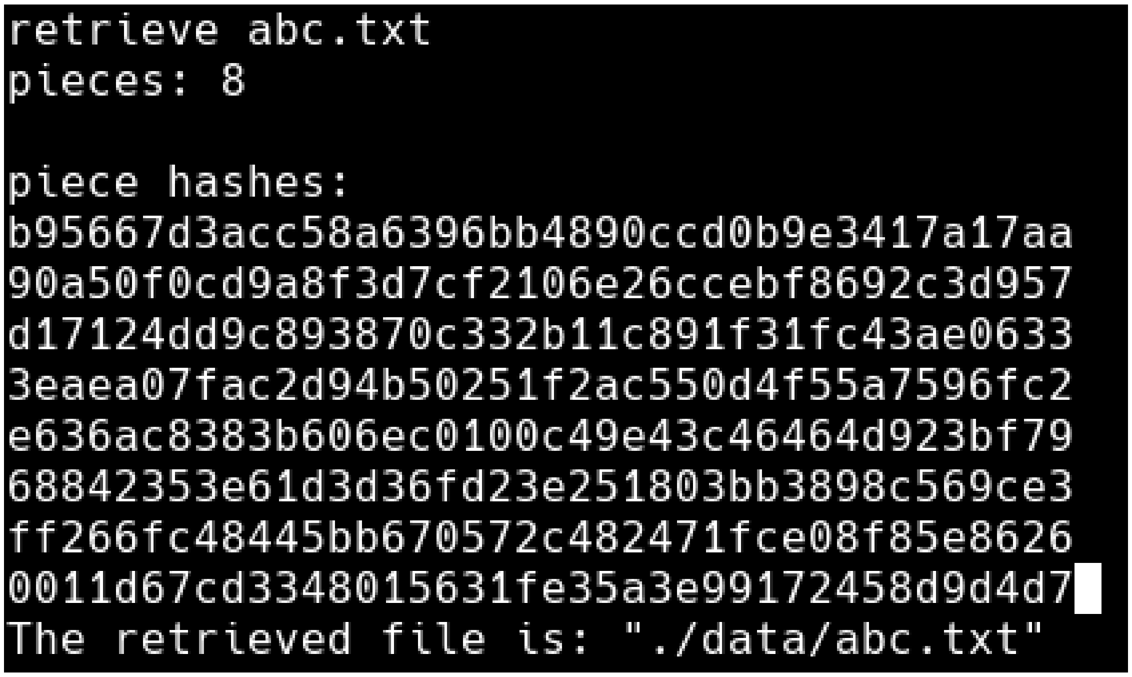


```
.
├── abc.txt
├── sth1
│   ├── data
│   ├── fs
│   └── network_data
│       ├── 07d499051eef67131c79deb46ca4c124e8fbdef
│       ├── 324a1ab37d60a68e8439d044eae86d12af1f7
│       └── 5e6acf4e895f2ef9ade933762420d46848678752
├── sth2
│   ├── data
│   ├── fs
│   └── network_data
│       ├── 07d499051eef67131c79deb46ca4c124e8fbdef
│       ├── 5e6acf4e895f2ef9ade933762420d46848678752
│       ├── 9aade26ba938bb3ca245d791d45558b53dbaea6
│       ├── acfb8780d4a5218efb1117809556abc96b01966
│       ├── c2ee89bc11277e495f9b3f7057fc3d7d79616bbe
│       ├── c62fdf5b981736b9879cddeac27199b12129f
│       └── fc5e8c17d2cc7de3a61e0e516ac18e40f5df4f
├── sth3
│   ├── data
│   ├── fs
│   │   └── abc.txt
│   └── network_data
│       ├── 324a1ab37d60a68e8439d044eae86d12af1f7
│       ├── 5e6acf4e895f2ef9ade933762420d46848678752
│       ├── 9aade26ba938bb3ca245d791d45558b53dbaea6
│       ├── acfb8780d4a5218efb1117809556abc96b01966
│       ├── c2ee89bc11277e495f9b3f7057fc3d7d79616bbe
│       ├── c62fdf5b981736b9879cddeac27199b12129f
│       └── fc5e8c17d2cc7de3a61e0e516ac18e40f5df4f
├── sth_bootstrap
│   ├── data
│   ├── fs
│   └── network_data
│       ├── 07d499051eef67131c79deb46ca4c124e8fbdef
│       ├── 324a1ab37d60a68e8439d044eae86d12af1f7
│       ├── 9aade26ba938bb3ca245d791d45558b53dbaea6
│       ├── acfb8780d4a5218efb1117809556abc96b01966
│       ├── c2ee89bc11277e495f9b3f7057fc3d7d79616bbe
│       ├── c62fdf5b981736b9879cddeac27199b12129f
│       └── fc5e8c17d2cc7de3a61e0e516ac18e40f5df4f
└── tmux.sh
```

Figure 6.2: File structure(tree) of different instance after store

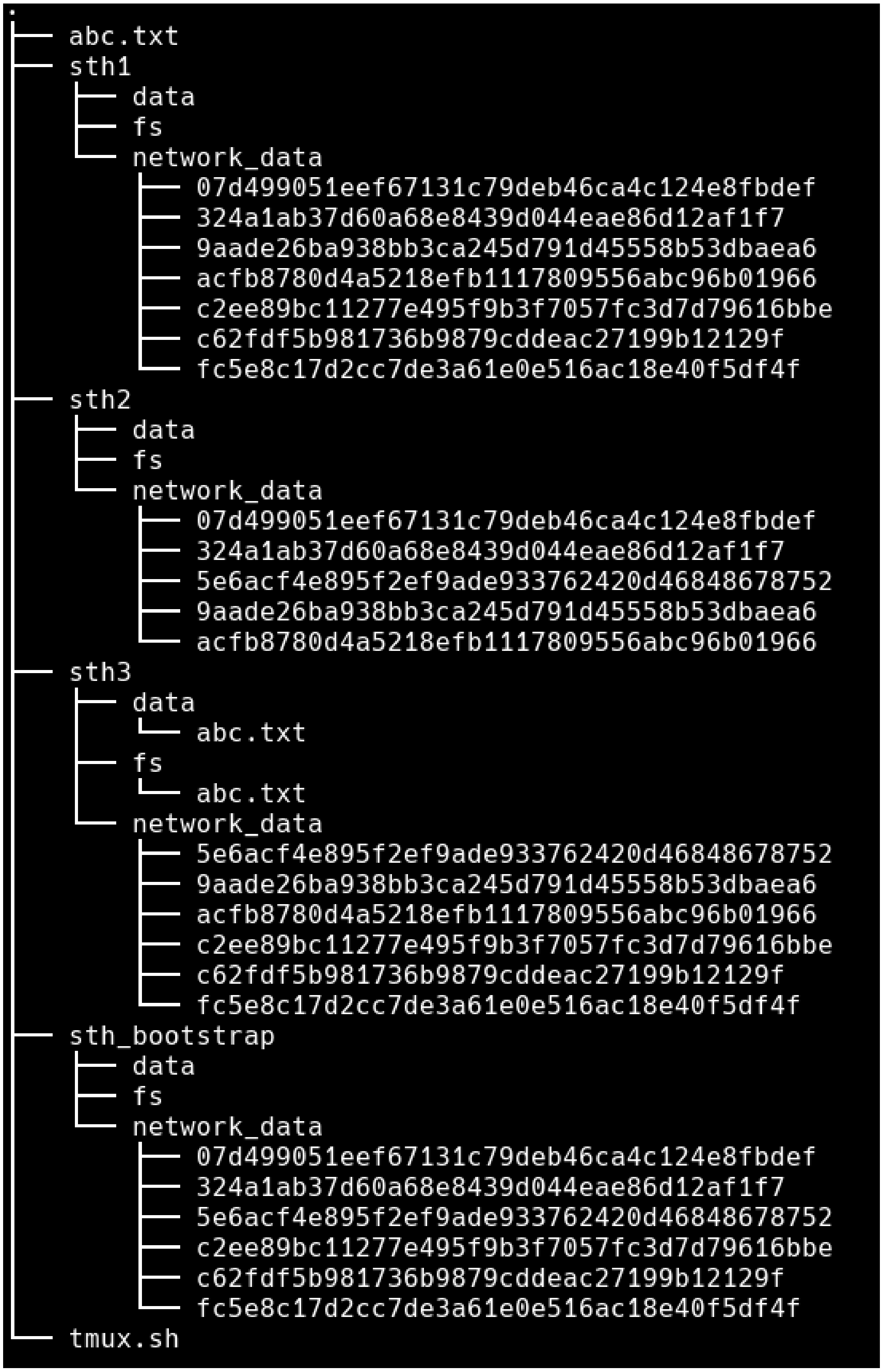
6.2 Retrieve

The file(abc.txt) stored above is retrieved to verify the operations. The following image should justify the above statement.



```
retrieve abc.txt
pieces: 8

piece hashes:
b95667d3acc58a6396bb4890ccd0b9e3417a17aa
90a50f0cd9a8f3d7cf2106e26ccebf8692c3d957
d17124dd9c893870c332b11c891f31fc43ae0633
3eaea07fac2d94b50251f2ac550d4f55a7596fc2
e636ac8383b606ec0100c49e43c46464d923bf79
68842353e61d3d36fd23e251803bb3898c569ce3
ff266fc48445bb670572c482471fce08f85e8626
0011d67cd3348015631fe35a3e99172458d9d4d7
The retrieved file is: "./data/abc.txt"
```

Figure 6.3: Retrieve function output



```
.
├── abc.txt
├── sth1
│   ├── data
│   ├── fs
│   └── network_data
│       ├── 07d499051eef67131c79deb46ca4c124e8fbdef
│       ├── 324a1ab37d60a68e8439d044eae86d12af1f7
│       ├── 9aade26ba938bb3ca245d791d45558b53dbaea6
│       ├── acfb8780d4a5218efb1117809556abc96b01966
│       ├── c2ee89bc11277e495f9b3f7057fc3d7d79616bbe
│       ├── c62fdf5b981736b9879cddeac27199b12129f
│       └── fc5e8c17d2cc7de3a61e0e516ac18e40f5df4f
├── sth2
│   ├── data
│   ├── fs
│   └── network_data
│       ├── 07d499051eef67131c79deb46ca4c124e8fbdef
│       ├── 324a1ab37d60a68e8439d044eae86d12af1f7
│       ├── 5e6acf4e895f2ef9ade933762420d46848678752
│       ├── 9aade26ba938bb3ca245d791d45558b53dbaea6
│       └── acfb8780d4a5218efb1117809556abc96b01966
├── sth3
│   ├── data
│   │   └── abc.txt
│   ├── fs
│   │   └── abc.txt
│   └── network_data
│       ├── 5e6acf4e895f2ef9ade933762420d46848678752
│       ├── 9aade26ba938bb3ca245d791d45558b53dbaea6
│       ├── acfb8780d4a5218efb1117809556abc96b01966
│       ├── c2ee89bc11277e495f9b3f7057fc3d7d79616bbe
│       ├── c62fdf5b981736b9879cddeac27199b12129f
│       └── fc5e8c17d2cc7de3a61e0e516ac18e40f5df4f
├── sth_bootstrap
│   ├── data
│   ├── fs
│   └── network_data
│       ├── 07d499051eef67131c79deb46ca4c124e8fbdef
│       ├── 324a1ab37d60a68e8439d044eae86d12af1f7
│       ├── 5e6acf4e895f2ef9ade933762420d46848678752
│       ├── c2ee89bc11277e495f9b3f7057fc3d7d79616bbe
│       ├── c62fdf5b981736b9879cddeac27199b12129f
│       └── fc5e8c17d2cc7de3a61e0e516ac18e40f5df4f
└── tmux.sh
```

Figure 6.4: File structure(tree) of different instance after retrieve

7. Conclusions

This project has successfully explored the feasibility and potential of peer-to-peer (P2P) storage sharing as an alternative to centralized solutions. Through rigorous development and analysis, we have demonstrated the effectiveness of decentralized architectures in maximizing storage resource utilization while prioritizing security and privacy. The system's robust framework ensures data integrity and confidentiality, offering a reliable platform for diverse storage needs. With its scalability and adaptability, it presents a practical solution for individuals and organizations seeking supplementary storage capacity. Furthermore, the project's academic contribution extends beyond practical applications, deepening our understanding of P2P and decentralized systems. As we move forward, this project paves the way for future innovations and advancements in decentralized technologies, contributing to the broader landscape of digital storage solutions.

8. Limitations and Future enhancement

8.1 Limitations

- **Hole Punching:** Hole punching, while effective in establishing direct connections between peers in a P2P network, can be limited by network configurations such as NAT (Network Address Translation) and firewall settings. These configurations may prevent successful hole punching, resulting in connectivity issues and hindering the seamless operation of the P2P system. Additionally, hole punching techniques may not be universally supported across all network environments, leading to inconsistencies in connection establishment and reliability.

- **Lack of File Status Checking:** File owners lack the ability to check the health and condition of their stored files across network nodes. This limitation hinders data integrity monitoring and proactive issue resolution, potentially leading to data loss or inconsistency.

8.2 Further Enhancements

- **User Interface:** Transitioning from a command line interface (CLI) to a graphical user interface (GUI) can vastly improve usability. A GUI enables intuitive navigation, interactive features like drag-and-drop enhancing user experience and accessibility.

- **Proper Error Handling:** Implementing robust error handling mechanisms is crucial for ensuring the stability and reliability of the P2P storage sharing system. Enhancements in error detection, notification, and resolution can help users identify and address issues promptly, minimizing disruptions to their experience.

- **Incentivized Participation:** Building upon the concept of the reward system, further enhancements can focus on incentivizing user participation and contribution to the P2P network. In addition to storage-based rewards, introducing incentives such as badges, levels, or virtual currencies can encourage active engagement. Collaborative challenges, referral programs, and exclusive benefits for top contributors can foster a sense of community and drive sustained involvement in the network.

References

[1] Bram Cohen. Bittorrent - a new p2p app. 2001.

[2] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. 2002.

[3] Juan Benet. Ipfs - content addressed, versioned, p2p file system. 2014.

[4] Protocol Lab. Filecoin: A decentralized storage network. 2017.

[5] James Cope. What's a peer-to-peer (p2p) network? 2002.

[6] Robert ; Karger David ; Kaashoek M. Frans ; Balakrishnan Hari Stoica, Ion ; Morris. Chord: A scalable peer-to-peer lookup service for internet applications. 2001.

[7] Sasu Tarkoma. Overlay networks: Toward information networking. 2010.

[8] Pyda Ford, Bryan; Srisuresh and Dan Kegel. Peer-to-peer communication across network address translators. 2005.