St. Pölten University of Applied Sciences

/informatik & security

/fh/// st.pölten

Decentralized Browser-based Cloud Storage: Leveraging IPFS for Enhanced Privacy

Diploma thesis

For attainment of the academic degree of

Diplom-Ingenieur/in

submitted by

Georg Eilnberger, BSc

is211806

in the

University Course IT Security at St. Pölten University of Applied Sciences

Supervision

Advisor: FH-Prof. Dipl.-Ing. Peter Kieseberg

Assistance: -

St. Pölten, January 16, 2024 _____ _____

(Signature author) (Signature advisor)

St. Pölten University of Applied Sciences, Campus-Platz 1, A-3100 St. Pölten,T: +43 (2742) 313 228, F: +43 (2742) 313 228-339, E: office@fhstp.ac.at, I: www.fhstp.ac.at

Declaration

I hereby affirm that

- I have written this thesis independently, that I have not used any sources or aids other than those indicated, and that I have not made use of any unauthorised assistance.
- I have not previously submitted this thesis topic to an assessor, either in Austria or abroad, for evaluation or as an examination paper in any form.
- this thesis corresponds to the thesis assessed by the assessor.

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Date *Signature*

Kurzfassung

Diese Arbeit befasst sich mit dem Bereich der dezentralen Datenspeicherung, wobei der Schwerpunkt auf dem InterPlanetary File System (IPFS) liegt. Sie befasst sich mit dem wachsenden Bedarf an Datenverwaltungslösungen, die Sicherheit, Datenschutz und Benutzerkontrolle inmitten der Einschränkungen herkömmlicher zentralisierter Speichersysteme in den Vordergrund stellen. Das Hauptziel besteht darin, die Effizienz, die Herausforderungen und das Potenzial von IPFS bei der Revolutionierung der Datenspeicherung und -verwaltung zu untersuchen.

Eine Literaturrecherche bildet die Grundlage und beschreibt die Entwicklung von zentralisierten zu dezentralisierten Systemen sowie die technischen Aspekte von IPFS und verwandten Technologien wie Distributed Hash Tables (DHT), Content Identifiers (CIDs), das InterPlanetary Name System (IPNS) und libp2p. Ein wesentlicher Beitrag dieser Arbeit ist die Entwicklung einer Proof-of-Concept-Webanwendung, die IPFS zur sicheren und effizienten Datenverarbeitung einsetzt. Diese Anwendung dient als praktische Veranschaulichung der Integration von IPFS in reale Datenverwaltungsszenarien.

Die Sicherheit und Leistung der Anwendung innerhalb des dezentralen IPFS-Rahmens werden gründlich bewertet. Die Studie hebt die Stärken von IPFS bei der Gewährleistung der Datenintegrität und des Datenschutzes hervor, räumt aber auch die Herausforderungen bei der Skalierbarkeit und Leistung ein, insbesondere bei der Handhabung umfangreicher Dateien und der Bewältigung von Inkompatibilitätsproblemen zwischen WebRTC und TCP-Sockets.

Zuletzt werden in dieser Arbeit Empfehlungen für zukünftige Verbesserungen der Proof-of-Concept Webanwendung gegeben. Dazu gehören die Verbesserung direkter Dateiübertragungsfunktionen, die Weiterentwicklung von Dateiverarbeitungstechniken, die Integration robuster Schlüsselverwaltungslösungen und die Entwicklung dynamischer Datenreplikationsstrategien. Die Forschungsarbeit unterstreicht das Potenzial dezentraler Systeme wie IPFS für die Gestaltung der Zukunft der Datenspeicherung, da sie einen sichereren, privateren und benutzerzentrierten Ansatz bieten.

Abstract

This thesis delves into the realm of decentralized data storage, with a particular focus on the InterPlanetary File System (IPFS). It addresses the growing need for data management solutions that prioritize security, privacy, and user control, amidst the limitations of traditional centralized storage systems. The core objective is to explore the efficiency, challenges, and potential of IPFS in revolutionizing data storage and management.

A literature review lays the foundation, outlining the evolution from centralized to decentralized systems, and detailing the technical aspects of IPFS and related technologies like Distributed Hash Tables (DHT), Content Identifiers (CIDs), the InterPlanetary Name System (IPNS), and libp2p. A significant contribution of this thesis is the development of a proof of concept web application that employs IPFS for secure and efficient data handling. This application serves as a practical illustration of integrating IPFS into real-world data management scenarios.

The security and performance of the application within the decentralized IPFS framework are thoroughly assessed. The study highlights the strengths of IPFS in ensuring data integrity and privacy while acknowledging the challenges in scalability and performance, particularly in handling large files and addressing WebRTC-TCP socket incompatibility issues.[1]

In conclusion, the thesis presents recommendations for future enhancements of the proof of concept web application. These include improving direct file transfer capabilities, advancing file handling techniques, integrating robust key management solutions, and developing dynamic data replication strategies. The research underscores the potential of decentralized systems like IPFS in shaping the future of data storage, offering a more secure, private, and user-centric approach.

Contents

1 Introduction

1.1 Context and Motivation

The field of data storage and access is experiencing a rise in popularity of decentralized models. Centralized data management systems, while established and efficient, present limitations in security, privacy, and user autonomy. This thesis examines the transition towards decentralized systems, with a focus on the InterPlanetary File System (IPFS) and distributed data management principles. The motivation for this study arises from an increasing need for secure, private, and user-centric data management solutions.

1.2 Research Objectives and Questions

This thesis aims to explore the capabilities, challenges, and potential of decentralized data storage systems, particularly IPFS. The research addresses several questions:

- How do decentralized systems like IPFS compare with traditional centralized storage solutions in terms of efficiency, security, and data integrity?
- What are the main challenges associated with the implementation and use of decentralized storage systems?
- How can web applications effectively integrate decentralized systems like IPFS for data management, and what are the associated challenges and security implications?

1.3 Thesis Outline

This thesis is structured into several chapters, each focusing on distinct aspects of decentralized systems and their application in web-based data storage:

- **Chapter 1: Literature Review** explores the evolution of decentralized systems, examines current trends in cloud storage, and provides an in-depth overview of IPFS.

- **Chapter 2: Technical Background** delves into key concepts such as Distributed Hash Tables (DHT), Content Identifiers (CIDs), the InterPlanetary Name System (IPNS), and libp2p.
- **Chapter 3: Design and Development of the Web Application** focuses on the architecture, design principles, and the development process of a proof of concept application utilizing IPFS.
- **Chapter 4: Security and Performance Assessment** assesses the security mechanisms of the application, its performance in the decentralized context, and highlights the encountered challenges.
- **Chapter 5: Future Work and Recommendations** suggests avenues for improvement and potential research directions in the field of decentralized data storage.

The thesis synthesizes theoretical and practical aspects of decentralized systems, offering insights into their application in modern data management.

2 Literature Review

2.1 Evolution of Decentralized Systems and the Web

Decentralization in computing and in the web represents a shift from centralized to distributed control. This shift is not merely technical but also philosophical, highlighting ideas such as autonomy, resistance to censorship, and enhanced robustness against failures or attackers. In the early days of computing, centralized systems dominated due to their simplicity. However, the inherent drawbacks such as single points of failure, scalability issues, and potential for abuse of power led to the exploration of decentralized alternatives.[2]

2.1.1 Historic Look on Decentralization

The concept of decentralization in computing started taking shape with early developments. A pivotal development in this direction was the emergence of Peer-to-Peer (P2P) networks, characterized by their lack of reliance on central servers. Napster, one of the first widely used P2P networks, facilitated file sharing by allowing direct file transfers between users' computers. Despite its legal controversies, Napster demonstrated the potential for efficient, decentralized data distribution. Similarly, BitTorrent further advanced this model, efficiently handling large files and numerous simultaneous uploads and downloads, a significant step toward practical decentralized data sharing. [3]

2.1.2 Blockchains

The advent of blockchain technology represented a critical development in decentralized systems. Blockchain's introduction of a tamper-proof ledger without central authority was first successfully implemented by Bitcoin, the initial decentralized digital currency. This implementation of blockchain technology demonstrated the feasibility of achieving consensus in a trustless environment Subsequently, the development of platforms such as Ethereum expanded the blockchain's applicability. Ethereum introduced functionalities like smart contracts, which allowed for a broader range of decentralized applications, illustrating the versatility and potential of blockchain technology in various domains.[4]

2.1.3 Web 3.0

The concept of a decentralized web, commonly referred to as Web 3.0, aims to transform the internet into a user-centric environment. This concept contrasts with the prevailing centralized web structure controlled by a limited number of entities. Central to this vision are technologies like Ethereum, which provides a platform for smart contracts, and IPFS (InterPlanetary File System), enabling decentralized file storage. These technologies are crucial in the development of Web 3.0, as they offer increased control over data, enhanced security, and greater autonomy for users, thereby marking a shift from traditional web architectures.[4]

2.2 Current Trends in Cloud Storage

Cloud storage represents an integral part of today's digital infrastructure. It has evolved significantly from its origins as simple online data repositories to sophisticated ecosystems encompassing massive data centers. The cloud storage market is largely dominated by a few key players such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud (see figure 2.1). These corporations control a significant portion of the market, offering a wide range of services beyond mere storage, including computing and networking solutions. This does not only reflect their capability to provide scalable and robust services but also highlights a market trend towards centralization. [5]

2.2.1 Challenges with Centralized Cloud Storage

Centralization, however, brings its own set of challenges. From a security perspective, centralized systems can be vulnerable to targeted attacks and can present a single points of failure. Privacy concerns are also prevalent, as users entrust their personal and sensitive data to enviroments controlled by such cloud storage entities.

In response to these challenges, the industry is witnessing the emergence of decentralized cloud storage options. Decentralized storage addresses many of the centralization-related vulnerabilities by distributing data across a network, thereby enhancing security and data privacy. [6]

2.3 IPFS: Overview

The InterPlanetary File System (IPFS) is a protocol designed to create a peer-to-peer method of storing and sharing media in a distributed file system. Developed as a response to the limitations of the traditional,

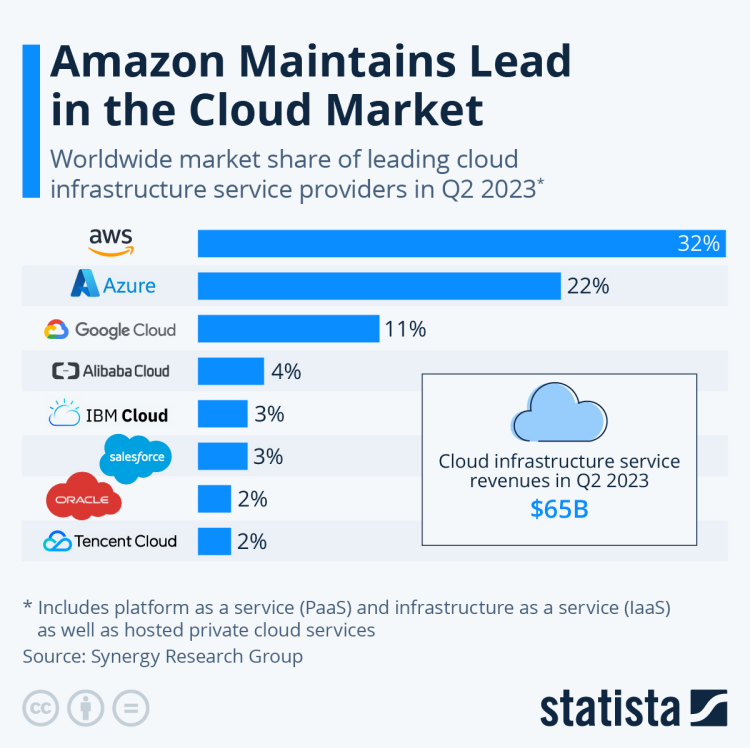

Figure 2.1: Cloud Storage Market Share [5]

centralized web storage model, IPFS represents a paradigm shift in how information is distributed and accessed.[7]

2.3.1 Key Features of IPFS

A defining aspect of IPFS is its decentralized nature. Unlike conventional web storage solutions, which rely on centralized servers, IPFS distributes data across a network of nodes. This distribution of data not only mitigates risks associated with single points of failure but also enhances data accessibility and data permanence.

Central to IPFS's functionality is content addressing. Traditional web uses location-based addressing, for example, URLs pointing to specific server addresses. In contrast, IPFS addresses content through its content itself by utilizing cryptographic hashing. This approach results in unique content identifiers (CIDs), making content retrieval more efficient and less redundant. Compared to location-based addressing, this method significantly enhances both the efficiency and security of data storage and access. By relying on the content's cryptographic hash, immutability is inherent, allowing for verifiable data integrity and contributing significantly to the system's overall robustness.[7]

2.3.2 IPFS Storage and Retrieval

In IPFS, data storage and retrieval operate by using distributed hash tables (DHTs). When a file is uploaded to IPFS, it is split into smaller blocks, hashed, and a unique CID is calculated. This CID can then be used to retrieve the file from the network, allowing for efficient and reliable access. The networking layer of IPFS, capable of working over different network protocols, ensures robust connectivity within the decentralized network, facilitating efficient data transfer between nodes and participants.[7] [8]

2.3.3 IPFS and its Role in Decentralization

IPFS plays a crucial role in the advancement of a decentralized web. By allowing users to store and access data in a distributed manner, IPFS addresses key challenges faced by traditional centralized storage systems, such as data redundancy and inefficiencies in data retrieval. It enhances data sovereignty and privacy, standing as a pivotal tool in the development of decentralized applications (DApps), content distribution networks, and archiving services. These applications utilizing IPFS demonstrate its potential in enhancing the internet, moving towards a more distributed and user-empowered model.[7]

2.4 Security Concerns and other Challenges in Decentralized Systems

Security in decentralized systems presents a unique set of challenges and considerations distinct from those in traditional centralized architectures. The decentralized nature, while offering advantages in terms of redundancy and resistance to certain types of attacks, also introduces specific vulnerabilities that must be addressed. [9] [10]

2.4.1 Security Challenges in Decentralized Systems

Decentralized systems face distinct security vulnerabilities. One key issue is the increased attack surface due to the distributed nature of these systems. Each node in a decentralized network can potentially become a target for attacks. Furthermore in public decentralized systems such as IPFS each participating node can potentially be malicious.

To ensure that data remains unaltered and private over a distributed network robust encryption and validation mechanisms are required. However, implementing these effectively in a decentralized context, where control is inherently distributed, presents unique challenges in itself.[11]

2.4.2 Vulnerabilities in DHT-Based Routing Protocols

Distributed Hash Table (DHT) based routing protocols, such as IPFS, have their own vulnerabilities. These include Sybil attacks, where an attacker tries to create a large number of fake identities/nodes to gain a disproportionately large influence on the network [12][13], and Eclipse attacks, where the attacker isolates a single node or user from the rest of the network, potentially feeding it false and/or harmful information.

2.4.3 Other Challenges in Decentralized Systems and IPFS

Network Reliability: Maintaining a consistent and reliable network is another critical challenge in decentralized systems. In IPFS, for instance, the absence or unavailability of nodes can lead to difficulties in data retrieval, highlighting the need for a robust network health.

Data Persistence and Redundancy: In decentralized systems like IPFS, data persistence is dependent on nodes electing to store that data. Unlike centralized systems where data storage can be systematically managed and guaranteed, IPFS faces the challenge of ensuring that data remains available even when the node originally providing that data go offline. This issue necessitates a redundant storage mechanism and an incentive for nodes to retain data.

2.5 DHT (Distributed Hash Table)

Distributed Hash Tables (DHT) are a fundamental component in various decentralized systems, including peer-to-peer networks and decentralized file storage systems like IPFS. At its core, a DHT provides a key-value store distributed across multiple nodes, typically in a circular network that lacks a centralized control structure. This decentralized nature of DHTs allows for efficient, scalable, and fault-tolerant data storage and retrieval.

The primary purpose of a DHT is to assign ownership of each piece of data to a particular node in the network and to facilitate the efficient location and retrieval of this data when requested. This is achieved through a unique combination of hashing and network algorithms that map 'keys' to 'values' and distribute this mapping across the network.

For example consider a scenario where a network (akin to a DHT) is tasked with storing and retrieving documents. Each document is assigned a unique identifier (key) generated via a hashing algorithm. In this network, various nodes (akin to servers) are responsible for storing these documents. When a request is made to retrieve a specific document, the DHT algorithm efficiently determines which node stores the document based on its key and directs the request to that node

DHTs play a pivotal role in decentralized systems by enabling a decentralized yet organized way of storing and accessing data. Their design principles address several challenges associated with decentralized environments, such as data consistency, load balancing, and fault tolerance. As such, DHTs are not just a storage solution but a foundational technology enabling the efficient operation of large-scale, decentralized networks.

In the context of IPFS, DHTs are instrumental in managing the location of files and resources. They allow IPFS to efficiently locate where content is stored in the network, making the retrieval of files reliable and fast. [14] [15]

2.5.1 Technical Details

Node Structure and Data Distribution

In DHTs, the network is composed of nodes, each with a unique identifier usually derived from the same hash function used for key generation. The DHT utilizes consistent hashing to allocate data across these nodes. Consistent hashing minimizes reorganization when nodes join or leave the network, as only a small portion of keys need to be reassigned. Each node is responsible for storing a specific range of keys. The assignment of keys to nodes is typically done based on the proximity of the key to the node's identifier in the keyspace, which is often represented as a circular ring. [16]

Data Lookup and Retrieval Process

Looking up data in a DHT involves querying nodes to find the one responsible for a specific key. Algorithms like Chord or Kademlia are often employed for this purpose. For instance, in the Kademlia algorithm, each node maintains a routing table of other nodes, sorted into 'k-buckets' based on the distance of their node identifiers. When a lookup is initiated, the querying node asks the closest node to the target key in its routing table. This process is iteratively repeated with closer nodes until the node responsible for the key is found (see figure 2.2). This iterative approach, coupled with the maintenance of k-buckets, ensures efficient routing with a relatively low number of hops, even in large networks.[17] [18] [19]

Fault Tolerance and Redundancy

DHTs try to be fault tolerant and data redundant by replicating data across multiple nodes. The value/data for each key is not only stored on the node responsible for that key-value pair but also on several other successive nodes in the keyspace. This replication strategy ensures that even if a node fails or leaves the

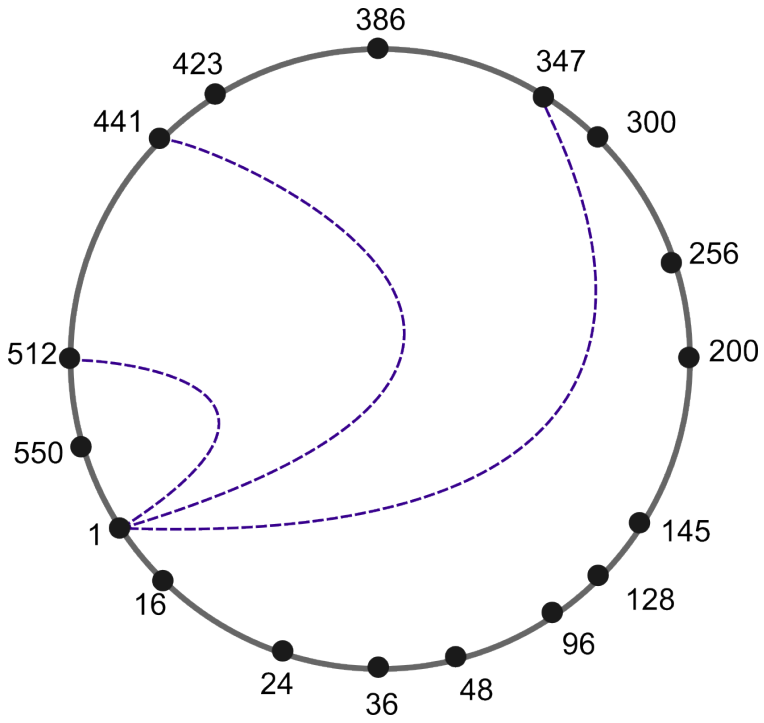

Figure 2.2: DHT Ring with Lookup Process

network, its data can still be retrieved from other nodes that have the replicated data. The number of nodes that replicate the data is called the "replication factor" and is an important design decision that balances data redundancy with resource usage.[20]

DHT in IPFS

IPFS employs a DHT for content discovery, using a variant of the Kademlia algorithm. In IPFS's DHT, each piece of content (aka. a file or a block) is associated with a content identifier (CID) derived from its cryptographic hash. Because files stored via IPFS can be quite large, nodes store index records indicating which node stores which CID rather than storing the content itself. When a user wants to retrieve a file, they query the DHT for a CID, and the DHT routes the query to find the node(s) storing the content.

The customized version of the Kademlia algorithm is designed to adhere to the specific requirements of its decentralized storage system. This adaptation includes optimizations for handling larger data sets typical in file storage, as opposed to the smaller data sets often managed by standard DHTs in other applications. This includes mechanisms to efficiently handle large files and data blocks, splitting large files into smaller blocks, each with its own CID, and effectively managing these blocks within the DHT. This approach allows IPFS to maintain high performance and reliability even when dealing with substantial amounts of data. [8]

2.5.2 Challenges and Limitations of Distributed Hash Tables (DHT)

Handling Network Dynamics

One of the primary challenges in DHT systems, including those employed by IPFS, is managing network dynamics. In a decentralized network, nodes can join or leave without any centralized coordination, which can lead to inconsistent network states. Ensuring data availability and reliability in this ever-changing environment is a key challenge. This is particularly relevant in IPFS, where the network must continuously update the distribution of content as nodes join and/or leave the network.

Sybil Attacks

In the context of DHTs, Sybil attacks pose a significant threat due to the inherent trust in node identity. In these attacks, a malicious actor creates numerous fake identities or nodes in the network. Technically, this can be achieved with relatively low computational effort, as generating fake identities does not require significant resources. The attacker then uses these identities to disproportionately influence the network, which could involve corrupting the routing process, misdirecting or intercepting queries, or even overwhelming the network with false data (see figure 2.3). In the example shown in 2.3 the attacker has control over a large amount of nodes in the network resulting in the disruption of the routing process. This is particularly problematic in IPFS's DHT structure, where trust in node identity underpins the routing and data storage mechanisms. A way to address this attack is to favor long lived peers. [12][13]

Eclipse Attacks

Eclipse attacks in DHTs involve an adversary controlling a significant portion of a target node's view of the network. The attacker strategically positions malicious nodes around the target node in the network's topology. Through this positioning, the attacker can effectively 'eclipse' the target node, filtering or altering its view of the rest of the network. In technical terms, this could mean manipulating routing tables or providing false responses to queries, leading the target node to store, retrieve, or route data incorrectly. In IPFS, where data retrieval is dependent on the integrity of the network's routing information, Eclipse attacks can significantly disrupt operations. The example shown in 2.4 shows the node '347' being 'eclipsed' by the attacker, consequently all routing requests to the target's node are blocked by the malicouis nodes. Addressing these attacks requires sophisticated network monitoring and anomaly detection techniques to identify and isolate malicious nodes, and robust routing table management to prevent attackers from dominating a node's perspective of the network. Such an attack has been successfully executed on the IPFS network before. [11]

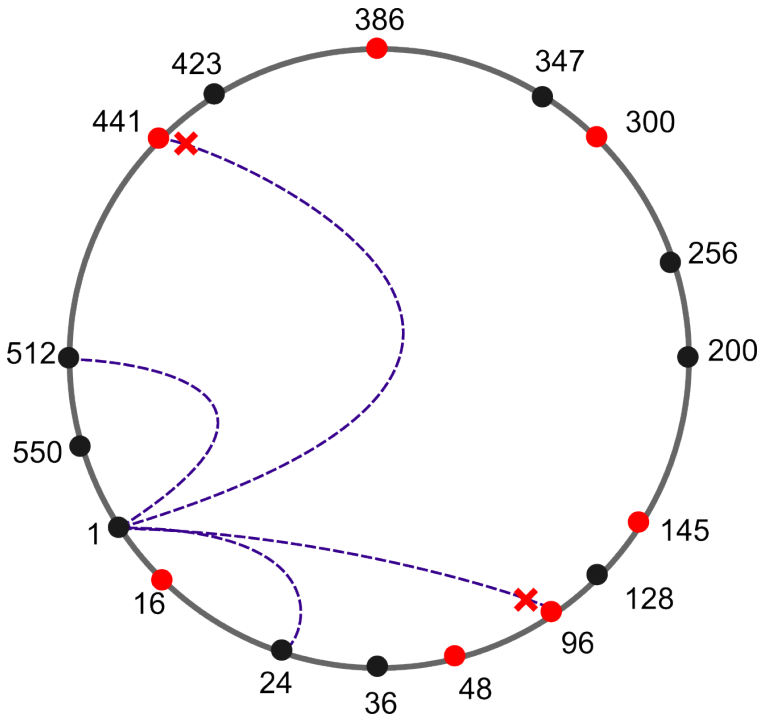

Figure 2.3: Sybil Attack disrupting DHT Routing

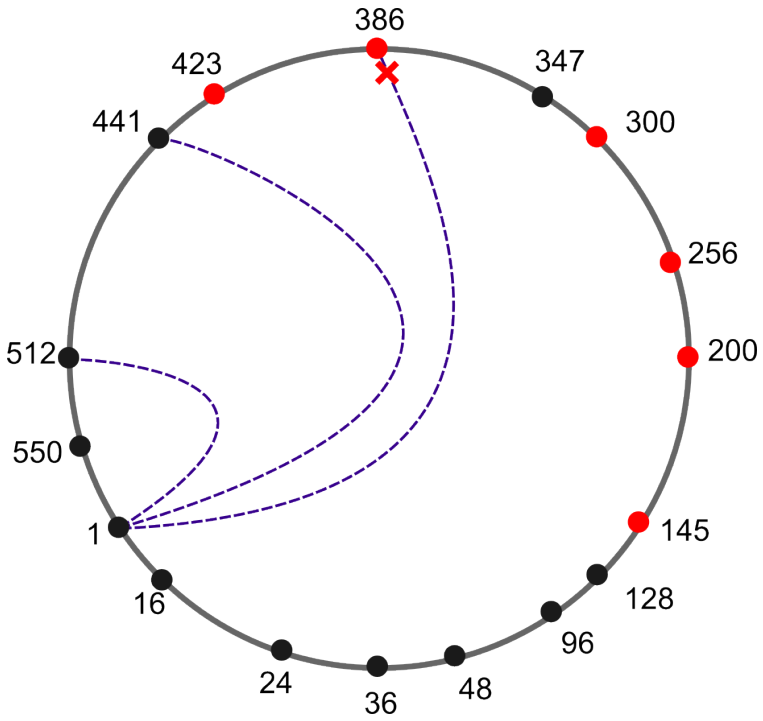

Figure 2.4: Eclipse Attack making a Node inaccessible

Scalability and Performance

Scalability remains a challenge for DHTs, especially in the context of large networks like IPFS. As the network grows, maintaining efficient routing tables and ensuring prompt data retrieval can become increasingly challenging. The balance between maintaining a comprehensive view of the network and the overhead required for updating and managing routing information is a key area of concern.

Data Redundancy and Persistence

Ensuring data redundancy and persistence is another challenge. In decentralized systems, there is no guarantee that a node responsible for storing particular data will remain online. Strategies for replicating and caching data across multiple nodes must be implemented, but these increase the complexity and resource requirements of the network. Another approach is to incentivize node uptime by compensation via crypto currencies, such a strategy is employed by Filecoin network. [21]

3 Technical Background

3.1 DHT (Distributed Hash Table) in IPFS

In the IPFS (InterPlanetary File System), the Distributed Hash Table (DHT) serves a pivotal role in content discovery and routing, functioning as an essential directory and navigation tool within its decentralized framework. The DHT in IPFS is distinct from traditional models, customized to efficiently manage key-value pairings specific to the network's operations. These pairings encompass provider records, linking data identifiers to content-hosting peers; IPNS records, connecting IPNS keys to content paths; and peer records, mapping peer IDs to network addresses for connectivity. This specialized implementation of DHT in IPFS is fundamental not only for data storage but also for enabling effective content location and retrieval across its decentralized network. [14] [15]

3.1.1 Kademlia Algorithm in IPFS

The Kademlia algorithm establishes an efficient and scalable framework for content routing within the decentralized IPFS network. It employs a unique address space, encompassing all possible peer identifiers from 0 to $2^{256} - 1$, and a metric for organizing these peers based on their hashed identifiers. This structure facilitates a virtual linear arrangement of nodes, allowing for streamlined search and retrieval operations. [18] [22]

IPFS's adaptation of Kademlia leverages a mechanism that assigns data to nodes in proximity to the hashed key of the data. The algorithm's design counters the inherent instability of peer-to-peer networks, where nodes may frequently join or leave, by maintaining multiple links to nodes at varying distances, defined by the parameter K. This approach not only ensures efficient routing but also contributes to the network's resilience and fault tolerance. The system parameters, including the value of K, are carefully chosen based on network dynamics, aiming to maintain a balance between data availability, query latency, and network stability. This meticulous configuration of Kademlia within IPFS underscores its capability to manage large-scale decentralized data storage and retrieval with notable efficiency. [16]

3.1.2 Addressing Undialable Peers in IPFS's DHT

In the context of IPFS's DHT, addressing the challenge of undialable peers is critical for maintaining network coherence and efficiency. Undialable peers, typically caused by network address translators (NATs) and firewalls, present a significant obstacle in a DHT, as they hinder the linearity and reachability assumptions inherent in the Kademlia algorithm. This can lead to network fragmentation, where data becomes accessible only to a subset of peers.

To mitigate this, IPFS employs libp2p's AutoNAT service, a distributed mechanism functioning similarly to "Session Traversal Utilities for NAT" (STUN). AutoNAT assesses peers' dialability by analyzing their observed addresses. Based on this assessment, peers dynamically adjust their participation in the DHT: only publicly dialable nodes actively engage as DHT servers capable of both querying and responding to DHT queries, while undialable nodes operate in a client mode, limiting their role to querying the DHT.

This dynamic mode switching, governed by AutoNAT, significantly enhances the DHT's overall robustness. Peers that are not publicly dialable, either due to changing network conditions or inherent network restrictions, transition to client mode, thereby reducing the incidence of dead-ends in content lookups and improving the DHT's response reliability. Furthermore, IPFS nodes offering AutoNAT services facilitate a more distributed approach to NAT traversal, reducing dependency on central infrastructure and promoting a more resilient and self-sustaining network.

However, challenges persist despite these mechanisms. The reliance on dynamic assessment of dialability means that the network's topology can be fluid, and IPFS must continuously adapt to changes in peer availability and connectivity. Moreover, ensuring that all DHT participants accurately assess their public reachability is crucial to prevent the inclusion of undialable nodes in the DHT, which could otherwise degrade the network's overall performance. [23][7]

3.1.3 Dual DHT Networks: WAN and LAN

The IPFS ecosystem incorporates a dual Distributed Hash Table (DHT) system, comprising two distinct networks: the Wide Area Network (WAN) DHT and the Local Area Network (LAN) DHT. This addresses the diverse needs of various IPFS users, some of whom operate in segregated or isolated network environments like local networks or VPNs, where standard DHT operations tailored for public internet settings may not be effective. [23][7]

WAN DHT: The WAN DHT, designed for public internet use, follows strict criteria for peer acceptance, primarily focusing on public address reachability. Peers in the WAN DHT are expected to have public IP

addresses, enabling wide accessibility and ensuring that the network nodes are broadly reachable. This network employs the DHT protocol ID '/ipfs/kad/1.0.0'. Nodes in this DHT dynamically switch between client and server modes based on their public dialability status, determined through AutoNAT assessments. This dynamic switching ensures that only peers capable of full participation and contribution to the DHT network are active as servers, thereby enhancing the effectiveness and efficiency of the DHT.

LAN DHT: Conversely, the LAN DHT is designed for non-public network environments. Nodes in this DHT are typically part of private networks and are not publicly dialable. The LAN DHT uses a different protocol ID, '/ipfs/lan/kad/1.0.0', to prevent accidental intermingling with the WAN DHT. Its primary criterion for peer acceptance is non-public addressability, making it suitable for closed or local network scenarios. Unlike the WAN DHT, nodes in the LAN DHT consistently operate in server mode, as the assumption of public dialability is not applicable in these environments. This configuration is crucial for local or private networks where none of the nodes are publicly dialable, ensuring that the DHT remains functional within its confined scope.

3.1.4 Routing Table Dynamics in IPFS's DHT

The Distributed Hash Table (DHT) in IPFS employs sophisticated routing table mechanisms to manage network connections and facilitate efficient data location and retrieval. [23][7]

Routing Table Structure: In IPFS's implementation of the Kademlia DHT algorithm, each peer maintains a routing table comprising several 'buckets'. These buckets are structured to store information about other peers in the network, organized based on their proximity in the DHT's address space. The address space is defined by a 256-bit space, where each peer is uniquely identified by a hash value. The proximity between peers is determined by their XOR distance in this space, enabling a structured and efficient approach to data routing. [18] [22]

Bucket Management: Each bucket within the routing table is designed to hold a fixed number of peer entries, typically up to 20 peers per bucket. This constraint helps manage the table's size and ensures a balanced distribution of peer information. The buckets are indexed based on the distance of the peers' identifiers from the peer owning the routing table. When a new peer is discovered, IPFS attempts to place it in the appropriate bucket based on this distance metric.

Peer Qualification and Replacement: For a peer to qualify for inclusion in a routing table, it must meet specific criteria. These include adherence to the DHT protocol and ensuring that the peer's network addresses are compatible with the DHT's operational scope (public or private network). If a bucket is full, IPFS employs a replacement policy to maintain an up-to-date and reliable set of peers. Peers that are unresponsive or less useful based on certain heuristics (such as response time or stability) are replaced with newer peers, ensuring that the routing table remains dynamic and reflective of the current network state.

Table Refresh Mechanism: To maintain accuracy, IPFS periodically refreshes the routing table. This refresh process involves probing the network to discover new peers and update bucket contents. The refresh frequency is carefully chosen to balance the need for up-to-date information with the overhead of network traffic. Typically, a refresh occurs every few minutes, a frequency determined based on empirical observations and network dynamics. During a refresh, IPFS may initiate lookups for random peer identifiers to populate the buckets with a diverse set of peers, enhancing the network's resilience and routing efficiency.

The routing table's dynamic nature in IPFS's DHT is crucial for the network's robustness and scalability. By continuously updating peer information and adapting to network changes, the DHT ensures efficient data discovery and retrieval across the decentralized network.

3.1.5 The Lookup Algorithm in IPFS

At its core, the lookup algorithm in IPFS's DHT seeks to answer the query: "What are the K closest peers to a given identifier X?" This query is central to the functioning of the DHT, as it facilitates the discovery of peers that are likely to have, or know about, the requested data. The algorithm operates within the Kademlia-based DHT framework, leveraging its structured network topology for efficient peer discovery.

The process begins by selecting the K closest known peers to the target identifier X from the peer's routing table. These peers form the initial active set for the query. IPFS then iteratively queries these peers for closer peers to X. In each iteration, IPFS allows up to a specified number of concurrent queries, optimizing the balance between network traffic and speed of discovery. As responses are received, the query queue is updated with closer peers, and the next set of queries is initiated.

The query process continues until the closest known peers to X have been successfully queried. Successful queries imply that there were no dial timeouts or errors[23][7]

3.1.6 Record Types in IPFS's DHT

The Distributed Hash Table (DHT) in the InterPlanetary File System (IPFS) manages various types of records, each serving distinct purposes within the network. The mechanisms for storing and retrieving these record types are tailored to their unique characteristics and roles in IPFS's content addressing and discovery model.[23][7]

Provider Records Management: Provider records in IPFS's DHT associate data identifiers, typically multihashes, with peers that store the corresponding content. To store a provider record, the DHT performs a lookup for the K closest peers to the hash of the content's identifier. The record is then stored at these peers. Retrieval involves a similar lookup process, extended with a request for the provider record corresponding to the sought-after data identifier. This system ensures efficient location of content providers across the decentralized network.

IPNS Records Handling: IPNS records, linking IPNS keys to IPNS records (signed and versioned pointers to content paths), are managed distinctively. To store an IPNS record, the DHT identifies the K closest peers to the hash of the IPNS key and stores the record at these locations. During retrieval, in addition to finding the closest peers, the DHT also retrieves the IPNS record. IPFS prioritizes newer records, identified by higher IPNS sequence numbers, to ensure that users access the latest version of the content.

Peer Records Strategy: Peer records, mapping a peer ID to its reachable multiaddresses, are handled implicitly within the DHT. As peers in IPFS's DHT connect, they automatically exchange such information. The process of locating a peer involves a DHT lookup for the K closest peers to the hash of the peer's ID, coupled with a request for the peer's record. The goal is to establish a direct connection to the peer as soon as its address information is obtained.

3.2 Content Identifiers (CIDs) in IPFS

Content Identifiers, commonly known as CIDs, are a fundamental aspect of decentralized data storage systems such as the InterPlanetary File System (IPFS). CIDs are distinct from traditional location-based addressing mechanisms, offering a unique approach to data identification and retrieval.
CIDs in decentralized networks, like IPFS, are used for identifying and locating data. They differ from conventional web addresses, which point to a physical server location, as CIDs reference the content itself.

This method enhances data integrity and security; any alteration in the content leads to a change in the CID, thereby maintaining the original state of the data. [8] [7]

3.2.1 The Anatomy of a CID

A CID's structure is crucial to its function within IPFS, encapsulating several key elements[7]:

1. **Cryptographic Hash Function:** The core of a CID, this hash is immutable and unique, generated by algorithms like sha2-256, with flexibility to accommodate various cryptographic methods.
2. **Multihash Format:** Enhancing CIDs with future-proofing capabilities, the multihash format includes metadata about the hash's type and length, adhering to a Type-Length-Value (TLV) scheme.
3. **Multicodec Prefix:** This specifies the data's encoding format, a crucial factor for data processing across different decentralized systems, ensuring interoperability.
4. **Multibase Prefix:** Introduced in CIDv1, this prefix denotes the base encoding used for string representation, expanding CIDv1's adaptability across various technical environments.

3.2.2 Multihash

Multihash augments the cryptographic hash with vital metadata. The fundamental design of multihash is rooted in accounting for the evolving landscape of cryptographic algorithms. By embedding the type of cryptographic algorithm and its length within the hash, multihash endows CIDs with a level of adaptability and longevity, crucial for maintaining their relevance and security in a rapidly advancing technological environment. Multihash adheres to the Type-Length-Value (TLV) format, comprising three distinct components: the type of cryptographic hash used, its length, and the hash value itself. This structure ensures that every CID carries a detailed description of its cryptographic foundation.

Moreover, multihash's standardized approach to cryptographic hashing promotes interoperability across various decentralized systems. This uniformity is essential for ensuring a consistent method of data identification and retrieval. In the field of cryptography, where algorithms may become obsolete or compromised, multihash provides CIDs with the necessary resilience to adapt to new cryptographic standards. This adaptability is a key factor in preparing decentralized systems against evolving digital security threats.[7]

3.2.3 Transition from CIDv0 to CIDv1

CIDv0, the initial version of the Content Identifier used in IPFS, represented a significant step in the development of decentralized data storage. However, its structure and design posed limitations that necessitated the evolution to CIDv1.

CIDv0 was designed to be a simple and effective way to identify content in a decentralized manner. It utilized a single hash function, sha2-256, to generate a cryptographic hash of the content. This hash was then encoded using base58btc, a binary-to-text encoding scheme. The key characteristics of CIDv0 included:

- **Fixed Hash Function:** CIDv0 was bound to the sha2-256 hash function, which, although secure and widely adopted, limited flexibility in terms of cryptographic advancements.
- **Base58btc Encoding:** This encoding was chosen for its compactness and URL-friendliness but limited the CID's usability in varied technical environments, especially those that required different encoding standards.
- **Lack of Self-Descriptiveness:** CIDv0 did not include information about the hash function or encoding used, making it less adaptable to changes and future enhancements.

CIDv1 addresses these constraints by introducing a multicodec prefix, which explicitly defines the data encoding format. This adaptation enables the support of a diverse range of data formats, increasing interoperability. A significant change in CIDv1 is its support for multiple base encodings. Contrasting with CIDv0s restriction to base58btc, CIDv1 allows the use of various base encodings like base32, broadening its applicability across different systems and requirements. [7]

3.2.4 Key Components of CIDv1

CIDv1 incorporates several advancements over its predecessor:

- **Multicodec Prefix:** It specifies the data format, essential for processing diverse data types within IPFS. For example, the binary sequence '01110000' in Figure 3.1 corresponds to the 'dag-pb' codec.
- **Version Prefix:** This distinguishes between CID formats, ensuring compatibility and facilitating smooth evolution of the CID specification.
- **Multibase Prefix:** It expands the encoding capabilities of CIDs, allowing for adaptability in various systems.

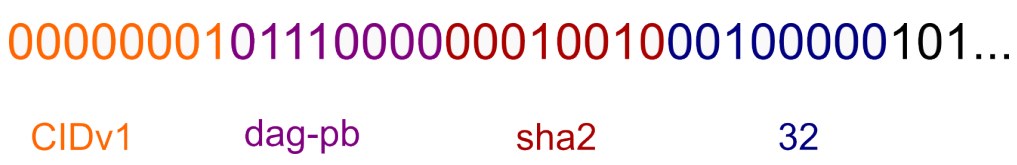

Figure 3.1: An example of a Multicodec Prefix in CIDv1.

3.3 InterPlanetary Name System (IPNS)

3.3.1 Mutability in IPFS

Content addressing in IPFS, which generates unique hashes for file data, inherently leads to immutable content identifiers (CIDs). However, scenarios often require mutable pointers to content-addressed data that need regular updates. The InterPlanetary Name System (IPNS) addresses this need by creating mutable pointers, or **names**, that link to CIDs. These pointers act as updateable links, retaining the verifiability of content addressing while introducing the flexibility of mutability. [24] [7]

3.3.2 How IPNS Works

IPNS Keys and Types

IPNS employs various key types for cryptographic operations, with Ed25519 being the default for its efficiency and security. Legacy support for RSA is maintained for interoperability with older IPNS names. Optional support for Secp256k1 and ECDSA is available, though not universally resolved in the public IPFS swarm. IPNS keys are serialized using a protobuf format containing a KeyType and the key data. This format is integral to ensuring consistent key handling across different implementations.
An IPNS name, represented as a Multihash of a serialized PublicKey, typically appears as a CIDv1 with a libp2p-key multicodec. It's advisable to use a case-insensitive, base36 multibase encoding for the IPNS name and prefix it with /ipns/ for clarity. [24] [7]

IPNS Record Structure

An IPNS record comprises several fields [24]:

- **Value:** Points to a mutable or immutable content path, like /ipns/{ipns-key}, /ipns/example.com, or /ipfs/{CID}.
- **Validity Type and Validity:** Determines the conditions under which the record is valid, usually an expiration date.
- **Sequence and TTL:** Indicates the version of the record and suggests how long the record should be cached.
- **Public Key:** Necessary if the key isn't embedded in the IPNS name, especially for larger keys.
- **Signature:** Cryptographically verifies the record's authenticity.
- **Extensible Data:** Encoded in DAG-CBOR, allowing for flexible augmentation of the record.

- **Record Serialization:** IPNS records are serialized using a strict subset of CBOR, named DAG-CBOR, ensuring a deterministic and interoperable format.

IPNS Protocol Operations

The IPNS protocol encompasses the following operations [24]:

- **Record Creation:** Involves constructing the IPNS Entry protobuf, signing it, and ensuring it adheres to size limits.
- **Record Distribution:** Utilizes various routing systems like DHT or PubSub, each offering trade-offs in speed and consistency.
- **Record Resolution:** Entails fetching the record using its unique identifier, verifying its authenticity and validity.

3.3.3 Creating an IPNS Entry

The process of creating an IPNS entry involves several steps, from generating the cryptographic keys to finally serializing and signing the IPNS record. Here's a step-by-step breakdown [24]:

Key Generation and Serialization

Generate Key Pair: Start by generating a cryptographic key pair. Ed25519 is the default key type. For backward compatibility, RSA keys may also be used.

Serialize Public Key: Serialize the public key using a protobuf format. The serialization includes the key type and the key data, ensuring consistent handling across implementations. For small keys like Ed25519, the public key can be embedded in the IPNS name itself.

Constructing the IPNS Record

Create IPNS Record Structure: Construct an IPNS record with essential fields. The primary components are the value (the content path it points to), validity type (usually expiration date), sequence number (representing the version), TTL (cache duration hint), and optionally the public key if not already embedded in the IPNS name.

Populate Record Fields:

- *Value:* Assign the content path (e.g., /ipfs/{CID} or another /ipns/{key}) to the record.
- *Validity Type and Validity:* Set the validity type (typically 0 for expiration date) and the expiration date in RFC3339 format.

- *Sequence:* Initialize or increment the sequence number for version control.
- *TTL:* Suggest a duration for caching the record (default is often one hour).
- *Public Key:* Include if it can't be derived from the IPNS name.
- *Encode Record to DAG-CBOR:* Encode the populated fields into a subset of CBOR named DAG-CBOR.

Signing the IPNS Record

Prepare Data for Signing: Concatenate the ipns-signature: prefix (in bytes) with the raw CBOR bytes of the IPNS entry data.
Sign the Record: Use the private key corresponding to the public key in the IPNS record to sign the concatenated data. The resulting signature is proof of the record's authenticity.
Complete the IPNS Entry: Attach the generated signature to the IPNS Entry protobuf as signatureV2. Also, include the serialized public key if it's not inlined in the IPNS name.

Finalizing the IPNS Entry

Validate Record Size: Ensure that the serialized IPNS Entry, including the signed data and the public key (if present), adheres to the recommended size limit (typically up to 10 KiB).
Serialize the IPNS Entry: Serialize the complete IPNS Entry, including the DAG-CBOR encoded data, the public key (if applicable), and the signature, into a protobuf format. This serialized form is used for storing and distributing the IPNS record.
By following these steps a verifiable IPNS entry that can be published and resolved within the IPFS network is created, ensuring mutable references to immutable data. [24]

Publishing the IPNS Entry

Once the IPNS entry is created and signed, the next step is to publish it to the network, making it accessible to other peers. Publishing an IPNS entry involves disseminating the entry through the network's routing system, which could for example be the Distributed Hash Table (DHT) in IPFS. Here's a brief overview of the publishing process:

- **Routing System Selection:** Choose a routing system for publishing the IPNS record. The default in many IPFS implementations is the libp2p Kademlia DHT.
- **Prepare the Record for Publishing:** Convert the IPNS entry's name to the appropriate format based on the chosen routing system. When using the DHT, the IPNS entry's name (derived from the public

key's hash) is used as the key in the DHT.

- **Publish to the Network:** Use the IPFS node's publish function to disseminate the record. In the case of the DHT, this involves distributing the record to peers responsible for storing information about the hashed key.
- **Handle Record Propagation:** The network's routing system takes over the propagation of the IPNS record. In the DHT, peers store the record and respond to queries from other nodes looking for that specific IPNS name.
- **Periodic Republishing:** Due to the ephemeral nature of the records in the DHT, it's necessary to periodically republish the IPNS record to keep it alive in the network. This interval is typically shorter than the record's validity to ensure continuous availability.

Through this process, the IPNS entry becomes discoverable and resolvable by other peers in the IPFS network, allowing them to retrieve the latest content the IPNS name points to, even as the underlying data changes over time. [24] [7]

3.4 libp2p

Libp2p, short for "library peer-to-peer," is a modular framework designed for building peer-to-peer (P2P) network applications. This collection of protocols, specifications, and libraries is essential for facilitating peer-to-peer communication among network participants, commonly referred to as "peers." Originating as part of the InterPlanetary File System (IPFS) project, libp2p has evolved into an independent networking stack, applicable across various projects and environments.[25] [26] [27]

The development of libp2p within IPFS was driven by the necessity to overcome the challenges of traditional P2P networking. Prior to libp2p, P2P applications often had to craft their own distinct solutions for peer discovery and connectivity, leading to a lack of standardized, reusable protocols. This fragmentation in the P2P landscape was a significant motivator behind the creation of libp2p.

3.4.1 Fundamentals and Transports in libp2p

Libp2p, a versatile peer-to-peer networking framework, is structured around fundamental components and transport protocols that enable flexible and efficient network communication. This subsection delves into the core aspects of libp2p, focusing on its addressing system, peer identity, and transport protocols. [28]

Addressing in libp2p: Central to libp2p's operation is its innovative addressing scheme, the multiaddress or

multiaddr, which allows for encoding multiple layers of addressing information. This system accommodates various transport and overlay protocols, enabling a consistent and future-proof method for handling diverse addressing schemes across different networks. For instance, a multiaddr such as /ip4/10.0.0.1/udp/1337 clearly specifies the protocols (IPv4 and UDP) and their associated addressing details (IP address and port number).

The multiaddr also integrates peer identity, linking a peer's unique identifier (PeerID) with its transport addresses. This combination of location and identity in a single address, such as /ip4/10.0.0.2/tcp/1234/p2p /QmYyQSo1c1Ym7orWxLYvCrM2EmxFTANf8wXmmE7DWjhx5N, enables peers to establish verifiable and direct connections within the libp2p network.

Peer Identity: Each libp2p peer possesses a unique PeerID, derived from its public cryptographic key. This identity serves not only as a unique identifier but also as a mechanism for establishing secure communication channels through public-private key cryptography. The PeerID is crucial for verifying peer authenticity, thereby enhancing the security and trustworthiness of peer interactions.

Transports in libp2p: Libp2p's transport-agnostic nature is one of its defining features. The framework supports various transport protocols, such as TCP, UDP, QUIC, and WebRTC, each with its own mechanisms for listening and dialing. This flexibility allows libp2p applications to operate across different network environments and use cases.

- *Common Transport Interfaces:* The core operations of libp2p's transport protocols are listening and dialing. Listening involves accepting incoming connections on specified addresses, while dialing entails initiating outbound connections. Multiaddresses play a key role in both processes, providing the necessary information to establish connections between peers.
- *Multiple Transports:* Libp2p applications often support multiple transport protocols simultaneously. This capability is managed by the switch component, which coordinates the overall connection process, including protocol negotiation and stream multiplexing. This multi-transport approach enhances the versatility and adaptability of libp2p applications.

Transport Protocols: Libp2p includes support for specialized transport protocols like QUIC and WebRTC, each offering distinct advantages for specific scenarios.

- *QUIC:* QUIC is a transport protocol that provides an encrypted, stream-multiplexed connection over UDP. It addresses several challenges of TCP, such as head-of-line blocking and handshake ineffi-

ciency. QUIC is particularly effective for reducing latency in connection establishment and for supporting multiple streams without the issues inherent in TCP-based multiplexing. [28]

- *WebRTC:* WebRTC is a protocol designed for real-time communication, particularly in browser environments. In libp2p, WebRTC is utilized to establish browser-to-server and browser-to-browser connections. It includes features like peer-to-peer data channels and NAT traversal mechanisms, making it suitable for decentralized applications that require direct browser-based peer interactions. [28]
- *WebTransport:* WebTransport is a newer specification that utilizes QUIC to offer an alternative to WebSocket, allowing multiple streams over a single connection. It reduces the round-trip times required for establishing connections compared to traditional WebSocket, making it a more efficient option for web-based libp2p applications. [28]

3.4.2 Secure Communication in libp2p

Secure communication is a fundamental aspect of peer-to-peer networking in libp2p, ensuring data transmitted between peers is encrypted and authenticated.

Overview of Secure Channels

Before two peers can transmit data over a transport protocol like TCP, QUIC, WebSocket, or WebTransport, the communication channel needs to be secured. While some transports have built-in encryption (e.g., QUIC), others require an additional security handshake to establish a secure channel.
After establishing the transport connection, libp2p negotiates a secure channel using e.g. TLS 1.3. Following the handshake, the next step is to negotiate a stream multiplexer (like QUIC or others) for the connection.
In libp2p, TLS 1.3 is used to secure channels for transports lacking built-in security. The protocol ensures that both parties have derived a key (the TLS master secret) unknown to others and used to encrypt application data. During the TLS 1.3 handshake in libp2p, peers authenticate each other's libp2p peer IDs. The public key is encoded in the TLS certificate, allowing nodes to authenticate non-standard key types like secp256k1. The TLS 1.3 protocol is identified with the protocol ID /tls/1.0.0. [28]

3.4.3 NAT Traversal in libp2p

Network Address Translation (NAT) traversal is a crucial aspect of peer-to-peer communication in libp2p, enabling nodes behind NATs or firewalls to communicate with the outside world. This subsection explores various strategies and protocols employed by libp2p to facilitate NAT traversal.[28]

AutoNAT: AutoNAT (Automatic NAT Detection) is a protocol that helps nodes determine whether they are behind a NAT. By having peers request external nodes to dial their presumed public addresses, nodes can ascertain their connectivity status. Nodes detected to be behind NATs are advised not to advertise unreachable private addresses and may opt to use relay servers instead. AutoNAT's protocol ID is /libp2p/autonat/1.0.0, and it involves exchanging Dial and DialResponse messages.

Circuit Relay: The Circuit Relay protocol in libp2p allows routing traffic between two peers over a third-party relay peer. This is particularly useful when direct peer-to-peer communication is hindered by NAT or firewall restrictions. In this setup, the relay peer acts as an intermediary, forwarding traffic between the source and destination peers. The relay connections are end-to-end encrypted, maintaining privacy and security.

Hole Punching and DCUtR: For scenarios where relayed connections are suboptimal, libp2p employs hole punching techniques. Hole punching is a method used to establish direct connections between nodes behind NATs. This involves predicting external addresses and synchronizing connection attempts. The Direct Connection Upgrade through Relay (DCUtR) protocol in libp2p coordinates hole punching to establish direct peer-to-peer connections without relying on centralized signaling servers.

3.4.4 Peer Discovery and Routing in libp2p

In peer-to-peer (P2P) networks, each node needs to discover and interact with other nodes. Peer discovery is the process of finding other peers in the network, while routing involves locating a specific peer's position in the network.

Kademlia DHT: The Kademlia Distributed Hash Table (Kad-DHT) is a fundamental part of libp2p's peer discovery and routing mechanism. It is a distributed hash table that organizes peers based on the similarity of their keys, utilizing a routing table to achieve efficient lookups. The routing table divides the keyspace into smaller segments, called "buckets," containing nodes with a common prefix of bits in their SHA-256 hash. This structure aids in locating nodes and data in the network, as well as maintaining a healthy network topology.[18] [22]

mDNS: Multicast DNS (mDNS) is another method used in libp2p for local peer discovery. It allows peers on the same local network to discover each other without any additional configuration. Peers broadcast topics they're interested in, and other local peers can respond with their multiaddresses, facilitating easy discovery and connection on local networks.

Rendezvous Protocol: The rendezvous protocol in libp2p is a federated approach to peer discovery. It enables nodes to discover each other by connecting to a common rendezvous point. Although it introduces

a single point of failure and is not fully decentralized, it serves as an effective method for initial bootstrap and ongoing peer discovery. Peers can register their presence at a rendezvous point, allowing others to find them.

3.4.5 Security Considerations in libp2p

While libp2p simplifies the process of establishing encrypted, authenticated communication channels, there are other vital security considerations to ensure the overall integrity and reliability of the network.[28]

Identity and Trust

In libp2p, every peer is uniquely identified by a Peer ID, derived from a private cryptographic key. This mechanism allows for authentication of remote peers, ensuring you are communicating with the intended party and not an imposter. However, libp2p does not provide a built-in authorization framework, as requirements can significantly vary across peer-to-peer systems. Developers must implement authorization using their own models.[28]

DoS Mitigation

DoS (Denial of Service) mitigation involves designing protocols that minimize resource usage and prevent untrusted amplification mechanisms. Key strategies include [28]:

- Limiting the number of connections and streams per connection
- Implementing transient connections
- Using resource management tools like the Resource Manager in go-libp2p
- Rate limiting incoming connections
- Monitoring application performance for signs of attacks
- Responding to attacks by identifying and blocking misbehaving peers, potentially using tools like fail2ban

4 Design and Development of the Web Application

The complete source code for the proof-of-concept application is provided in a repository as an additional deliverable to this diploma thesis.

4.1 Architecture and Design Principles

The web application developed as part of this thesis represents a proof-of-concept for a secure, decentralized file storage system. It operates within the broader ecosystem of IPFS, leveraging the decentralized nature of the platform to offer a novel approach to data storage and access. The application is hosted directly on IPFS, which provides a resilient and distributed hosting solution. This hosting choice aligns with the overarching theme of decentralization, ensuring that the application itself is as robust and distributed as the data it manages.

4.1.1 Security Design Overview

Security is a primary concern in the application's design, particularly due to the decentralized and open nature of IPFS, where interactions with potentially malicious nodes are possible. To address these concerns, the application incorporates robust security measures, including encryption, to protect user data from unauthorized access and manipulation. While the application cannot directly mitigate inherent DHT vulnerabilities like Sybil and Eclipse attacks, it focuses on securing user data against unauthorized access by malicious IPFS nodes through encryption techniques.

Attacker Models and Mitigation Strategies

- Malicious IPFS Nodes: Given the open nature of IPFS, the application may interact with nodes that attempt to access or manipulate user data. To mitigate this, the application employs end-to-end encryption, ensuring that data remains secure and unreadable by unauthorized nodes.

- Data Manipulation Attacks: The possibility of an attacker altering the data in transit is addressed through the use of IPFS's content addressing and the application's encryption mechanisms. The integrity of data is maintained as any alteration in the encrypted data will be detectable due to the change in its CID.
- Eavesdropping Attacks: The risk of data interception is countered by encrypting the data before it is shared or stored on the network. This ensures that even if the data is intercepted, it remains incomprehensible to the attacker.

4.1.2 Connectivity and File Processing Framework

In the current IPFS ecosystem, direct file transfers using the Helia library face challenges due to a WebRTC-TCP incompatibility issue in current IPFS nodes. As a pragmatic approach, the application utilizes third-party HTTP APIs for interactions with storage providers like Filebase or Pinata. This strategy is a temporary solution until the direct transfer of files via IPFS becomes feasible. The use of HTTP APIs as a current means of file handling offers reliable storage and retrieval, albeit with a modest departure from the ideal decentralized model. [1]

4.1.3 User Experience Design in a Decentralized Context

Designing user experiences for decentralized applications presents unique challenges. The application aims to bridge the gap between user expectations, shaped by experiences with centralized services, and the realities of decentralized systems. The application employs a unique file synchronization method that is designed to be intuitive and familiar. This approach helps in making decentralized storage accessible to a broader audience.

4.2 Custom Identity Management and File Synchronization

This chapter elaborates on the unique approach to identity management and file synchronization within the developed web application. The system hinges on the creation and utilization of a user-specific identity file, coupled with a dynamic file indexing mechanism, ensuring secure and efficient interactions with the IPFS network.

4.2.1 Identity File Creation and Usage

The first step is the creation of an identity file for the user. This pivotal process involves:

- Generation of an AES private key, either supplied by the user or automatically generated by the application.
- Requirement for the user to input an API key from a chosen third-party storage provider. This is a temporary measure due to current Helia limitations (WebRTC - TCP incompatibility).

The identity file, essentially a JSON object, encompasses crucial components for user identification and interaction with the IPFS network:

- The IPNS name, pointing to the file index JSON object.
- The user's AES private key.
- The user's third-party API key.

This identity file represents the core of user data portability, enabling access to their IPFS storage from any device by merely transferring this file.

4.2.2 File Structure and Index File Mechanism

Every file, including the index file, adheres to a structured format:

- Composed as JSON objects.
- Contains encrypted data as a Uint8Array.
- Includes the AES-GCM initialization vector it was encrypted with.

Upon uploading the first file, the index file is generated, and an IPNS entry is created to consistently point to the latest CID of this index file. The index file plays a crucial role in the system:

- Structured as a JSON object.
- Contains an encrypted list of all files, each entry detailing:
  - File CID.
  - File name.
  - File size.
  - SHA-256 hash of the file.
  - Optional metadata for enhanced file information (e.g. a timestamp of the last change).

4.2.3 File Retrieval Process

The steps to retrieve a file are as follows:

1. Connect to IPFS and query the CID of the index file JSON object.
2. Download the index file JSON object.

3. Decrypt the file list using the attached AES-GCM initialization vector and the user's AES256 private key.
4. Display metadata of all files contained in the index.
5. On file request, query the specific CID.
6. Download the requested file JSON object.
7. Decrypt the file using the attached AES-GCM initialization vector and the user's AES256 private key.

4.2.4 File Storage Process

When a new file is uploaded or an existing one modified, the following steps are undertaken:

1. Encrypt the file using the user's AES256 private key and a newly generated initialization vector.
2. Create a JSON object for the file, storing the encrypted data and the initialization vector.
3. Upload the file to an IPFS storage provider (using Helia or third-party HTTP APIs).
4. Record the file's CID and metadata, appending it to the index file.
5. Request the storage provider to pin the new file on IPFS.
6. Upon successful pinning, upload the updated index file and request pinning.
7. Update the IPNS entry to reflect the new index file.
8. Unpin the old index file (and the old file if it was an update) after successful IPNS update and new file pinning.

This architecture ensures a secure, user-friendly, and efficient mechanism for managing files on the decentralized IPFS network, addressing current limitations while laying the groundwork for future improvements in direct file transfer capabilities.

4.3 Encryption in the Application

The web application implements AES-GCM, an Advanced Encryption Standard in Galois/Counter Mode, for its data encryption and decryption processes. AES-GCM is chosen primarily due to its integration with the Web Crypto API, along with its Authenticated Encryption with Associated Data (AEAD) properties and widespread hardware acceleration support. This section details the application's encryption strategy and discusses the choice of AES-GCM. [29] [30]

4.3.1 Rationale for AES-GCM

The selection of AES-GCM for encryption in the application is driven by several factors:

- **Web Crypto API Compatibility:** AES-GCM is readily available in the Web Crypto API, facilitating easy implementation in web applications. [30]
- **AEAD:** AES-GCM provides both encryption and data integrity, ensuring data confidentiality and protection against tampering. [29]
- **Hardware Acceleration:** The widespread hardware support for AES enhances encryption and decryption performance, especially important for handling large files.

4.3.2 Encryption and Decryption Process

The application uses two key functions: encryptFileData for encryption and decryptFileData for decryption, as follows:

encryptFileData Function

This function handles the encryption of files:

- Generates a unique AES-GCM key and Initialization Vector (IV).
- Encrypts file metadata and contents.
- Creates an encrypted file object containing the encrypted data and IV.

decryptFileData Function

This function manages the decryption process:

- Decrypts the metadata and contents from the encrypted file.
- Converts decrypted metadata into a readable format.
- Reconstructs the file with its decrypted content.

Core Encryption Methods

The essential methods for encryption and decryption of the application using the Web Crypto API are as follows:

```
// Function to encrypt data using AES-GCM
async function encryptAESGCM(data, key, iv) {
    const encryptedData = await window.crypto.subtle.encrypt(
    { name: "AES-GCM", iv: iv }, key, data
    );
```

```
    return new Uint8Array(encryptedData);
}

// Function to decrypt data using AES-GCM
async function decryptAESGCM(data, key, iv) {
    const decryptedData = await window.crypto.subtle.decrypt(
    { name: "AES-GCM", iv: iv }, key, data
    );
    return new Uint8Array(decryptedData);
}
```

4.3.3 Conclusion and Future Enhancements

In summary, the adoption of AES-GCM aligns with the application's goal of ensuring a secure and efficient environment for data storage and retrieval within the decentralized web framework. The chosen encryption method stands as a critical component of the application's security mechanism.

Future enhancements could include exploring diverse encryption algorithms and more sophisticated key management solutions.

5 Security and Performance Assessment of the Decentralized Web Application

5.1 Security Evaluation

The security evaluation of the web application focuses on assessing its resilience against potential threats and vulnerabilities, particularly in the context of the decentralized IPFS network. This evaluation is anchored in the previously identified attacker models, assessing the effectiveness of the application's security design and encryption mechanisms.

5.1.1 Attacker Model Analysis

The web application was designed with specific attacker models in mind, primarily focusing on safeguarding user data from unauthorized access and manipulation. This section evaluates the application's defense mechanisms against these models:

1. **Malicious IPFS Nodes:** The primary threat comes from malicious nodes within the IPFS network that may attempt to access or tamper with user data. The application's use of AES-GCM encryption effectively counters this threat by ensuring data confidentiality. Encrypted files, even if intercepted, remain inaccessible to unauthorized parties.
2. **Data Manipulation Attacks:** Another concern is the potential for data manipulation during transmission. The self-verifying nature of IPFS CIDs, combined with the integrity assurance of AES-GCM, provides robust protection against such attacks. This dual layer of security ensures that any tampered data is easily detectable.
3. **Sybil and Eclipse Attacks:** While the application does not directly mitigate DHT vulnerabilities like Sybil and Eclipse attacks, it minimizes their impact on user data privacy. The encrypted data stored on IPFS remains secure against these attacks, as the encryption layer acts independently of the underlying DHT's vulnerabilities. [13] [11]

5.1.2 Encryption Efficacy

The use of AES-GCM for encryption plays a crucial role in securing user data. This subsection evaluates the effectiveness of AES-GCM in the application:

- **Data Confidentiality:** AES-GCM ensures that file contents remain confidential. By encrypting data before it is uploaded to IPFS, the application prevents unauthorized access, even if the data is replicated across potentially untrustworthy nodes.
- **Data Integrity:** Alongside confidentiality, AES-GCM provides data integrity checks. This feature is critical in a decentralized setting where data passes through multiple nodes, as it enables the detection and rejection of tampered data.
- **Performance Considerations:** While AES-GCM is computationally efficient due to widespread hardware acceleration support, the application's encryption process is dependent on the user's device capabilities. This can impact performance, particularly for larger files.

Conclusion: The security evaluation reveals that the web application effectively addresses key security concerns within the decentralized IPFS framework. The robust encryption strategy ensures data confidentiality and integrity, mitigating risks associated with decentralized data storage and transmission. The application's current security measures provide a solid foundation, though future enhancements could focus on advanced key management and addressing broader DHT vulnerabilities.

5.2 Performance and Stability Evaluation

This chapter delves into the performance and stability evaluation of the web application, particularly focusing on its efficiency in handling large files, WebRTC and TCP socket compatibility issues, and the implications of these factors on data loss prevention and redundancy strategies.

5.2.1 WebRTC and TCP Socket Incompatibility

One of the primary limitations in the current implementation of the application is the incompatibility between WebRTC and TCP sockets within the IPFS ecosystem. This limitation impacts the stability of the application in the following ways:

- **Connection Limitations:** Due to this incompatibility, the application primarily relies on a few nodes that act as gateways for browser-based interactions. This reliance can lead to bottlenecks and potential points of failure.[1]

- **Reliance on HTTP APIs:** The application currently uses HTTP APIs of third-party IPFS storage providers like Filebase or Pinata for file handling, which, while reliable, deviates from the ideal decentralized model and could impact long-term scalability and decentralization goals. [1]

5.2.2 Efficiency and Large File Handling

The application's current architecture faces challenges in managing large files due to the limitations in splitting and handling large data sets. Key observations include:

- **Encryption and Upload Time:** Large files lead to extended encryption and upload times, constrained by the device's RAM and processing power.
- **Scalability Concerns:** Without the ability to split large files into manageable blocks, the application's scalability is hindered, particularly when dealing with extensive data sets or high-volume storage requirements.

5.2.3 Data Loss Prevention and Redundancy

In addressing the concerns of data loss and ensuring redundancy, the application leverages the inherent strengths of the IPFS network:

- **Decentralized Storage:** Data availability in IPFS is independent of the storage location, allowing for convenient replication and enhanced scalability.
- **Affordable Redundant Storage:** Utilizing crypto-based storage providers like Filecoin offers a cost-effective solution for redundant storage. At the time of writing, the cost of storing 10TB of data on Filecoin ($1.95 USD per month) is significantly lower than traditional cloud storage options like Amazon S3 ($235 USD per month). [31] [32]
- **Collateral-Based Reliability:** Filecoin storage providers are required to provide collateral, adding an additional layer of reliability and commitment to data preservation. [33]

Conclusion: The evaluation of the application's performance and stability highlights key areas for improvement, particularly in large file handling and direct file transfer capabilities. Despite these challenges, the application benefits from the decentralized, scalable nature of IPFS and the cost-effective, redundant storage solutions offered by crypto-based storage providers. Future work should focus on enhancing file transfer capabilities and exploring more efficient file processing methods to bolster the application's performance and scalability within the decentralized web.

5.3 Future Work and Recommendations

This chapter outlines prospective enhancements and areas of improvement for the web application, with a focus on advancing its capabilities within the decentralized file storage domain. The recommendations serve as a blueprint for future development, aimed at refining the application's functionality and aligning it with the evolving needs of decentralized storage solutions.

Enhancing Direct File Transfer Capabilities

Actively track progress in the IPFS community regarding the resolution of the WebRTC-TCP socket compatibility issue. Integrating these developments once available will significantly bolster the application's ability to perform direct file transfers between browsers and IPFS nodes, thereby elevating its efficiency and decentralization. [1]

Advanced File Handling Techniques

Implement file chunking and streaming techniques for more effective handling of large files. This approach will allow the application to break down large files into smaller segments, enabling incremental encryption and upload, which in turn reduces memory usage and enhances user experience.

Improved Key Management Solutions

Investigate integrating hardware security solutions for superior key management. Utilizing such hardware security modules could offer a secure method to store the private encryption key, significantly improving overall security and protecting against various digital threats.

Dynamic Data Replication Strategies

Develop a feature for automated data replication based on user-defined redundancy factors. This feature should harness the power of decentralized storage networks like Filecoin for data replication. Given the affordability of services like Filecoin, where data storage costs are significantly lower than traditional centralized services, this replication strategy is not only feasible but also cost-effective.

Conclusion

The enhancements and improvements proposed in this chapter are pivotal for the evolution of the application in the decentralized storage space. By focusing on these strategic areas, the application can significantly

improve its functionality, user experience, and security, ensuring its effectiveness in a decentralized digital world. The implementation of these recommendations will propel the application towards a future aligned with the principles of a secure, efficient, and user-centric decentralized web.

6 Conclusion

This thesis has provided a detailed examination of the InterPlanetary File System (IPFS) and its application in decentralized data storage, culminating in the development of a proof-of-concept web application. This final chapter summarizes the key findings, reflects on the implications of the research, and suggests directions for future work in this field.

Summary of Findings The comprehensive analysis conducted throughout this thesis highlights the advantages and challenges of using IPFS and related technologies such as DHT, CID, IPNS, and libp2p. It has been shown that decentralized systems like IPFS offer significant improvements in data security, privacy, and autonomy compared to traditional centralized storage models. These systems mitigate common issues such as single points of failure and enhance data integrity with unique content addressing techniques.
The development of the web application provided practical insights into the implementation of IPFS. While demonstrating the effectiveness of decentralized storage, it also brought to light specific technical challenges, including WebRTC-TCP socket incompatibility and difficulties in managing large files. [1]

Reflections on Decentralized Storage This research contributes to the understanding of how decentralized storage systems can impact the future of data management on the internet. Technologies like IPFS represent not only a technical shift but also a move towards a more user-focused model of data ownership and control.

Recommendations and Future Directions Based on the findings, several recommendations are proposed for advancing the field:

1. **Technical Improvements:** Addressing the limitations identified in IPFS, particularly in file transfer protocols and handling of large files, is essential for future development.

2. **Enhanced Key Management:** Investigating more sophisticated encryption key management techniques, including hardware solutions, could significantly improve security.

3. **Automated Data Replication:** Developing mechanisms for user-defined data replication will increase redundancy and reliability, especially using cost-effective storage solutions like Filecoin.

Concluding Remarks The findings of this thesis contribute to the expanding research in decentralized data storage, offering a combination of theoretical insights and practical applications. The potential of technologies like IPFS in reshaping data storage and management is substantial. As the digital landscape evolves, the significance of decentralized systems in protecting data becomes increasingly paramount. Future work in this area promises further advancements towards a secure, efficient, and decentralized digital infrastructure.

List of Figures

Acronyms

AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
API	Application Programming Interface
CID	Content Identifier
DAG	Directed Acyclic Graphs
DApp	Decentralized Application
DCUtR	Direct Connection Upgrade through Relay
DHT	Distributed Hash Table
DNS	Domain Name System
HTTP	Hypertext Transfer Protocol
ID	Identifier
IPFS	InterPlanetary File System
IPNS	InterPlanetary Name System
IT	Information Technology
IV	Initialization Vector
JSON	JavaScript Object Notation
mDNS	Multicast DNS

NAT	Network Address Translation
P2P	Peer-to-Peer
PoC	Proof of Concept
SHA	Secure Hash Algorithm
TB	Terabyte
TCP	Transmission Control Protocol
USD	United States Dollar
WebRTC	Web Real-Time Communication

Bibliography

[1] ipfs-helia. "Helia github repository." Accessed: 14.0.2024. (), [Online]. Available: https://github.com/ipfs/helia/issues/256.

[2] Mats-Åke Hugoson, "Centralized versus decentralized information systems: A historical flashback," in *History of Nordic Computing 2: Second IFIP WG 9.7 Conference, HiNC2, Turku, Finland, August 21-23, 2007, Revised Selected Papers 2*, Springer, 2009, pp. 106–115.

[3] P. Raj, Anupama Raman, D. Nagaraj, and S. Duggirala, "High-performance peer-to-peer systems," in *2015*, 2015. [Online]. Available: https://consensus.app/papers/highperformance-peertopeer-systems-raj/ac24e8e5abed563e9776ecb35bc43996/.

[4] Wenbo Wang, D. Hoang, Peizhao Hu, Zehui Xiong, D. Niyato, Ping Wang, Yonggang Wen, and Dong In Kim, "A survey on consensus mechanisms and mining strategy management in blockchain networks," *IEEE Access*, vol. 7, pp. 22 328–22 370, 2018. DOI: 10.1109/ACCESS.2019.2896108.

[5] Statista. "Worldwide market share of leading cloud infrastructure service providers (as of q2 2023)." Accessed: 11.0.2024. (), [Online]. Available: https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/.

[6] Anwar Ghani, Afzal Badshah, Saeedullah Jan, Abdulrahman A Alshdadi, and Ali Daud, "Issues and challenges in cloud storage architecture: A survey," *arXiv preprint arXiv:2004.06809*, 2020.

[7] IPFS. "Official ipfs documentation." Accessed: 14.0.2024. (), [Online]. Available: https://docs.ipfs.tech/.

[8] Juan Benet, "Ipfs-content addressed, versioned, p2p file system," *arXiv preprint arXiv:1407.3561*, 2014.

[9] Enis Karaarslan and Enis Konacakl, "Data storage in the decentralized world: Blockchain and derivatives," *arXiv preprint arXiv:2012.10253*, 2020.

[10] Nurzhan Zhumabekuly Aitzhan and D. Svetinovic, "Security and privacy in decentralized energy trading through multi-signatures, blockchain and anonymous messaging streams," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, pp. 840–852, 2018. DOI: 10.1109/TDSC.2016.2616861.

[11] Bernd Prünster, Alexander Marsalek, and Thomas Zefferer, "Total eclipse of the heart disrupting the InterPlanetary file system," in *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA: USENIX Association, Aug. 2022, pp. 3735–3752, ISBN: 978-1-939133-31-1. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/prunster.

[12] Liang Wang and Jussi Kangasharju, "Real-world sybil attacks in bittorrent mainline dht," Dec. 2012, pp. 826–832, ISBN: 978-1-4673-0920-2. DOI: 10.1109/GLOCOM.2012.6503215.

[13] JR Douceur, "The sybil attack," Jan. 2002, pp. 251–260, ISBN: 3-540-44179-4.

[14] Sebastian A. Henningsen, Martin Florian, Sebastian Rust, and B. Scheuermann, "Mapping the interplanetary filesystem," *2020 IFIP Networking Conference (Networking)*, pp. 289–297, 2020.

[15] Yongle Chen, Hui Li, Kejiao Li, and Jiyang Zhang, "An improved p2p file system scheme based on ipfs and blockchain," *2017 IEEE International Conference on Big Data (Big Data)*, pp. 2652–2657, 2017. DOI: 10.1109/BigData.2017.8258226.

[16] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, ser. STOC '97, El Paso, Texas, USA: Association for Computing Machinery, 1997, pp. 654–663, ISBN: 0897918886. DOI: 10.1145/258533.258660. [Online]. Available: https://doi.org/10.1145/258533.258660.

[17] Emil Sit, "Storing and managing data in a distributed hash table," 2008.

[18] Petar Maymounkov and David Eres, "Kademlia: A peer-to-peer information system based on the xor metric," vol. 2429, Apr. 2002, ISBN: 978-3-540-44179-3. DOI: 10.1007/3-540-45748-8_5.

[19] Ion Stoica, Robert Morris, David Karger, M. Kaashoek, and Hari Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM Computer Communication Review, vol. 31*, vol. 31, Dec. 2001. DOI: 10.1145/964723.383071.

[20] Salma Ktari, Mathieu Zoubert, Artur Hecker, and Houda Labiod, "Performance evaluation of replication strategies in dhts under churn," vol. 284, Dec. 2007, pp. 90–97. DOI: 10.1145/1329469.1329481.

[21] Filecoin. "Filecoin docs - block rewards." Accessed: 11.0.2024. (), [Online]. Available: https://docs.filecoin.io/storage-providers/filecoin-economics/block-rewards.

[22] Ingmar Baumgart and Sergio Mies, "S/kademlia: A practicable approach towards secure key-based routing," vol. 2, Jan. 2008, pp. 1–8, ISBN: 978-1-4244-1889-3. DOI: 10.1109/ICPADS.2007.4447808.

[23] IPFS. "Ipfs 0.5 content routing improvements: Deep dive." Accessed: 14.0.2024. (), [Online]. Available: https://blog.ipfs.tech/2020-07-20-dht-deep-dive/.

[24] ——, "Official ipns specification." Accessed: 14.0.2024. (), [Online]. Available: https://specs.ipfs.tech/ipns/ipns-record/.

[25] E. Almeida, G. Sunyé, Yves Le Traon, and P. Valduriez, "Testing peer-to-peer systems," *Empirical Software Engineering*, vol. 15, pp. 346–379, 2010. DOI: 10.1007/s10664-009-9124-x.

[26] Gao Li-yan, "A service-oriented net framework for p2p network," *Journal of Henan University*, 2008.

[27] M. Amoretti, "Introducing artificial evolution into peer-to-peer networks with the distributed remodeling framework," *Genetic Programming and Evolvable Machines*, vol. 14, pp. 127–153, 2013. DOI: 10.1007/s10710-013-9182-0.

[28] libp2p-Github. "Documentation site for the libp2p project." Accessed: 14.0.2024. (), [Online]. Available: https://github.com/libp2p/docs/tree/master.

[29] David McGrew, *An Interface and Algorithms for Authenticated Encryption*, RFC 5116, Jan. 2008. DOI: 10.17487/RFC5116. [Online]. Available: https://www.rfc-editor.org/info/rfc5116.

[30] Mozilla Foundation. "Web crypto api." Accessed: 14.0.2024. (), [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Web_Crypto_API.

[31] Amazon Web Services. "Amazon s3 pricing." Accessed: 14.0.2024. (), [Online]. Available: https://aws.amazon.com/s3/pricing/.

[32] Storage.market. "Ipfs storage market." Accessed: 14.0.2024. (), [Online]. Available: https://file.app/.

[33] Filecoin. “Official filecoin documentation.” Accessed: 14.0.2024. (), [Online]. Available: `https://docs.filecoin.io/`.