TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

A
PROGRESS REPORT
ON

P2POCKET

SUBMITTED BY:

MANDIP THAPA (PUL077BCT044)
MANISH KUNWAR (PUL077BCT045)

SUBMITTED TO:

DEPARTMENT OF ELECTRONICS & COMPUTER ENGINEERING

February 2024

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 on the ongoing phase of the project. His willingness to give his time so generously has been very much appreciated.

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

Contents

List of Figures

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
RPC	Remote Procedure Call
SHA	Secure Hash Algorithm

1. Introduction

1.1 Background

With an ever increasing trend of digitizing our lives, the demand for personal storage has reached unprecedented levels. Whether it's capturing memories in images and videos, jotting down thoughts, reading ebooks, or creating presentations, everything is computerized and requires storage. However, the storage capacity of our hardware is inherently limited, prompting many to turn to cloud storage solutions. Unfortunately, these alternatives often come with heavy subscription costs and pose challenges related to privacy and security.

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[1], 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

Current file storage systems predominantly rely on centralized architectures, where data is stored on single server. However, this approach introduces limitations in scalability, maintenance and resilience. Centralized systems can become bottlenecks due to server overloads, lack of adaptability to changing user demands, and vulnerability to single points of failure. Centralized systems come with privacy and security issues as well. The last thing you'd want is for your images to be used as data for training artificial intelligence.

1.3 Objectives

The primary goal of the project is to develop a P2P system for the optimized utilization of storage resources of the network. The project targets the development of a robust framework that prioritizes security and privacy for sharing storage among nodes in the network.

1.4 Scope

The proposed system can be used in real life as an alternative to cloud storage for extra storage needs. This project aims to assess the viability of decentralized solutions within diverse domains, encompassing file sharing, content delivery, and communication protocols, among others. Beyond addressing practical storage needs, this project also aims to advance our understanding of P2P and decentralized system within the academic context.

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 protocol, hypermedia and file sharing P2P network for storing and sharing data in a distributed file system. IPFS uses content-addressing to uniquely identify each file in a global namespace connecting IPFS hosts.

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 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.

2.2.4 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 Requirement Analysis

3.1.1 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 file system.
- A user should be able to view the status of the file pieces stored in other nodes.

3.1.2 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.

4. Proposed System design

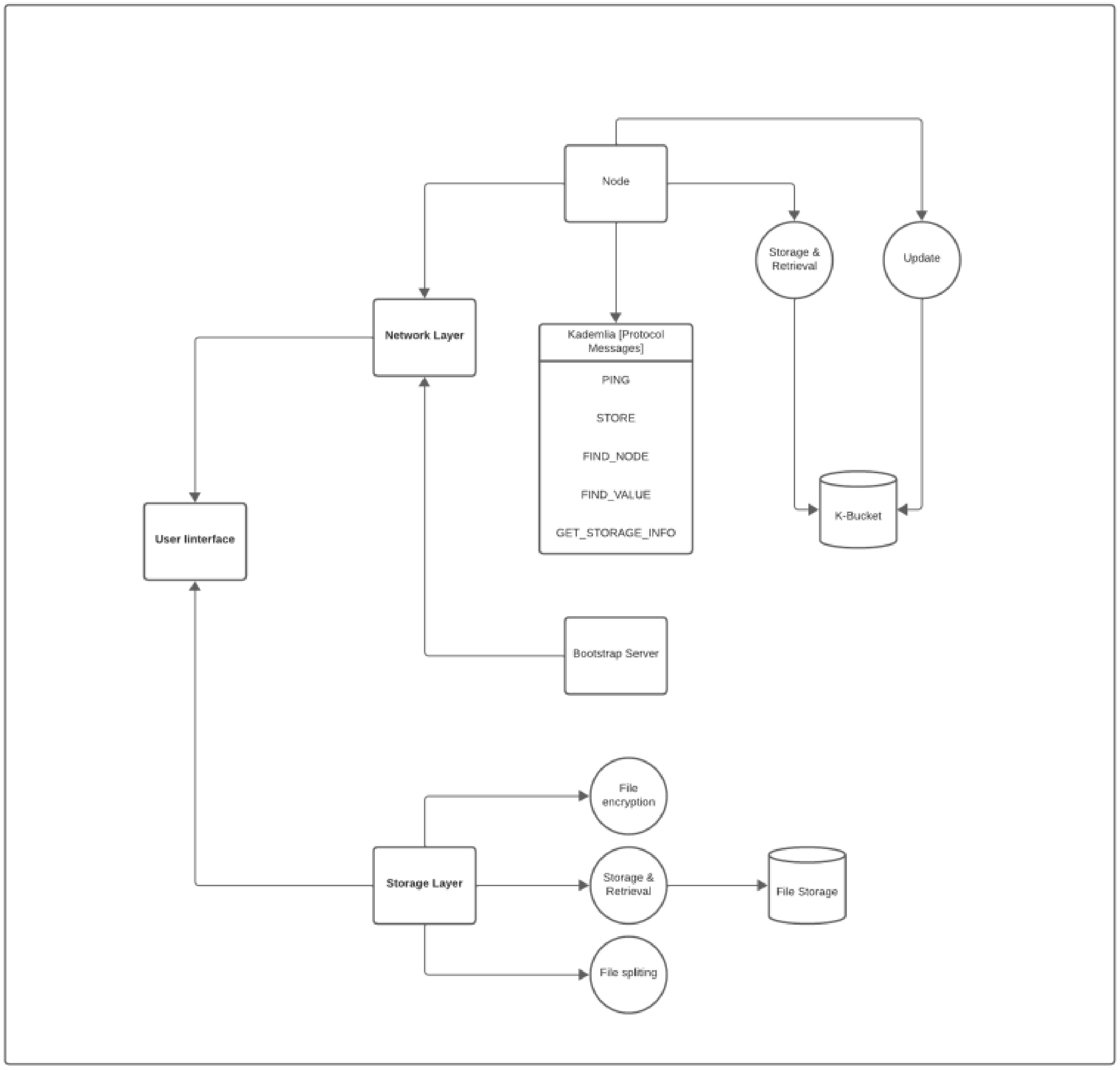

Figure 4.1: Proposed System Block Diagram

4.1 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]

4.1.1 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 keeps 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.

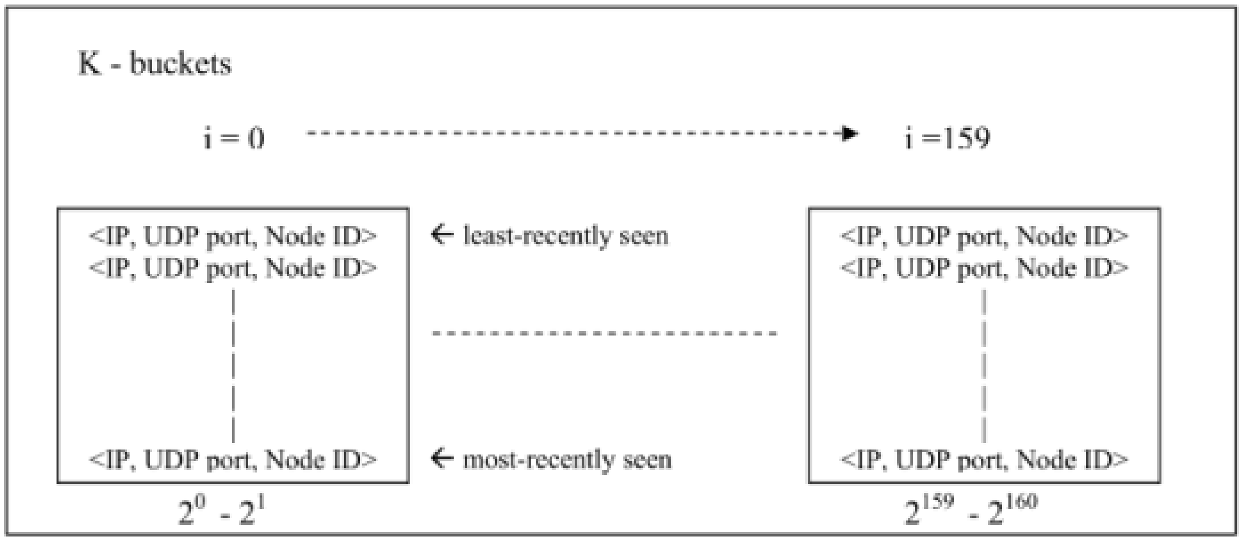

Figure 4.2: K-Bucket

4.1.2 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.

4.2 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. In our case, a bootstrap node is identified via domain name, which resolves to 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.

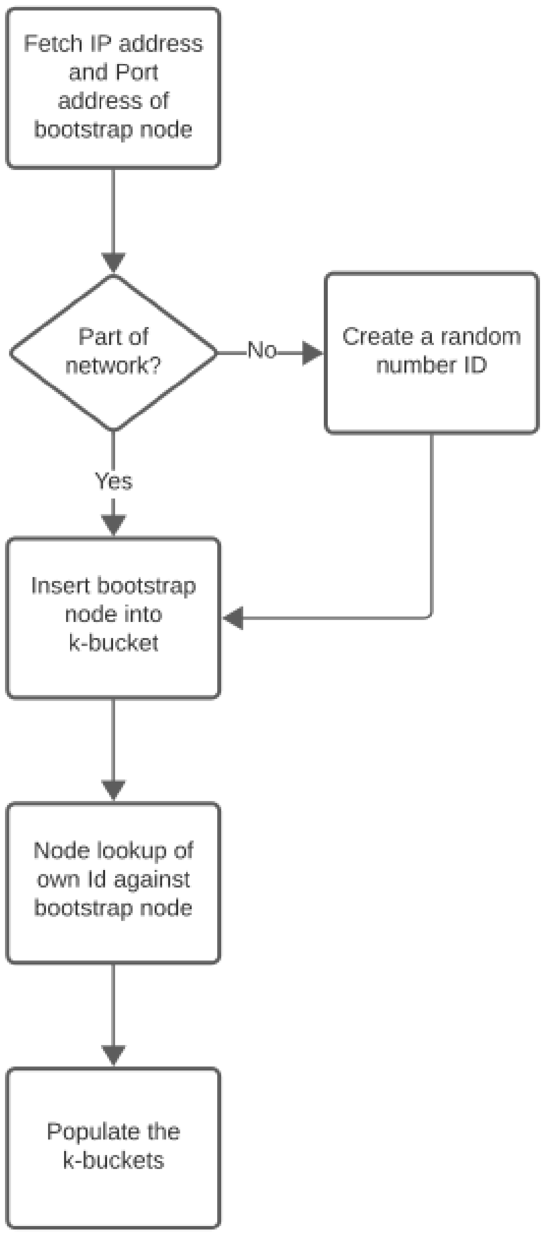
Fetch IP address and Port address of bootstrap node
Part of network?
No
Create a random number ID
Yes
Insert bootstrap node into k-bucket
Node lookup of own Id against bootstrap node
Populate the k-buckets

Figure 4.3: Joining the network

4.3 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.[2]

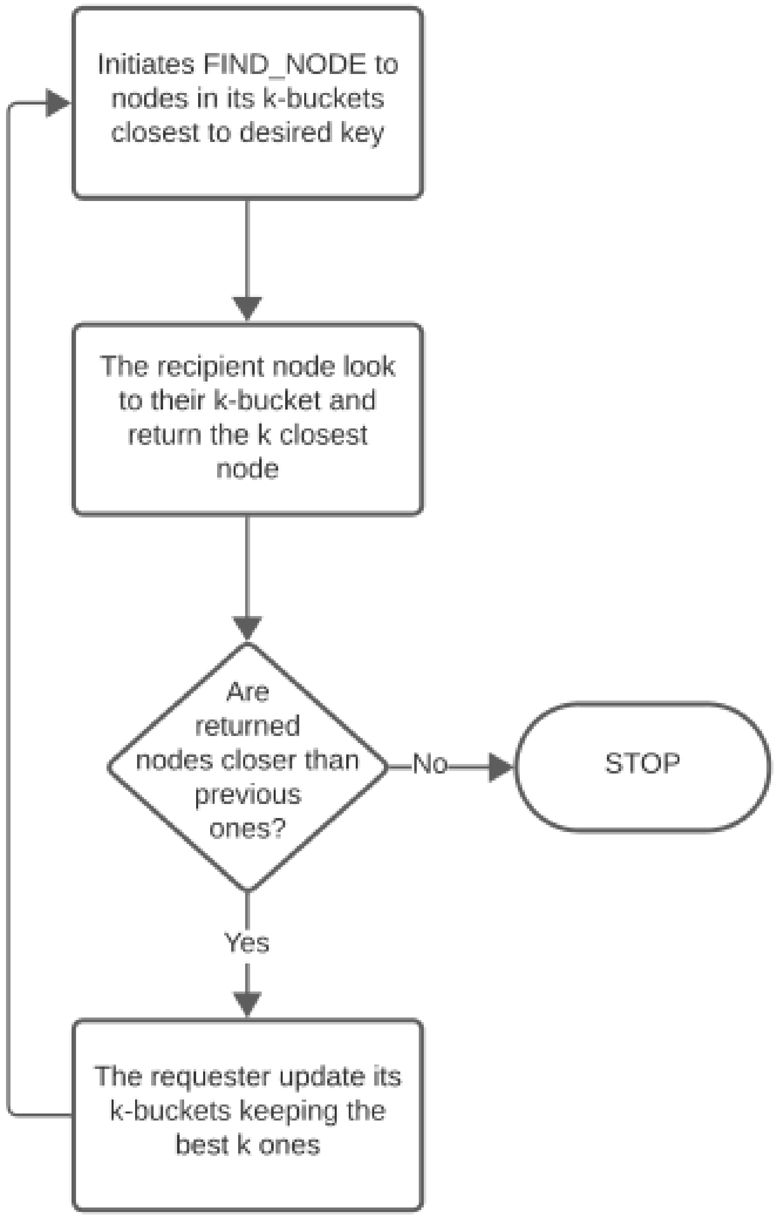
Initiates FIND_NODE to nodes in its k-buckets closest to desired key
The recipient node look to their k-bucket and return the k closest node
Are returned nodes closer than previous ones?
No
STOP
Yes
The requester update its k-buckets keeping the best k ones

Figure 4.4: Locating Node

4.4 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 explores 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.[2]

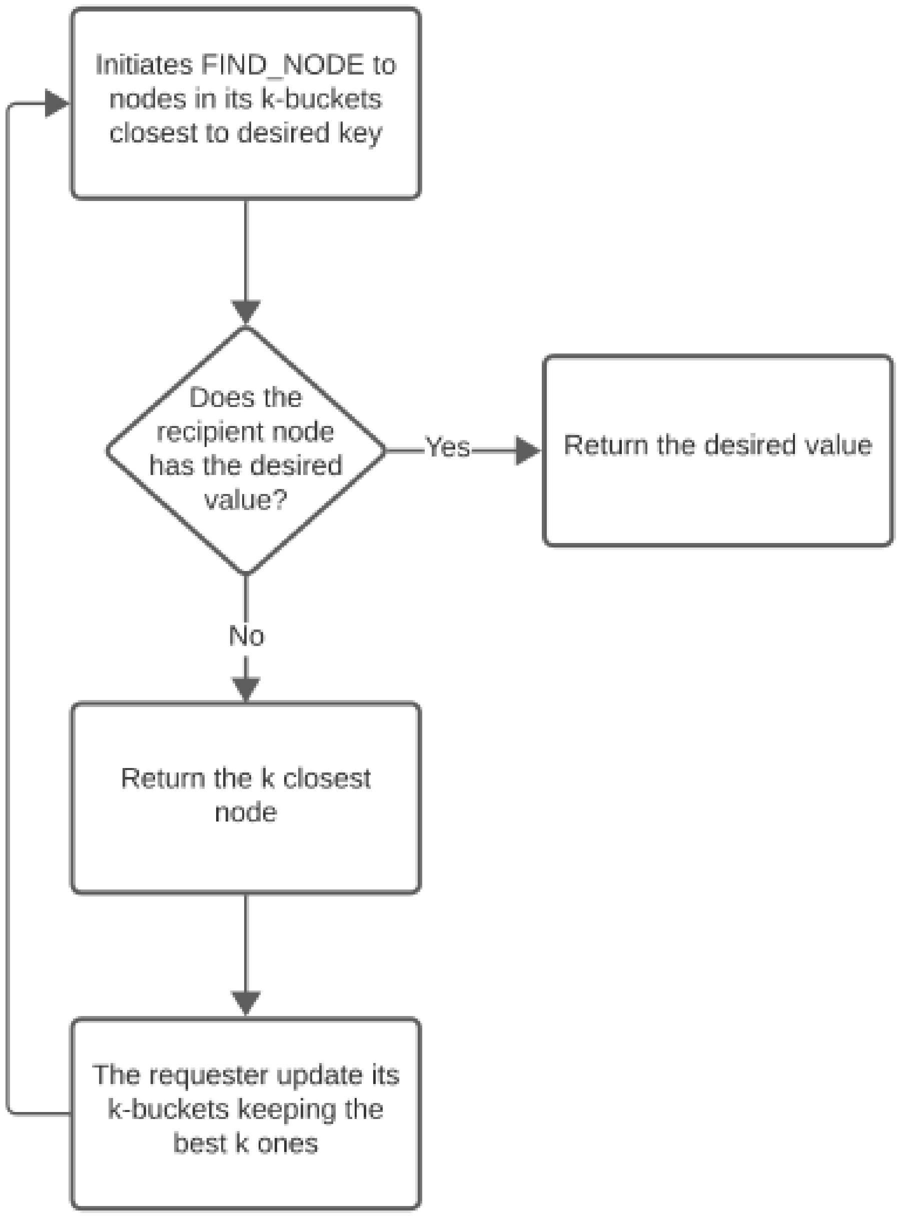
Initiates FIND_NODE to nodes in its k-buckets closest to desired key
Does the recipient node has the desired value?
Yes
Return the desired value
No
Return the k closest node
The requester update its k-buckets keeping the best k ones

Figure 4.5: Locating resources

4.5 User interface

An user interface is created to be used by the end users. The user interface abstracts all the networking and p2p logic and provides an intuitive experience to the users.

5. Task Completed

5.1 Kademlia Implementation

We have successfully implemented Kademlia protocol. This implementation provides a way for node identification, routing, and lookup mechanisms via RPCs. All the tasks of storing and retrieving the files is done to top of the overlay network created by Kademlia.

Integration Details

The integration process involved adapting Kademlia's algorithms to fit our system architecture seamlessly. We ensured that nodes could efficiently discover and communicate with each other, optimizing the network's performance.

Functionality Highlights

Key functionalities such as node joining, routing table maintenance, content storage and retrieval were implemented robustly. This allowed our system to scale effectively and handle a large number of nodes without compromising performance.

5.2 File Splitting and Hashing

Developed a pragmatic approach to break down large files into smaller chunks and generate cryptographic hashes for each segment using SHA1 hash function. Additionally, employing cryptographic hashing guarantees data integrity, as any tampering with the file chunks would be immediately detected during retrieval.

Chunking Strategy

Implemented an efficient chunking strategy to divide files into manageable sizes, balancing between chunk granularity and overhead. The size of chunk is fixed to be 100 bytes.

Hashing Mechanism

Utilized cryptographic hash functions to generate unique identifiers for each file chunk. This hashing mechanism not only ensures data integrity but also facilitates quick verification of chunk authenticity during retrieval.

5.3 File Storing and Retrieval

Implemented robust functionalities for storing and retrieving files within the decentralized network seamlessly. Leveraging the Kademlia protocol and our file splitting and hashing techniques, users can store files across multiple nodes, enhancing redundancy and fault tolerance. Our retrieval mechanisms are designed to efficiently locate and fetch relevant file chunks, prioritizing speed and reliability in accessing stored data. Currently, each file is stored in 3 nodes for fault tolerance and handling of node unavailability.

6. Task Remaining

6.1 Hole Punching

Implement a hole punching mechanism to address the challenge of establishing direct communication between nodes that are behind NAT (Network Address Translation). NAT often prevents direct connections between nodes, hindering peer-to-peer communication. By implementing hole punching, we can enable nodes to bypass NAT restrictions and establish direct connections, thereby improving the efficiency and reliability of our decentralized network.

6.2 Encryption

Integrate encryption algorithms to ensure the confidentiality and security of file chunks during both storage and transmission processes. Encrypting file chunks before storing them in the decentralized network ensures that sensitive data remains protected from unauthorized access and tampering. Similarly, encrypting file chunks during transmission provides an additional layer of security, safeguarding data as it moves between nodes in the network. By prioritizing encryption at the file chunk level, we can maintain data integrity and privacy throughout the entire lifecycle of files within our system.

6.3 Interactive GUI

To develop an interactive graphical user interface (GUI) that simplify the interaction of user with the decentralized file storage system. The GUI will be minimal where the user will be able to upload, download and manage files. The major goal of GUI is to hide the complexity of the underlying mechanism that has been implemented to establish the decentralized file storage system. Whether the user be from technical or non-technical background both will be able commute the platform with ease.

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.