A Peer-to-Peer Document Distribution Network

Tom Taylor

A report submitted in partial fulfilment of the requirements
for the degree of MSc in Computer Science.

September, 2023

University of London

Abstract

This project aims to research peer-to-peer network technology through the development of a peer-to-peer content distribution and social networking service. This exploration has been undertaken to provide an alternative to corporate-owned social networks and the increasingly centralised control of internet infrastructure.

Peer-to-peer networks offer advantages in the form of decentralisation, self-organisation and efficient resource utilisation. The contribution of network resources by participants eliminates the need for hosting, administration and monetisation of the service. Despite these benefits, peer-to-peer systems have experienced a decline in popularity due to their inherent challenges, such as that of resource location, connectivity and authentication. While cloud computing has emerged to fill the need for scalable and reliable systems, an increase in demand for resilient decentralised systems has reignited research into peer-to-peer technology.

Along with the identification of risks and challenges associated with this area, this project presents a specification for such a system and an evaluation of the implementation. The results reveal that while creating a peer-to-peer social network is feasible, the complexity of such a system should not be underestimated, and highlights the importance of a rigorous testing plan and a well-defined protocol for any future endeavours in this field.

Contents

Chapter 1

Introduction

1.1 Aims and Objectives

This project aims to determine the viability of a peer-to-peer distributed network infrastructure as an alternative to centralised commercially operated social networks. This will be explored through the development of a peer-to-peer content distribution service, which enables users to share items across the internet without the need for a central, dedicated server.

The initial objectives of this work involve investigating peer-to-peer technologies and identifying their distinct characteristics and challenges through comprehensive background research.

This research will inform the outcome of the project, a functional application which:

- Provides "micro-blogging" functionality via a distributed document exchange, in order to demonstrate the workings of a peer-to-peer content distribution system
- Sets out a framework to assess the challenges and feasibility of such a system
- Informs future research and development in the field of peer-to-peer technologies and decentralised social networks.

1.2 Problem Description

Motivation

Contemporary social media companies have recently come under scrutiny for their reliance on targeted advertising, algorithmic manipulation, and data harvesting from users for revenue generation. Such platforms are susceptible to compromise for political, stakeholder, or nation-state interests, raising concerns about user's privacy and control over their own data. High-profile incidents, such as those involving Cambridge Analytica, Elon Musk's takeover of Twitter, or China's state influence on TikTok, have underscored the shortcomings of these systems. This project seeks to explore the implementation of a decentralised, peer-to-peer network service as a potential solution.

Peer-to-peer Networks

Peer-to-peer networks present an alternative to traditional client-server internet services. In these networks, peers interact directly with each other to share resources and services without the reliance on a central authority or server. Common applications of peer-to-peer networks include file-sharing systems (e.g. BitTorrent), real-time communication tools (e.g. Skype or Zoom,) multiplayer gaming platforms, distributed storage systems and decentralised finance.

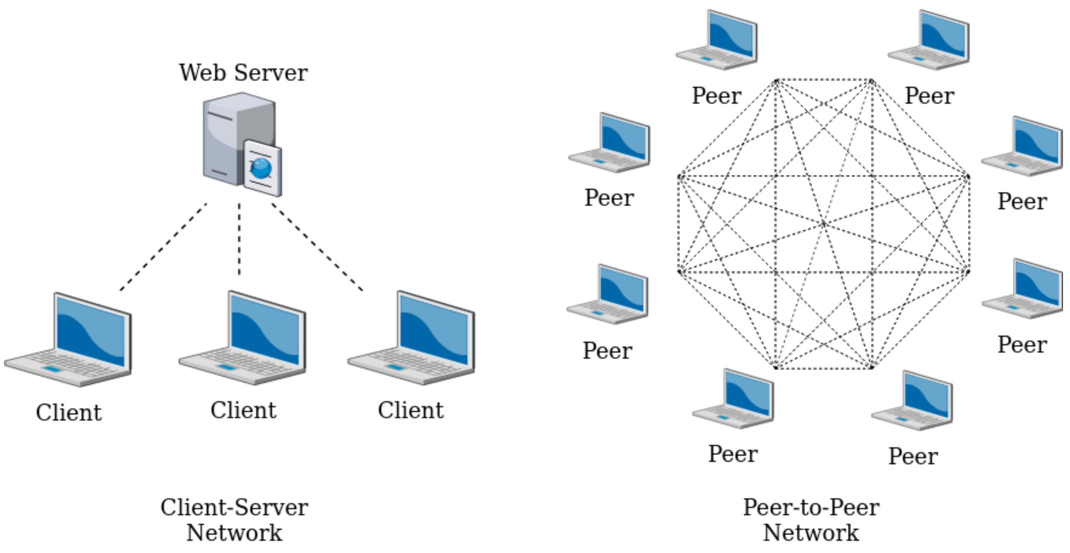


Figure 1.1: *In a peer-to-peer network there is no central server, resources are acquired from other users.*

Peer-to-peer networks provide the following benefits:

- A fully decentralised network which lack a single point of control can offer a resilient and user-centric service.
- The network protocols are self-organising and adaptive, capable of adjusting to changing user demand. Providing inherent load-balancing properties and scalability.
- Peer-to-peer networking harnesses end-users' surplus resources, utilising unused bandwidth, storage, and computation power connected to the network.

- The absence of dedicated hosting and server management eliminates the need to finance the service.

In the early 2000's peer-to-peer networks were vaunted as the next disruptive internet technology due to the advantages that they offer. However, the myriad challenges that they presented, which will be identified in the literature review section, contributed to a decline in their popularity. The rise of cloud computing has instead transformed the industry by removing the need for dedicated hardware configuration, providing availability, scalability and security, but at a cost of centralised ownership over infrastructure and data in the hands of a small number of corporations. With the recent growth of distributed systems, decentralised finance and federated networks such as Mastodon, the internet landscape is possibly shifting once more, and peer-to-peer paradigms could provide a solution.

1.3 Report Overview

This document will provide an analysis of the identified problem area, peer-to-peer networking technology and existing decentralised social networks. A project methodology, software specification and development plan will be outlined, followed by the implementation details, and finally, a presentation of findings and evaluation.

Chapter 2

Background

2.1 Introduction

The following section provides an overview of specific peer-to-peer concepts and a review of existing research in the field of distributed and decentralised systems. This research will explore the main challenges inherent in decentralised networks of this nature and identify selected pre-existing strategies for addressing them. In addition, a brief analysis and summary of noteworthy projects involving existing decentralised social networks will be undertaken, noting their relevance, successes and shortcomings.

2.2 Overlay Networks

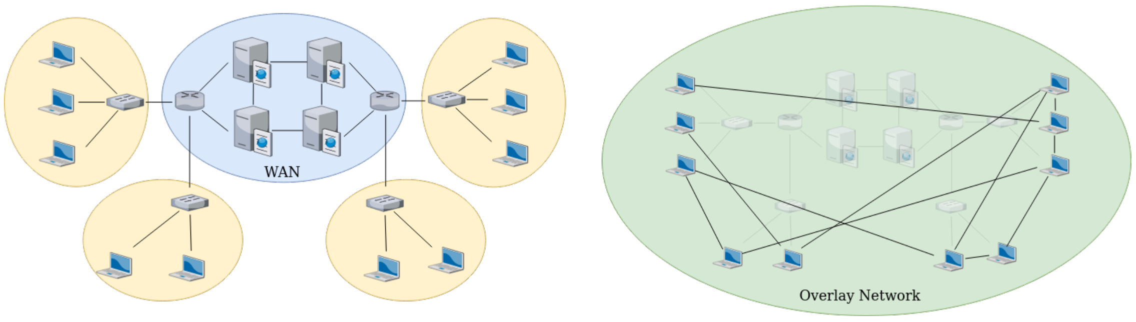


Figure 2.1: *An overlay provides an abstraction which allows the application to ignore the underlying physical topology of sub-networks.*

Resource location poses a notable challenge in peer-to-peer networks, due to the absence of a central index. Resources can be distributed randomly across different nodes. As a result, the network requires a mechanism to identify which node has holds a particular resource and provide means to communicate with nodes to which they don't have a direct connection.[1]

Requiring each node to maintain it's own up-to date index of every peer and item on the network is impractical, or even impossible for large networks. One solution to this issue, adopted in early peer-to-peer systems, is the use of a dedicated server to track the location of nodes and resources, however this creates a single point of weakness which must be maintained and hosted, negating the strengths of a decentralised system.

An overlay network (See Figure 2.1) is an additional layer of abstraction designed to address this issue. An overlay is the protocol which allows nodes to route messages and data, with a minimum of information and connections, regardless of the underlying physical network hardware. Two nodes can be said to be directly connected in an overlay, despite the physical route comprising multiple connections and bridges. Nodes participate in relaying requests or as a distributed node or item look-up service.[2] See [3, 4] for a comparison of common overlay schemes, and [5] for a comparison to federated network topologies, which are a hybrid of centralised and fully peer-to-peer networks.

Overlay networks can be categorised as either structured or unstructured. In unstructured overlays, such as in a gossip or epidemic protocol [6, p. 64-67][7], nodes typically maintain a cache of random neighbours without any specific organisation. By contrast, structured overlays employ a more hierarchical approach, usually partitioning and evenly distributing indexing information to allow for more efficient routing, key-based item look-up and resource mapping.[6, p. 75]

Unstructured Overlays

The simplest method of resource discovery in unstructured overlays is via flooding or breadth-first-search, as used in the original Gnutella protocol.[6, p. p55][8] A node sends requests for a resource to all it's connected peers, who will then forward the query to their connected peers until the resource is found. This deterministic method can incur high network traffic[9], and requires care to prevent redundant messaging and infinite loops.

While more sophisticated searching techniques can be used, such as iterative deepening, nearest neighbour and more effective caching, ultimately the performance is not guaranteed to be optimal, and is best suited for smaller or localised peer-to-peer networks. A thorough review of searching techniques can be found in [10].

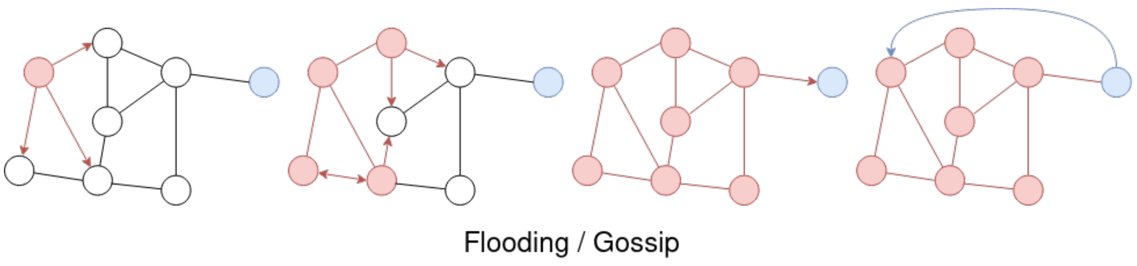


Figure 2.2: *Flooding can rapidly broadcast a message, but at a cost of exponential messaging overhead for a large number of connected peers. In a search, once the target node is located, it responds to the originator.*

Unstructured networks rely on the inherent properties of complex networks: A more popular item will be distributed across more nodes, increasing it's availability. If each node in a network represents a person, and their connected peers their friendship group, then the 'small world' effect means that nodes which are not direct neighbours of each other can still be reached by a number of small hops via these mutual neighbours.[11]

Structured Overlays

Structured overlays provide availability comparable to unstructured networks, but offer significantly improved performance, with fewer inter-node messages and a shorter round trip time required for search. Unlike unstructured overlays, the relationship between peers and the location of data is strictly defined by the particular protocol used.[10] The state space is evenly divided across the network, ensuring greater load distribution and peer equity. They offer a guaranteed lower asymptotic time and space requirement, but at the trade-off of a higher implementation complexity, requiring more intricate design, initialisation and maintenance procedures. A comparison of the advantages and disadvantages of structured and unstructured overlay networks can be found in appendix A.1.

Distributed Hash Tables

One tool for resource location in structured overlays is the Distributed Hash Table, or DHT.[2] Items to be stored are assigned a key through consistent hashing, as in a normal key-value hash table, and allocated to specific nodes in the network. Each node in the network is responsible for storing items within an allocated portion of the finite available hash key-space. In order to retrieve an item, other nodes must locate the node responsible for that item, the way this is performed depends on the algorithm used. Once located, the file can be retrieved via a direct peer-to-peer connection. Distributed hash tables allow a group of peers to create an efficient routing table and look-up service, usually with guaranteed asymptotic performance.[1, 4] Depending on the implementation, the DHT can be deployed as a 'Co-operative File System', or as a DNS-like host discovery service.[3]

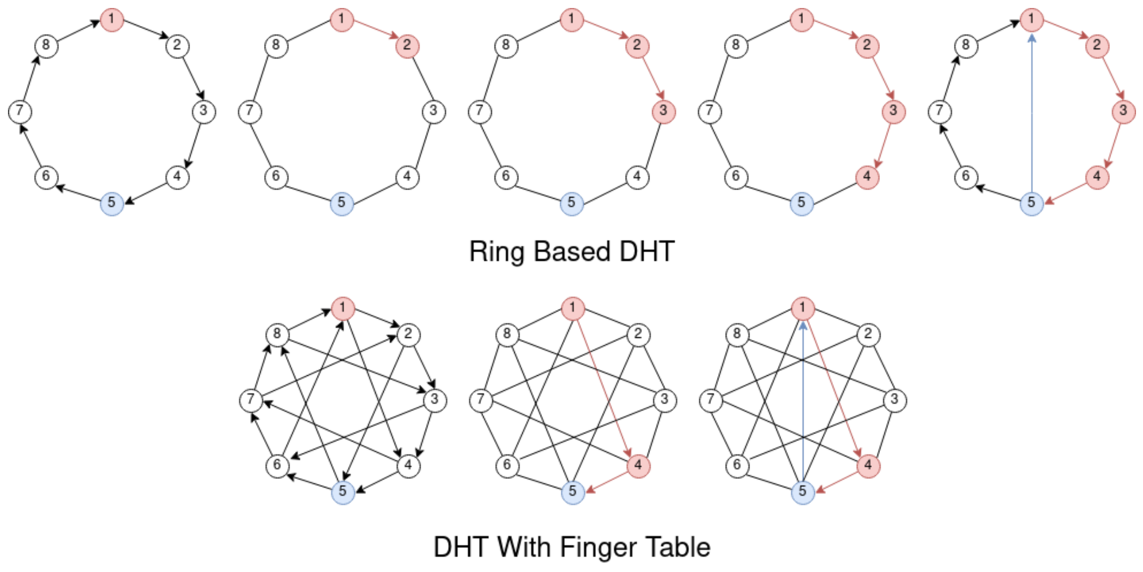


Figure 2.3: *In a Distributed Hash Table, nodes could be arranged in a linked list with each node only aware of it's direct neighbours. However look-up time can be improved if each node stores a limited cache of other node addresses, such as in the Chord algorithm.*

Numerous Distributed Hash Table algorithms have been developed, such as Chord, Kademlia, Pastry, Tapestry and CAN[3], each with different characteristics, strengths and weaknesses. This review will focus on Chord, one of the first Distributed Hash Table algorithms, and Kademlia, which is prominently used in many real-world applications.

Chord

The Chord DHT arranges the nodes in a ring topology and assigns each node a hash ID and a portion of the divided key-space according to it's position in the ring. The hash of the item determines which node is responsible for it's storage. Each node keeps a 'finger table' storing the addresses of its neighbours as well as specific successor nodes spread around the ring, to reduce the number of 'hops' in node look-ups. To find a resource each node forwards a request to the node in it's table closest to that which is responsible for the key-space. A detailed explanation can be found in the original paper [12], and [13] in which the authors describe how Chord could be used to build a peer-to-peer file sharing network.

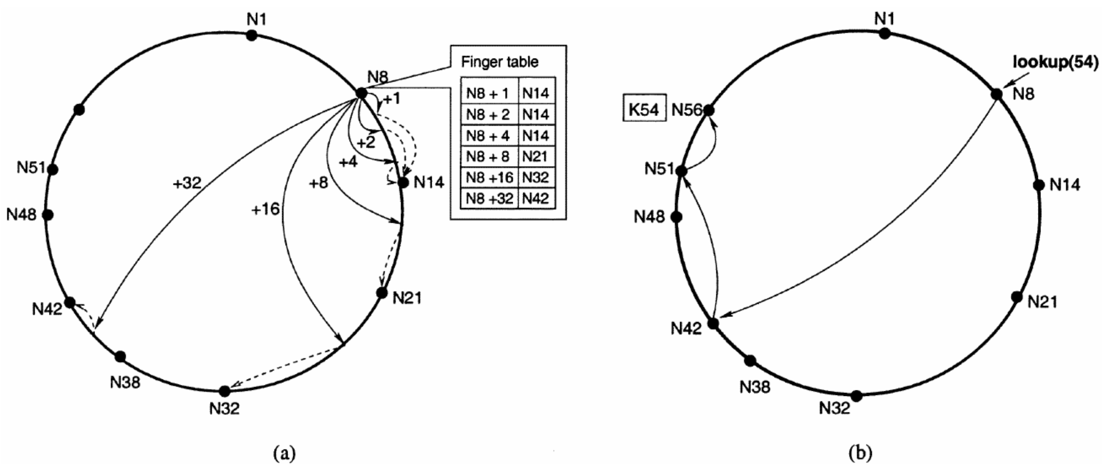


Figure 2.4: *The finger table contains nodes spaced around the ring, to minimise hops. The figure (b) shows a look-up for key 54 at node 56. (Figure taken from the Chord paper[12])*

Kademlia

The Kademlia[14] protocol, operates on similar principles to Chord. Nodes are assigned a random 160-bit identifier and an equal segment of the available key-space, with items to be stored allocated to the node with the ID closest to the hash of the item. The nodes are structured in a binary tree, with the novel use of a XOR-based metric to calculate distances between them. The optimised structure of Kademlia reduces redundant messaging, provides quicker node look-ups and ensures an evenly distributed key-space.[3] Kademlia is used in the BitTorrent "Mainline" DHT, Ethereum blockchain, eDonkey Kad and the InterPlanetary FileSystem.[15, 16, 17, 18]

2.3 Challenges

Bootstrapping and Churn

Without a single static 'gateway' address to connect to or a static map of the filesystem, a more dynamic approach called bootstrapping is adopted to allow nodes to join a peer-to-peer network.[6, p.155-156]

A straightforward bootstrapping approach is to use a static server as an index or tracker[19] to provide a list of peer addresses, although this compromises the decentralised characteristics of a network. Other methods involve each node maintaining a seed cache of known participating nodes, a dynamic broadcast/multi-cast service, such as Rendezvous or mDNS[20] which allows devices on a network to find each other, or for peers to exchange direct invitations/addresses via an outside communication channel.

As the network functionality is dependent on all the participants, it is essential that overlay mechanisms are able to cope with peers continuously joining and leaving a network [21], this is known as peer churn. An analysis of the effects of peer churn on three peer-to-peer networks; Gnutella, BitTorrent and Kad, can be found in [22].

The health of an overlay network can be maintained by the use of a heartbeat, a regular refreshing of connections to ensure each peer is still online. Ensuring that peers announce their intention to connect and disconnect from a network, allows the overlay to preemptively re-distribute resources, and waiting for faults or timeouts to occur.

Incentive

As network functionality is dependent on the peer's resources, peer-to-peer networks face the challenge of incentivising peers to maintain active instances, even if they aren't actively using the service. Many structured overlays require a minimum number of nodes in order to perform effectively; by ensuring instances remain online, keys and items do not need to be reallocated as frequently.

The Bitcoin protocol, despite its acknowledged flaws[23], possesses an effective incentivisation technique. A proof-of-work mechanism is used whereby all peers who provide the nodes required for transaction verification are rewarded with a fraction of Bitcoin currency.[24, 25] However, due to cryptocurrency speculation and the selection of a computationally-intensive 'mining' algorithm, this technique has been widely criticised for it's high electricity consumption.[26]

Authorisation

Peer-to-peer networks have the same authorisation challenges as standard networks, with the added complexity of the lack of a central authority and the fact that resources are provided by un-trusted peers and not a central trusted agent.

Broad authorisation challenges include:

- Verification that messages are from the relevant author
- Management of user credentials without a centralised authority
- Distribution of encrypted or private messages

Addressing these issues is complex and generally considered an open problem in the field, possible solutions have been implemented based on trust and reputation [27], such as the 'Web-of-Trust' in PGP[28, 29, 30], and the 'Eigentrust' algorithm.[31]

Connectivity

NATs and sub-nets, while essential for network management, can present challenges for peer-to-peer connections due to their potential to obscure or restrict direct communication between devices.[6, p.148-153]

There have been numerous techniques and protocols developed in order to allow fire-walled or NAT connected devices to establish peer-to-peer connections, with varying levels of reliability and safety. Descriptions of such techniques can be found in appendix A.2.

Distributed Systems and Consistency

Although there are shared concepts between peer-to-peer and distributed systems, such as distributed databases (e.g. Apache Cassandra[32]) and server clusters (e.g. Amazon DynamoDB[33]), their comparatively low instance count, limited fragmentation, and single central authority make them less relevant for investigation in this project, consequently these types of systems have not been analysed in depth.

Fault-tolerance and consistency are a key challenge in distributed systems. In an e-commerce or financial transaction based application, consensus on information is essential. Guaranteed immediate consistency is less essential for functionality in a social application, and a delayed eventual consistency across the network is an acceptable compromise. Cassandra relies on a Gossip-based protocol for communication and implements the Paxos algorithm[34] to provide fault tolerance.[35]

In distributed system theory, the CAP theorem[36] states that any distributed system can fulfil at most two of the following guarantees;

- Consistency - every node provides the most up-to-date value for any read.
- Availability - nodes can respond to all requests at any time.
- Partition Tolerance - the network can remain functional in the event of a partial failure.

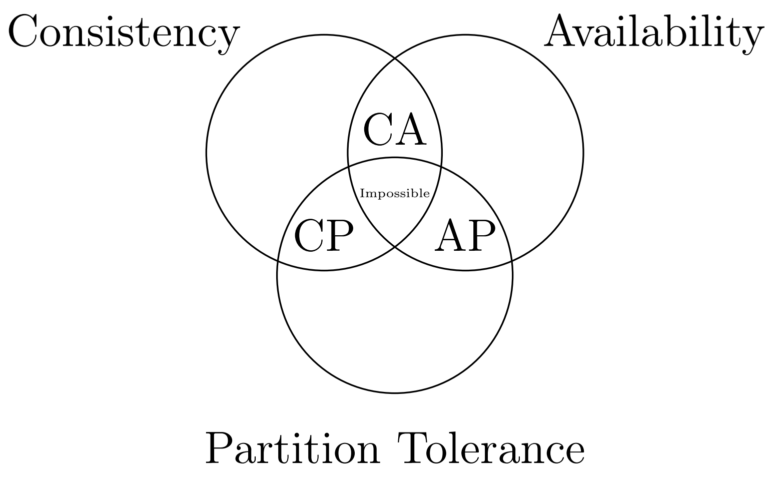


Figure 2.5: *CAP Theorem*

2.4 Existing Projects

The majority of existing decentralised social networks use a federated network topology, whereby multiple distinct networks are able to communicate and share resource via the use of a standardised protocol, effectively operating as a single uniform network. They share similar challenges to peer-to-peer networks, however resource location, authorisation and moderation are generally simpler due to the use of dedicated servers and administration.

The federated networks Usenet, Mastodon and Nostr have been examined due to their novel approaches at solving different challenges, Secure Scuttlebutt is a decentralised peer-to-peer social messaging network, which shares common objectives to this project.

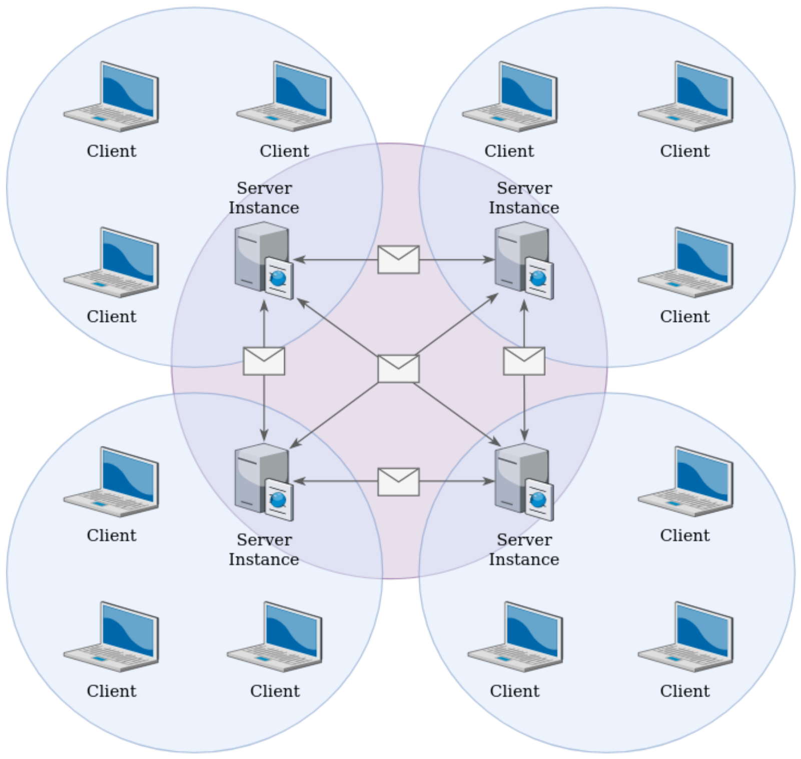


Figure 2.6: *Federated networks are composed of sub-networks of inter-connected, distributed server instances.*

Usenet

Usenet, originating in the 1970s, relies on a distributed network of servers for storing and exchanging messages[37]. Operating independently with their own access and moderation policies, each server hosts discussion forums known as newsgroups, and messages are exchanged between all interconnected servers.[38] An early form of content moderation was proposed in the form of "killfiles"[39, 40], whereby custom content filters are implemented by the application at a client level. The increased storage requirements, user subscription fees, the rise of spam, the misuse of the network for large file sharing[41], as well as the introduction of more sophisticated protocols, all led to the decline of Usenet. Nevertheless, some servers still remain active today.

Mastodon

Mastodon is a "Decentralised Online Social Network"[42] which implements the ActivityPub protocol[43]. Content hosting and user identities are managed by separately owned but interconnected server instances. The network is divided across multiple servers, and thus user access, accounts and content are controlled by multiple private hosts[5]. To prevent instances growing beyond a manageable size, administrators may lock new user registration[44].

Nostr

Nostr also uses a federated network topology to distribute messages, however user accounts are authenticated purely cryptographically and not by the relay servers[45]. With this approach, many issues associated with peer-to-peer networks are avoided[46], but the network is still dependent on private hosting and dedicated management for the relays. The developer has proposed charging a small fee for each post to provide an incentive for maintaining a server and to prevent spam or spoofing attacks.[47]

Secure Scuttlebutt

Secure Scuttlebutt (SSB) is a peer-to-peer social media protocol using a gossip algorithm[48] which is accessed through third-party client applications. The protocol has a strong focus on security and encryption and discussion 'topics'. Each user publishes a cryptographically signed feed which provides a history of their posts[49]. SSB relies on an unstructured overlay and the ‘small world phenomenon’[11]. This algorithm may be prone to a high messaging overhead and inefficient search[6, p.196], however as yet there is no available academic analysis on the scalability or reliability of SSB beyond the project’s white-paper[48].

The SSB project defines a protocol and library, a third-party developed client application must be selected and used by the end-user. SSB has solved many of the problems inherent with peer-to-peer networks applied to a social networking application and currently presents a successful existing example of a network of this type. While the strong focus on privacy and encrypted messaging is a key aim of the project, it also makes the network impenetrable to newcomers. Interactions are more suited to invite-only private messaging between groups of users, than the open sharing and distribution of public content. While well documented, the protocol and implementation are not designed for pedagogical purposes, with a high degree of complexity.

Chapter 3

Methodology

3.1 Research Methodology

The research methodology of this project will follow a constructive research approach, in which a real-world problem is identified and solved by implementing a new construction.[50] Specifically the aim being to understand and address the challenges of a peer-to-peer social application through the construction of a piece of software which functionally provides decentralised peer-to-peer sharing of user created content.

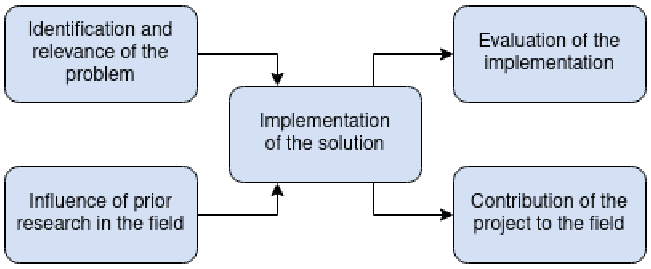

Identification and relevance of the problem
Influence of prior research in the field
Implementation of the solution
Evaluation of the implementation
Contribution of the project to the field

Figure 3.1: *Elements of the constructive research approach. Adapted from Lukka 2003.[50]*

In addition to the main aims and objective of the project, the literature review identified a need for a solution implementation which:

- **Is contemporary.** earlier peer-to-peer research projects are frequently outdated or irrelevant to the current landscape. For example references to long abandoned projects such as Napster or Kazaa, file-sharing platforms which are obsolete due to legal issues and the availability of streaming services. While contemporary peer-to-peer projects exist, they are heavily associated with file-sharing, total anonymity, "free speech" or cryptocurrency, all of which come with intricate ethical and legal ramifications.
- **Utilises modern technologies and high-level tech stacks.** Most online services are supported by cloud computing and the paradigm shift is increasingly away from user-hosted and controlled products. The chosen development ecosystem is not specifically designed for peer-to-peer applications, but this research should ascertain it's suitability.
- **Provides a broad functional overview.** Much of the existing research tends to

focus on an individual facet of a given topic, such as assessing the performance of a particular DHT algorithm. Consequently, these elements are often examined in isolation or purely theoretically. Therefore, one aim of this work is to produce a holistic 'reference' implementation, in order to explore how the separate solutions can be integrated within the context of a functional application.

- **Demonstrates a generic framework.** While existing peer-to-peer based projects tackle analogous challenges[1], the libraries and techniques they have developed are tailored to their specific use case. The intricacy of commercial project code-bases, coupled with their wide scope, can obscure an understanding of the underlying systems.

It is hoped that by implementing practical applicability, clear functionality and documentation, the outcome of this project should provide both a practical and theoretical framework of a modern peer-to-peer system. Thereby contributing towards further research and development in the field of peer-to-peer networking.

3.2 Evaluation Methodology

Comprehensive research was discovered during the literature review phase which provides an in-depth comparison and analysis of different existing algorithms. As this project will not concern novel algorithmic optimisations, further bench-marking may be redundant. The user interface and network elements should be expected to be responsive and resolve in a timely manner, but given the nature of the project, focus will be placed on assessing the functionality of the implementation, as opposed to performance metrics.

The evaluation will need to highlight additional challenges and successes encountered during development, and primarily concern the effectiveness of the chosen methods in addressing the identified peer-to-peer challenges. The following section will outline a specification for the features and requirements of the software project, in order to provide a framework for assessing the suitability of the proposed solution.

[1]Such as HyperDHT `https://docs.holepunch.to/building-blocks/hyperdht` and IPFS `https://ipfs.tech/`

Chapter 4

Proposed Specification

4.1 Feature Specification

To address the objectives of the project, an application will be developed which should:

- Allow creation, exchange and storage of user created documents (a JSON object) over a peer-to-peer network.
- Provide a basic interactive user interface.
- Host a server instance to forward or respond to external requests from other instances.
- Implement at least the following Remote Procedure Calls:

Command	**Parameters**	**Action**
Ping	<address>	Check a node is online, and update cache
Get Peer	<id>	Find the info and address of a peer in the network
Get Providers	<item key>	Find the addresses of the providers of a specific item
Get	<item key>	Retrieve a specific item from the network
Publish	<item object>	Upload a new item to the network

Additionally the application should:

- Provide the ability to 'follow' other peers and access a 'feed' of their latest activity.
- Provide each user with a unique identifier, a basic level of authentication and content verification.
- Use a dynamic peer-to-peer overlay network for peer discovery, content indexing and distribution, which is adaptive to moderate peer churn.
- Provide an automatic bootstrapping service to allow new participants to join the network.

Items are JSON objects which represent either a document, user or feed. A document contains any text based content which the user wants to share on the network. It has an owner, a timestamp and signature. The user object contains a user's public information such as their last known address and the key of their last content feed. A feed is a list of documents which belong to a user, multiple feeds are created by a user, but the most recent one is linked from their user object. In addition, a user profile item is stored locally, which contains the user's encrypted secret key and followed user list. See appendix B.1 for the full schema definitions.

The network components will adopt a hybrid approach; the leveraging of 'small world' social network characteristics is provided by using a cache of active recently connected and known peers, but the application will also utilise a distributed hash table for guaranteed item availability and load distribution. This should allow neighbour peers to efficiently exchange information with each other, but also allow for more efficient item look-up without flooding the whole network.

Bootstrapping will be performed by providing each instance a small set of active peers at static addresses through which to join the network. Upon program exit the cache stores every known active peer to disk, which if still online, can also be used to bootstrap on the next program start.

Each user will be assigned a 'unique' public key[1]. The corresponding private key could also be used to authenticate user identities and sign items.

4.2 Technology Choices

The solution will be a native (Linux/Windows/OS X) computer application. While typically, social network services are accessed through a web or smartphone application, a peer-to-peer application requires a higher level of complexity as it must manage peer-to-peer connections, store content locally and operate a server instance to handle outside user requests.

The JavaScript language and the Node framework will be used to allow for rapid prototyping and testing. Node is typically used for micro-services and distributed applications, it provides built-in asynchronous features, compatibility with internet protocols and is well-documented. A key strength of the Node framework is the Node Package Manager and the third-party package ecosystem, a full list of the third-party libraries used in the development process is included in the appendix B. JavaScript also allows the possibility to port the application to other platforms such as browsers or mobile, in the future.

Where possible, the 'airbnb' JavaScript style guide will be adhered to[2], this should ensure good code practice, minimise errors and aid maintainability. As the most used language globally, JavaScript is a solid choice for a reference project which will benefit the widest audience.[51]

All inter-node connections will be performed using the web-socket protocol, while requiring more overhead compared to using raw TCP/UDP sockets, it is hoped that this will help to reduce the complexity of the implementation, and increase readability and reliability.

4.3 Limitations

This project cannot hope to address or solve the whole spectrum of challenges presented by peer-to-peer networks, therefore a set of constraints has been adopted to limit the scope

[1]Using the Curve25519 elliptic curve, there are technically 2^{255} possible public keys

[2]`https://github.com/airbnb/javascript`

and focus the project aims.

While consideration of authentication and security concerns are essential for any network application, given the limited project time-frame, the implementation of end-to-end encryption, peer anonymity, or resistance to determined attackers cannot be guaranteed. Opening arbitrary ports and publicly divulging users' IP addresses is considered an insecure practice, but is integral to participation in a peer-to-peer network.

NAT and firewall traversal techniques are a major part of peer-to-peer networking, due to the complexity of the challenges however, connection guarantees fall outside the project requirements for the software. For testing and evaluation purposes it will be assumed that peers have open firewalls, enabled port forwarding or share a local network.

Censorship and content moderation are contentious issues when dealing with user provided content, with complicated social and legal ramifications for uncensored networks. These issues have no universally appeasing solution, and the scope of this project is limited to the computer science, as opposed to social, aspects of the problem.

Chapter 5

Project Design

5.1 System Architecture

The application functionality is divided up into distinct modules, as is idiomatic for Node applications. The modules are expected to encapsulate implementation detail as much as possible from each other, however a purely Object-Oriented paradigm has not been adopted for the project, to avoid adding additional constraints onto the development process.

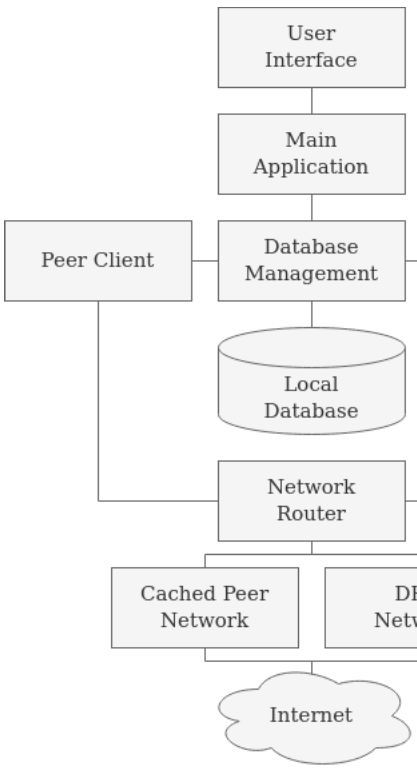


Figure 5.1: *The component architecture*

5.2 System Components Overview

Main

The main entry point for the application which performs parsing of command line parameters and initialisation of the Database, Interface, Client and Server modules.

User Interface

This component provides an interface for the user to interact with the program, search and view content, and manage networking. Initially, this will be an interactive terminal user interface (TUI) tool. The client interface is implemented in such a way to allow for the addition of different interface components at a later stage, such as a more typical graphical or web-based front-end.

Peer Client

Provides functions to handle requests from the user interface. Such as authenticating a new user session, accessing local and remote documents, creating and publishing new documents to the network. Interfaces with the network routing handler to obtain remote nodes (providers) and forward user requests.

Peer Server

As any peer-to-peer application performs both client and server roles, there is the requirement for a server component to handle incoming requests from remote peers. This is a self contained module which serves incoming requests in the background without any interaction required from the local user.

Database Storage

This component will manage the storage and caching of documents and feeds. The peer cache entries are also stored, including saved peers and recent connections.

As opposed to flat files, a database will be used to store content on the local user's filesystem. SQlite is a good choice as it lightweight, generic and well supported, and can support both relational and document schema. As an embedded database we do not need to host a networked database server instance, minimising complexity and latency.

Network Routing Handler

This is responsible for resolving the addresses of nodes which can provide the responses for the client queries. Transferring documents is performed via direct Peer Client/Peer Server socket connection. The network modules provide a distributed key-value store interface. Values can be retrieved either from cached peers or via the DHT. Both must be kept up-to-date (using a heartbeat) and should cope with timeouts and churn.

Each network module will be self-contained and should encapsulate behaviour from the handler as much as possible, making it possible to later experiment with different overlay implementations.

Cache Network Module

This will maintain a cache of last known addresses of saved friend peers and recently-communicated-with peers to allow for the querying of content providers over a restricted ‘small-world’ style network. Unlike a traditional gossip protocol, queries will not necessarily be repeated by the subsequent nodes, to avoid complex issues with network congestion and cycles.

DHT Network Module

A Distributed Hash Table is used to provide a ‘canonical’ index of each item and provider on the network, to be used if the cached peer network is insufficient. The Kademlia algorithm is well-used and provable[14]. This functionality will be provided by an external library, but would benefit from being replaced by a bespoke implementation at a later stage.

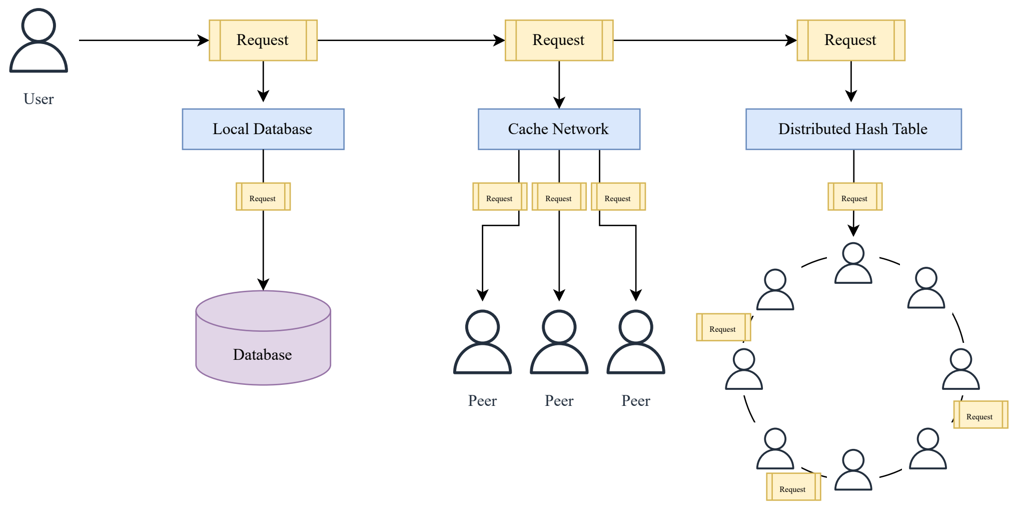


Figure 5.2: *A request for an item is sent to the local database, then the cache, then the distributed hash table. This way the ’quickest’ route is preferred.*

Chapter 6

Project Plan

6.1 Development Plan

While preparing the research proposal, an initial prototyping phase has been undertaken to provisionally scope out the requirements and limitations of the chosen technologies and tools. The main development phase will use an incremental methodology, which is well suited for complex systems. In this approach, the project functionality is divided into decoupled and modular components which provide core and extended functionality. An initial application is first assembled of the core components, assessed, and then additional features are added to subsequent 'releases' as the development progresses.

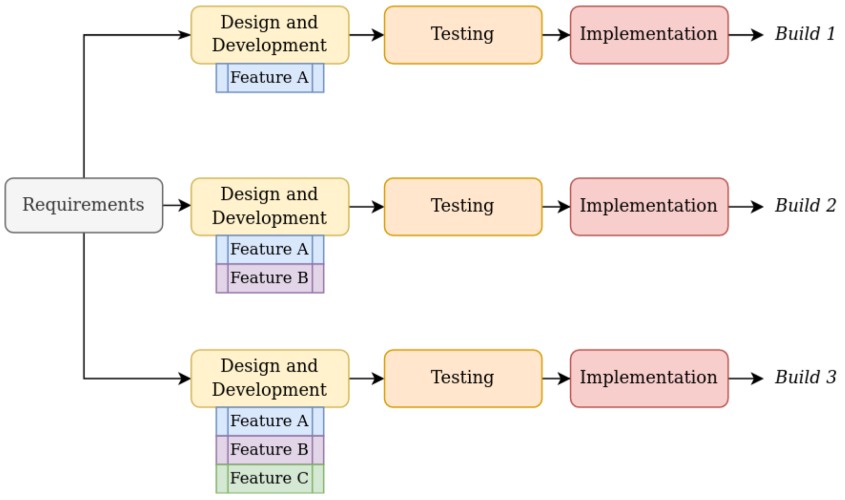
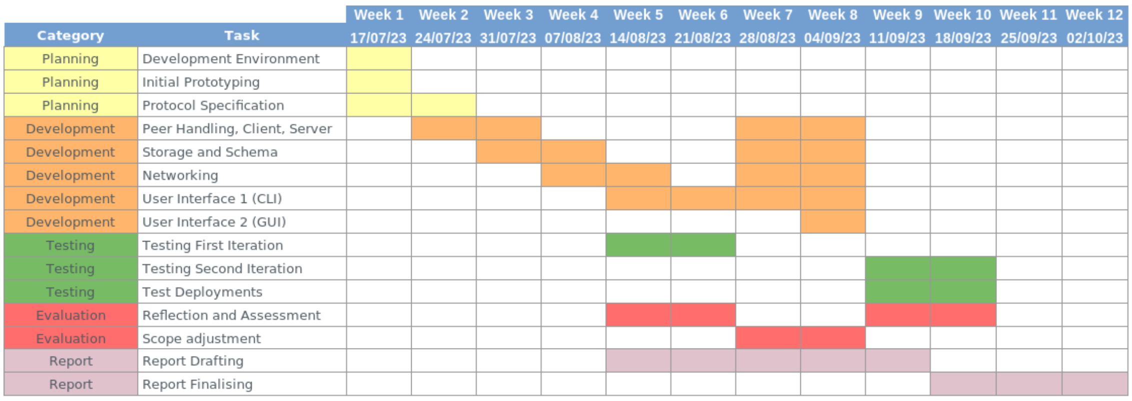


Figure 6.1: *The incremental model iterates to produce multiple builds with additional features.*

Given the broad scope of the chosen project, an incremental release should mitigate the risk of cascading missed timescale targets. Between each release the code-base can be refactored and, if necessary, the aims and scope can be re-evaluated. This should help ensure that a functional application is still produced in the event that extended features cannot be finished by the deadline.

6.2 Timeline



Category	Task	Week 1 17/07/23	Week 2 24/07/23	Week 3 31/07/23	Week 4 07/08/23	Week 5 14/08/23	Week 6 21/08/23	Week 7 28/08/23	Week 8 04/09/23	Week 9 11/09/23	Week 10 18/09/23	Week 11 25/09/23	Week 12 02/10/23
Planning	Development Environment	🟨											
Planning	Initial Prototyping	🟨											
Planning	Protocol Specification	🟨	🟨										
Development	Peer Handling, Client, Server		🟧	🟧				🟧	🟧				
Development	Storage and Schema			🟧	🟧			🟧	🟧				
Development	Networking				🟧	🟧		🟧	🟧				
Development	User Interface 1 (CLI)					🟧	🟧	🟧	🟧				
Development	User Interface 2 (GUI)								🟧				
Testing	Testing First Iteration					🟩	🟩						
Testing	Testing Second Iteration									🟩	🟩		
Testing	Test Deployments									🟩	🟩		
Evaluation	Reflection and Assessment					🟥	🟥			🟥	🟥		
Evaluation	Scope adjustment							🟥	🟥				
Report	Report Drafting					🟪	🟪	🟪	🟪	🟪			
Report	Report Finalising										🟪	🟪	🟪

Figure 6.2: *Provisional GANTT timeline for the project proposal, showing the planned stages of development.*

6.3 Software Testing

For each module there will be comprehensive unit tests written using the Mocha framework, to verify performance and assert correct behaviour. Skeleton/stub modules will need to be used to allow for integration testing at an earlier stage, for instance so the network and file transfer functions can be tested even if the data persistence section is incomplete.

The testing of distributed systems can be a complicated process[52], it is hoped that the choice of documented and well-used algorithms and open-source libraries for the lower level components will reduce potential problems.

Large scale trial network deployments with large quantities of nodes would provide a good indication of network performance, but a trade-off will have to made between allocating time towards building an elaborate automated testing environment or refining the functionality of the application.

6.4 Risks

The following project risks have been identified, along with potential mitigation strategies.

System Failures

Ensure hardware and software are in good working order. Use multiple automatic and remote backups. The project directories are automatically backed up daily. Progress to any software source code will be tracked by the git tool and pushed to a remote GitHub repository.

Time Management

Make sure to consult the timeline, regularly re-assess progress and if necessary re-adjust scope. Prioritise the most important features, and work incrementally on both project and report.

Unforeseen or Unsolvable Implementation Issues

Dedicate time beforehand to building a broad understanding of the problem area and what is achievable. The discovery of insurmountable challenges can be analysed and re-contextualised into a productive research outcome.

Chapter 7

Implementation

In its current state the program offers all of the core and extended functionality outlined in the specification. A full description of the program's functions can be found in the code documentation, however a brief overview will be provided.

7.1 Core Functionality

Command Line Parameters

The application accepts the following command line parameters:

```
Usage: psddn [options]

Options:
  -a, --address <address>      Specify the IP address of the server
  -p, --port <port>            Specify the port of the server
  -db, --dbname <dbname>       Specify the name of the database instance
  -b, --bootstrap <bootstrap>  Specify the bootstrap peers file
  -i, --interface <interface>  Choose the user interface to use (none, web,
                               terminal)
  -d, --debug                  Enable debug mode
  -u, --user <user>            Specify the user key to login with
  -s, --secret <secret>        Specify the user password to login with
  -h, --help                   display help
```

Terminal User Interface

The program is used through a basic terminal user interface which allows the user to perform actions and displays a response to the console.

The user interface accepts the following commands:

```
get, put, publish, newPost, followUser, unfollowUser, getFollowedFeeds,
getFollowedUsers, getFollowedDocuments, getUserDocuments, debug, help, exit
```

And the following debugging commands:

```
profile, cache, users, documents, hand, ping
```

Create a New Document

The *newPost* action creates a new document with a title and content, and publishes it by sending it to peers via the cache and DHT networks. Publishing a new document causes the current user's Feed to be appended with the new document key, and the 'lastFeed' parameter of the User item is replaced with the key of the new feed. The updated Feed and User items are also published onto the network, to allow other users to access the up-to-date information.

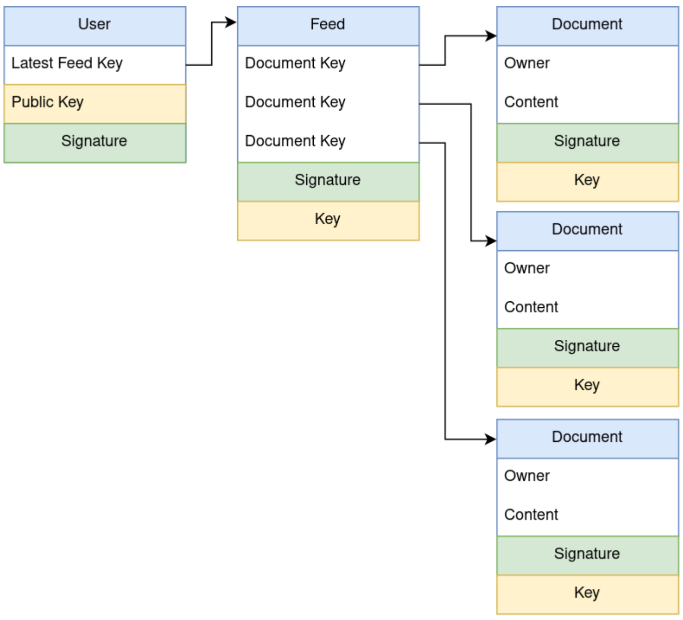


Figure 7.1: *A User item links to a Feed item, which links each of the user's created Documents.*

Getting Content

Using the *get* action requests a single item (User, Document or Feed) by it's key . It first checks local storage and then remote providers via the cache and DHT. It is unlikely the user would use this function directly and instead use getFollowedDocuments to get their followed feed content.

The *getFollowedDocuments* action allows the user to acquire an updated aggregated feed comprising all the documents from their followedPeers. It uses the functions *getFollowedUsers* and *getFollowedFeeds* to query the network for the most recent documents, and returns them within an array; *getFollowedUsers* returns the latest user objects for each user that the current is currently followed and *getFollowedFeeds* returns a list of all the latest lastFeed keys for every followed user. getUserDocuments allows the retrieval of all of a specifc user's documents.

Following Actions

The current user can manually add/remove followed peers on their user profile using *followUser/unfollowUser*. As the user profile is stored as JSON, an array is used with helper functions rather than a Set or Map.

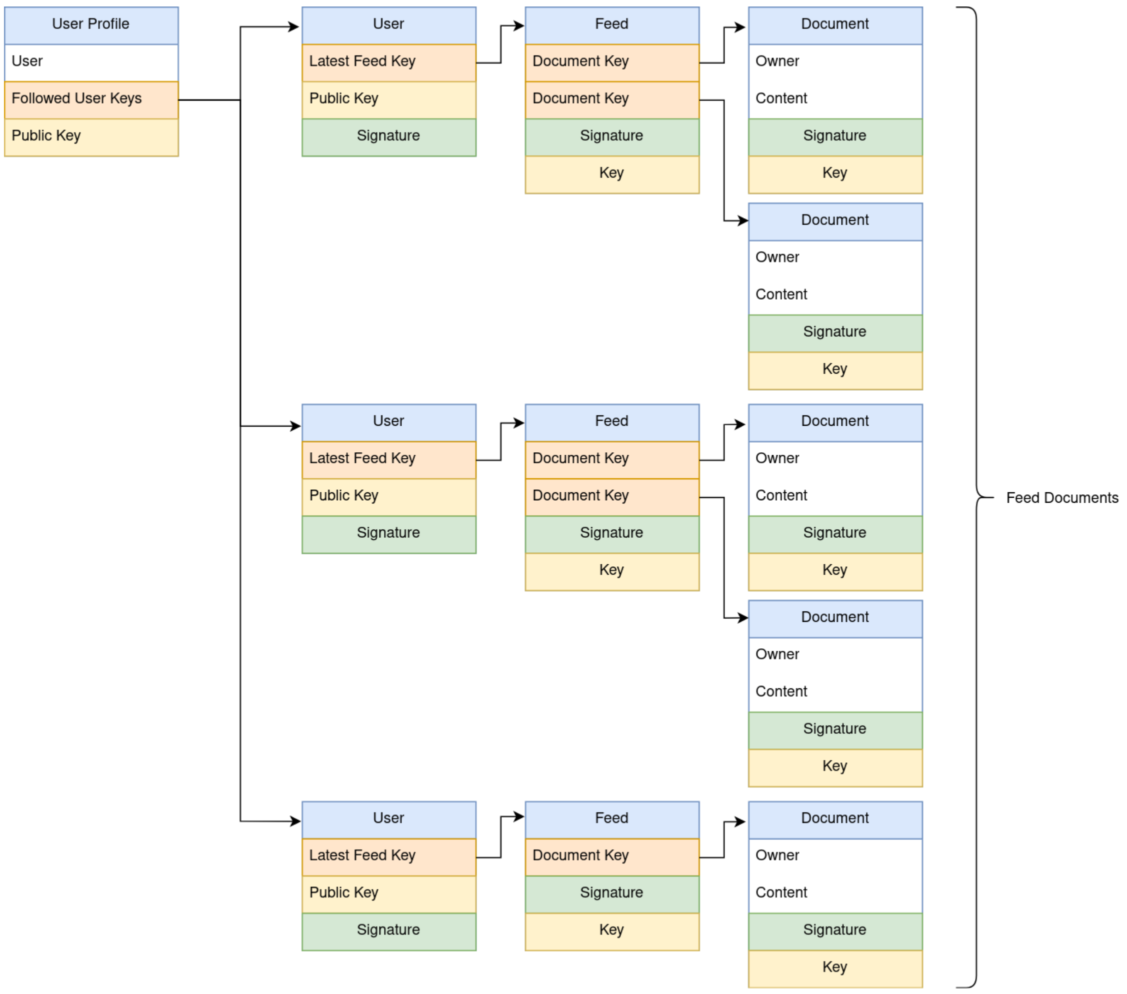


Figure 7.2: *The contents of Users, Feeds and Documents can be used to build an aggregated Document feed.*

Database

The database path is set to either the default location, or a user specified location. If a valid database is not found, then a new one is created. The database instance is initialised as a 'singleton' module and, to prevent multiple instantiating, provided via a dependency injection ('setDatabase') function to each module that requires access in order to prevent circular dependencies.

Provider Discovery and Bootstrapping

Upon program start the bootstrap file, if provided, is loaded and the local database is checked for any saved peer addresses. The bootstrap file is a JSON text file containing addresses and ports of 'static' nodes.

The cache module sends a handshaking query to each retrieved address, any unresponsive addresses are pruned, the target node reciprocates with a handshake and adds the new peer information to it's own cache. Upon program shutdown the cache stores all active peers to disk, to be used for bootstrapping on next program start.

Cache Network

To fufill client requests for item providers, the cache returns a list of all known connected peers. The cache network uses ping and handshaking functions in order to add new nodes and maintain the health of the network. The cache is automatically refreshed in a defined interval, this consists of pinging each stored node to check if it is still active and removing inactive peers.

DHT Network

The DHT acts as a auxiliary network to the cached peers. If an item cannot be found in the local database, or from the cached peers then a request is made to the DHT, which should be able to locate it from all connected peers. The DHT is useful in the event of a user wanting an item which is held by a peer to which they do not have a direct connection (isn't in the cache). Along with Documents and Feeds, the DHT can be used to obtain User items, from which a handshaking request can be made to add them to the cache. The original intent was for the DHT to act as both an item distribution network, as well as a look-up network, to provide the client module a list of peer address of who should have the queried item. Instead the implementation treats the DHT as a distributed key-value store, without revealing to the rest of the program the underlying routing and functionality of the network.

7.2 Extended Functionality

User Sessions and Authentication

In addition to the User item, the application also uses a UserProfile item which contains the User item, along with the encrypted secret key object and an initially empty 'followed users' array. In the absence of a centralised authorisation service, this User Profile represents the full private account information for a registered user, and must be transferred across systems if the user wishes to move their account. If the user loses their password or plain-text secret key, then that account cannot be restored and a new account must be created. As the secret key is encrypted, in theory it could be distributed on the network, to allow logging in on any machine, but this is not implemented.

On program start the user is prompted to login, or create a new user profile:

Creating a new user profile generates a new User item with a random public and secret key, a provided nickname and password. The nickname merely serves as a display name for user convenience and does not infer authorisation or uniqueness. The password is used to encrypt the generated secret key for local storage. A User Profile item is also generated. The current User Session is set to the new Profile and the User Profile is saved to the local database.

When the login option is selected, the user provides their Public Key and their given password. The application loads the User Profile from local storage and sets the User Session.

The User Session module to allows the application to store the User Session of the current authenticated User. It provides access functions in order to broadcast the user identity to other peers, to use/modify the followed peers and feeds and sign items.

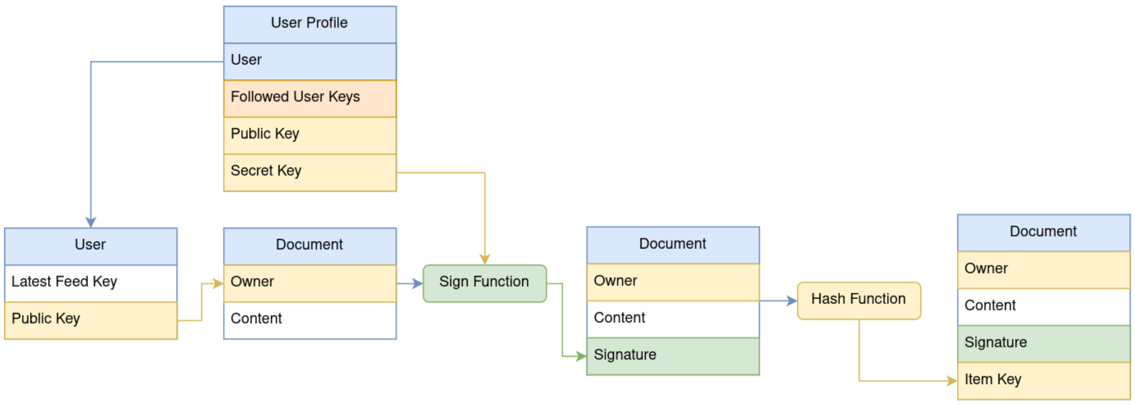


Figure 7.3: *After creation, items are signed with the user's private key, and the hash function is used to generate a 'unique' consistent key.*

Validation and Signatures

For feeds and documents, the item key is a hash of the entire item itself (minus the key field), this hash is checked upon item storage and retrieval. If an item has been modified then the hash will not match and will cause an error. As the feed, lastSeen and lastAddress parameters can change, the User item cannot be verified in the same way, and the public key is used.

The hash verification only checks that an item has not been modified after creation, it does not prevent a malicious user from creating and hashing a new document with another user's key. To solve this problem, a signature/verification system is implemented using the "tweetnacl" library and ed25519 encryption. New Items are signed upon creation or modification with the current users' private key, and remote items are authorised upon retrieval over the network with the creators public key. The User item can be signed on each feed update (ignoring the address and last seen fields).

Debugging Functions

Basic functions were added to allow run-time debugging of the application. Ping and Hand allow manual execution of the Ping and Handshake RPCs defined below, in order to directly query and add peers to the cache. Additional functions allow the display of the current peers in the cache network and the current user session information, as well as displaying the contents of the local database.

7.3 Protocol

At program start, the server module is initialised at the provided port to answer incoming requests from outside users' client modules. The Requests and Responses are defined in the following RPC Protocol.

RPC Requests

- **Get**<Item Key, Item Type>

  Queries the local database for the Item Type with the provided Item Key. If the item is found then it returns a Success response with the item. If the item is not found it returns a Success response (as the query executed successfully) with null content. If an error occurred (e.g. an invalid key) then it returns an Error response.

- **Put** <Item>

  Stores a provided item in the local Database. The Key and Type are extracted from the object's parameters and verified, along with the object schema verification for invalid / missing fields. Returns with a Success or Error response.

- **Ping** <Target Peer Key>

  Used for maintaining the Cache Network or as a debugging tool. The Target Peer Key parameter is used by the calling instance to check if the address is actually the expected Peer. Returns a Success if the target peer key matches, or if the provided target peer key is null. Returns an Error if the Target Peer Key is not the current user at this address.

- **Message** <Message Text>

  Allows for the transmission of temporary text messages to be printed to the console, for debugging or basic instant messaging.

- **Handshake** <Origin Peer Key, Origin Peer Port, Target Address>

  This user information sharing Request is reciprocated in order to add a new peer to the cache, mutually determine which unique peer is at which address, and allow a new peer to learn its own 'external' IP address.

RPC Responses

- **Success** <Message | Item>

  Used to indicate a procedure resolved successfully, contains an item if one was requested or a message.

- **Error** <Message>

  Used to indicate a problem occurred, with a message describing the error.

Chapter 8

Evaluation

8.1 Feature Implementation

As outlined in the methodology, the aims of the project require a qualitative rather than quantitative evaluation, this section will analyse how well the features of the initial specification were implemented, considering their performance and identifying any additional challenges encountered during development. This should help to build a comprehensive assessment of the project's strengths, and areas which require further attention.

Locating resources without a central index

The user is able to request items from any connected peers using the cache module. The feed system allows users to find any new posts which have been made by their followed peers. The DHT provides a method to distribute items in a more 'global' manner but is not as integrated as originally intended. Testing of this was performed manually by setting up a test environment of multiple nodes and verifying that documents could be published and accessed as expected. As the DHT requires more than a handful of peers to function properly, a test network script was employed to create large numbers of DHT nodes.

Managing peer churn and connectivity

The refresh scheduler on the cache prunes inactive nodes. A long refresh interval was used during testing to prevent network chatter, but the value would need to be optimised in a full deployment. Given the absence of a simulated testing environment, this feature was also tested manually using a dummy network of multiple local nodes, as well as integration tests. It should be noted that in the absence of NAT traversal techniques, instances required additional port forwarding to allow incoming connections from an external network.

Bootstrapping Peers into the network (without a direct invitation)

New node instances can join the network in three ways; via an optional 'bootstrap' file which contains address and ports of working bootstrap nodes, the cache module checks any saved peer address on startup, and the DHT has it's own bootstrapping parameter which is passed in at initialisation.

Testing was undertaken using a shell script to spawn multiple application instances using a provided bootstrapping file, containing addresses of active nodes on the local machine, on different computers connected through a LAN, as well as remote instances running on Google Cloud Services.

Locating resources in the event of offline users

The feed system allows a user to check if their followed peers have made any updates since they were last online. If a user in the cached peers has a copy of a searched resource then it can be found, even if they are not the original owner. However if there are simply too few users online then the service is inconsistent. The Kademlia protocol should ensure that any documents created are distributed across the network, reassigning them on dropout. The implementation of database persistence ensures that even if nodes go offline and reconnect, the integrity of any stored data should hold, and any stored items will be available for request.

Managing identity and verification without a central authority

This was satisfied through the implementation of a public key authentication system, described in the implementation section. This makes it impossible to impersonate a user without the corresponding secret key. The (ed25519) signing and verification functions ensure that a fake item cannot be distributed without being signed by the user, and the use of a hash function as a unique identifier for feed and document items prevents modification. To prevent storage of plain-text secrets, the user's private key is encrypted with a user provided password (PBKDF2) and only stored locally.

The cryptography libraries and algorithms used have been publicly audited[1], and unit tests were used to verify the implementation; checking for errors such as incorrect passwords, modification of signed documents and validation.

8.2 Additional Challenges

It should be noted that the code-base, as outlined in the Implementation section, fulfils the core features specified and demonstrates the transfer of user-generated content over a custom peer-to-peer network. Certain obstacles were anticipated in the initial proposal, and while some complexities proved to be underestimated, others were less troublesome than expected. In addition, several unexpected issues emerged during development, the following section provides a brief summary.

Modularity and Encapsulation

The lack of a rigorous pre-defined protocol for inter-module function calls made it difficult fully encapsulate each module's functionality. This could have been ameliorated by defining a comprehensive protocol before the development phase started. An application of this nature is a large undertaking for a solo developer, and the project would have benefited from being undertaken as a group effort, allowing the dividing of different feature implementations across teams.

[1]https://cure53.de/tweetnacl.pdf

Testing

It was noted in previous sections that the testing of any distributed system is complicated, and this is especially true in a peer-to-peer distributed system. For further development of a commercial-grade project of this nature it would be essential to employ a more scientific and rigorous testing framework, ideally fully specified at the outset of the project, before undertaking the main development phase.

Existing research and tooling in this area focuses on traditional client-server systems, or distributed systems of the kind identified in the literature review[53]. While tools such as PeerSim (https://peersim.sourceforge.net/) exist for the simulation of different peer-to-peer protocols, a bespoke tool would likely need to developed specifically for this project, to deploy, remotely control, and log information from large quantities of virtual nodes. This was unfortunately unattainable within the time-frame of this project.

Timescale

Having the opportunity to compare and demonstrate different resource location and overlay algorithms would have enriched the project, however this was not possible due to the finite development time needing to be invested in the core functionality. The time to debug errors in distributed systems can be higher, due to their bidirectional nature. Moving forward, time requirements of the core specification features should be allocated appropriately, with the scope limited, if needs be, to a specific feature.

Complexity

The complexity of aggregating results from a distributed source is easy to underestimate and developing a production-level distributed database was not the original aim of the project. A future application of this sort might benefit from reliance on an external distributed database implementation for file storage rather than developing a bespoke solution.

During the development phase the design of the Get and Put functions was compartmentalised from how the user would interact with the service. The implementation of a standardised API to fit between the user interfaces and the client functionality would have aided with this, but also have increased the complexity of the application.

Interface Requirements

While the current interface adequately servers its intended purpose, designed primarily for testing and debugging during development, it falls short of meeting the expectations of regular end users who would want to access such a service. The prioritisation of back-end functionality as per the original specification left limited time for the development of the interface.

Integration

During development the question arose whether the DHT should provide an index of item providers or the item itself. Integrating the DHT functionality into the existing application proved difficult, due to the level of abstraction used provided by the external library. A custom DHT implementation would allow for both the indexing of item providers and re-distribution of items from disconnecting peers as intended.

Chapter 9

Future Work

9.1 Improvements

This section provides some potential improvements to the existing functionality of the application, as well as additional new features which would benefit the application, along with further research in the field.

Interface

The program would benefit from an improved user ID / key handling mechanism, as the current interface requires manual copying and pasting of key hashes in order to perform actions. A more user-friendly approach within the terminal could be a pop-up menu of (cached and followed) user keys mapped to nicknames. With the development of a browser interface this would be also be easier to implement.

Security

Publicly announcing users' addresses and ports, and accepting connections from outside machines potentially exposes the user's computer to malicious agents. The handling of remote procedure calls and local storage of content from untrusted sources also present potential attack vectors. Importantly, the program would require a comprehensive security audit to prevent exposing users to attacks, this is fundamentally an additional challenge of peer-to-peer networks which was not discussed in the literature review of this project and should be addressed in any further work.

Social Aspects

The outcomes of the software were focused more on the technical peer-to-peer challenges as opposed to analysing factors such as how users interact with a social network and the user requirements from such a system. The social networking aspects were not explored; such as how do users find each other on the service and how do users find new content on the service.

Incentive

The challenge of incentive was largely unaddressed in the software project. However, due to the deliberate choice of text-only content, the resource demand on users is relatively low. Items can be distributed rapidly, using a low amount of bandwidth and occupying little local storage space, therefore the system does not require a large pool of resources in order to operate. Incentivisation requirements are more prominent in systems distributing large files or performing resource-intensive computation.

9.2 Features

In addition to the points raised in the previous section, there are other potential features which could be explored for the future of this project:

- Fully implementing the web-based front end allowing the user to access the service locally through a browser window, this would provide a cleaner interface and more usable interface, closer to a traditional social networking service and which could be integrated using a desktop application framework.[1].
- Both the Cache and DHT functionality would benefit from expansion. The Cache could support peer announce functions in order for peers to share more information with each other, and request forwarding should be implemented to allow for epidemic-style searching between further nodes.
- The system could be expanded to allow the sharing of binary data, such as files or audio/visual content, although this would introduce additional resource and moderation requirements.
- The application could be 'packaged' or re-written in order to run directly in a browser context, this would lower the barrier for end-users to access the service as they would not need to install software locally.[2]
- Automatic NAT traversal and "holepunching" techniques such as those detailed in the literature review.

[1]e.g. Electron: https://www.electronjs.org/

[2]e.g. Using the browserify tool: https://browserify.org/

Chapter 10

Conclusions

This project explored a technological alternative to centrally-controlled social networks by developing a peer-to-peer document distribution application. While fully distributed peer-to-peer technologies offer several advantages, there are also significant challenges hindering their widespread adoption.

The core motivations were building a comprehensive understanding of the field, evaluation of practical solutions to selected issues and identifying any additional challenges inherent in implementing such a system.

The resulting software demonstrates how a functional application can be assembled to fulfil all the features of the specification. It was found that such a system is feasible but the challenges of large peer-to-peer networks should be adequately planned for, as failing to address any issues can undermine the usability of the application. Given the idiosyncratic nature of the field, the development of a well-defined custom overlay network is recommended, and as correct behaviour of a high-instance distributed system is difficult to formally test, any project wishing to further explore this field would benefit from the inclusion of a bespoke simulation environment.

Bibliography

[1] E. Meshkova, J. Riihijärvi, M. Petrova, and P. Mähönen, "A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks," *Computer Networks*, vol. 52, no. 11, pp. 2097–2128, 8 2008.

[2] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Looking up data in p2p systems," *Commun. ACM*, p. 43–48, 2 2003. [Online]. Available: https://doi.org/10.1145/606272.606299

[3] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Communications Surveys & Tutorials*, vol. 7, no. 2, pp. 72–93, 2005.

[4] H. Park, J. Yang, J. Park, S. Kang, and J. Choi, *A Survey on Peer-to-Peer Overlay Network Schemes.* IEEE, 2 2008, pp. 986–988.

[5] A. S. Tigelaar, D. Hiemstra, and D. Trieschnigg, "Peer-to-peer information retrieval: An overview," *ACM Trans. Inf. Syst.*, p. 1–34, 5 2012. [Online]. Available: https://doi.org/10.1145/2180868.2180871

[6] J. Buford, H. Yu, and E. Lua, *P2P Networking and Applications 1st Edition.* Boston: Morgan Kaufmann, 2008.

[7] A. Allavena, A. Demers, and J. E. Hopcroft, "Correctness of a gossip based membership protocol," in *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, ser. PODC '05. New York, NY, USA: Association for Computing Machinery, Jul. 2005, pp. 292–301. [Online]. Available: https://dl.acm.org/doi/10.1145/1073814.1073871

[8] Clip2, "The gnutella protocol specification v0.4," Online, 2023. [Online]. Available: https://courses.cs.washington.edu/courses/cse522/05au/gnutella_protocol_0.4.pdf

[9] Z. Haas, J. Y. Halpern, and E. L. Li, "Gossip based ad-hoc routing," *IEEE INFOCOM, June 2002*, Sep. 2002.

[10] X. Li and J. Wu, "Searching techniques in peer-to-peer networks," *Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks*, 08 2005.

[11] D. Watts and S. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, jun 1998. [Online]. Available: https://www.nature.com/articles/30918

[12] Stoica, R. Morris, D. Liben-Nowell, Karger, . David, Kaashoek, . Frans, Dabek, and H. Frank & Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17–32, feb 2003.

[13] F. Dabek, E. Brunskill, M. Kaashoek, D. Karger, R. Morris, I. Stoica, and H. Balakrishnan, "Building peer-to-peer systems with chord, a distributed lookup service," in *Proceedings Eighth Workshop on Hot Topics in Operating Systems.* IEEE Comput. Soc, 2001.

[14] P. Maymounkov and D. Mazières, *Kademlia: A Peer-to-Peer Information System Based on the XOR Metric.* Berlin, Heidelberg; Berlin Heidelberg: Springer Berlin Heidelberg, 2002, pp. 53–65.

[15] AMule.org, "Faq ed2k-kademlia," Online, 2009. [Online]. Available: http://http://wiki.amule.org/wiki/FAQ_eD2k-Kademlia

[16] C. Baquero, "What ever happened to peer-to-peer systems?" *Commun. ACM*, p. 14–15, 2 2023. [Online]. Available: https://doi.org/10.1145/3579633

[17] K. Leffew, "A Brief Overview of Kademlia, and its use in various decentralized platforms," Jul. 2022. [Online]. Available: https://medium.com/coinmonks/a-brief-overview-of-kademlia-and-its-use-in-various-decentralized-platforms-da08a7f72b8f

[18] A. Loewenstern and A. Norberg, "Bep-0005," Online, 2008. [Online]. Available: http://bittorrent.org/beps/bep_0005.html

[19] B. Cohen, "Bep-0003," Online, 2008. [Online]. Available: https://www.bittorrent.org/beps/bep_0003.html

[20] S. Cheshire and M. Krochmal, "Multicast DNS," IETF, Tech. Rep. RFC 6762, Feb. 2013. [Online]. Available: https://datatracker.ietf.org/doc/rfc6762

[21] J. F. Kurose and K. W. Ross., *Computer networking : a top-down approach*, 6th ed. Pearson, 2013, ch. 2.6, pp. 144–182.

[22] D. Stutzbach and R. Rejaie, "Understanding churn in peer-to-peer networks," in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement.* New York, NY, USA: ACM, 10 2006, pp. 189–202.

[23] N. Weaver, "The death of cryptocurrency," Online, Dec. 2022. [Online]. Available: https://law.yale.edu/sites/default/files/area/center/isp/documents/weaver_death_of_cryptocurrency_final.pdf

[24] M. Jakobsson and A. Juels, "Proofs of Work and Bread Pudding Protocols(Extended Abstract)," in *Secure Information Networks: Communications and Multimedia Security IFIP TC6/TC11 Joint Working Conference on Communications and Multimedia Security (CMS'99) September 20–21, 1999, Leuven, Belgium*, ser. IFIP — The International Federation for Information Processing, B. Preneel, Ed. Boston, MA: Springer US, 1999, pp. 258–272. [Online]. Available: https://doi.org/10.1007/978-0-387-35568-9_18

[25] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Online, 1 2008. [Online]. Available: https://web.archive.org/web/20230627173008/https://bitcoin.org/bitcoin.pdf

[26] R. Dillak, D. Suchendra, R. Hendriyanto, and A. A. G. Agung, “Proof of work: Energy inefficiency and profitability,” *Journal of Theoretical and Applied Information Technology*, vol. 97, Mar. 2019.

[27] L. Xiong and L. Liu, “A reputation-based trust model for peer-to-peer ecommerce communities [extended abstract],” in *Proceedings of the 4th ACM conference on Electronic commerce.* ACM, 10 2003.

[28] A. Datta, M. Hauswirth, and K. Aberer, *Beyond "Web of trust": enabling P2P e-commerce.* Newport Beach, CA, USA: IEEE Comput. Soc, 2003, pp. 303–312.

[29] P. Feisthammel, “Explanation of the web of trust of pgp,” Online, Oct 2004. [Online]. Available: https://www.rubin.ch/pgp/weboftrust.en.html

[30] G. Guo, J. Zhang, and J. Vassileva, *Improving PGP Web of Trust through the Expansion of Trusted Neighborhood.* Lyon, France: IEEE, 8 2011, pp. 489–494.

[31] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, “The eigentrust algorithm for reputation management in p2p networks,” in *Proceedings of the 12th International Conference on World Wide Web*, ser. WWW ’03. New York, NY, USA: Association for Computing Machinery, 2003, p. 640–651. [Online]. Available: https://doi.org/10.1145/775152.775242

[32] A. Lakshman and P. Malik, “Cassandra: a decentralized structured storage system,” *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, Apr. 2010. [Online]. Available: https://doi.org/10.1145/1773912.1773922

[33] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, “Dynamo: amazon’s highly available key-value store,” in *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles.* ACM, oct 2007, pp. 205–220.

[34] L. Lamport, “The part-time parliament,” *ACM Transactions on Computer Systems*, vol. 16, no. 2, pp. 133–169, May 1998. [Online]. Available: https://dl.acm.org/doi/10.1145/279227.279229

[35] A. Cassandra, “Guarantees | Apache Cassandra Documentation,” 2023. [Online]. Available: https://cassandra.apache.org/doc/4.1/cassandra/architecture/guarantees.html#lightweight-transactions-with-linearizable-consistency

[36] S. Gilbert and N. Lynch, “Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services,” *SIGACT News*, vol. 33, no. 2, p. 51–59, jun 2002. [Online]. Available: https://doi.org/10.1145/564585.564601

[37] O. Kirch and T. Dawson, *Linux Network Administrators Guide*, 2nd ed. O’Reilly, 2000, p. 506. [Online]. Available: https://www.oreilly.com/openbook/linag2/book/ch20.html

[38] M. Horton, “Standard for interchange of USENET messages,” RFC-Editor, Tech. Rep. RFC 1036, Dec. 1987. [Online]. Available: https://datatracker.ietf.org/doc/rfc1036

[39] J. Heitkötter, “Killfiles – the cure for all that ails you,” Online, Sep. 1994. [Online]. Available: https://www.whitman.edu/mathematics/eegtti/eeg_79.html

[40] L. Phillips, “Rn kill file faq,” 1994. [Online]. Available: http://www.faqs.org/faqs/killfile-faq/

[41] R. Slawsky, “Usenet file sharing thrives despite aol dropping access to it,” *New Orleans CityBusiness*, p. 1, 3 2005. [Online]. Available: https://link.gale.com/apps/doc/A130245132/ITOF?u=ull_ttda&sid=summon&xid=0c5fd9e4

[42] L. La Cava, S. Greco, and A. Tagarelli, “Understanding the growth of the fediverse through the lens of mastodon,” *Applied Network Science*, vol. 6, no. 1, pp. 1–35, 9 2021.

[43] C. Lemmer-Webber, J. Tallon, E. Shepherd, A. Guy, and E. Prodromou, “Activitypub,” Online, Jan. 2018. [Online]. Available: https://www.w3.org/TR/2018/REC-activitypub-20180123/

[44] D. Zulli, M. Liu, and R. Gehl, “Rethinking the "social" in "social media": Insights into topology, abstraction, and scale on the mastodon social network,” *New Media & Society*, vol. 22, jul 2020. [Online]. Available: libgen.li/file.php?md5=800317f88ab8970a169ec05f240c6cfb

[45] R. Huang, “Nostr Is The Decentralized Protocol That Might Replace Elon Musk’s Twitter,” 2023. [Online]. Available: https://www.forbes.com/sites/rogerhuang/2022/12/29/nostr-is-the-decentralized-protocol-that-might-replace-elon-musks-twitter/

[46] fiatjaf, “nostr - Notes and Other Stuff Transmitted by Relays,” Jun. 2023, original-date: 2020-11-09T03:17:41Z. [Online]. Available: https://github.com/nostr-protocol/nostr

[47] ——, “relayer/examples/expensive at master · fiatjaf/relayer,” may 2023. [Online]. Available: https://github.com/fiatjaf/relayer

[48] D. Tarr, E. Lavoie, A. Meyer, and C. Tschudin, “Secure scuttlebutt: An identity-centric protocol for subjective and decentralized applications,” in *Proceedings of the 6th ACM Conference on Information-Centric Networking (ICN ’19).* New York, NY, USA: Association for Computing Machinery, 2019, pp. 1–11.

[49] SSBC, “Scuttlebutt protocol guide,” Online, p. 28, 6 2023. [Online]. Available: https://ssbc.github.io/scuttlebutt-protocol-guide/

[50] K. Lukka, “The Constructive Research Approach,” in *Case Study Research in Logistics*, Jan. 2003, pp. 83–101.

[51] StackOverflow, “Stack Overflow Developer Survey 2023,” 2023. [Online]. Available: https://survey.stackoverflow.co/2023/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2023

[52] P. Maddox, “Testing a distributed system: Testing a distributed system can be trying even under the best of circumstances.” p. 10–15, 7 2015. [Online]. Available: https://doi.org/10.1145/2800695.2800697

[53] B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag, “Dapper, a large-scale distributed systems tracing infrastructure,” 2010. [Online]. Available: https://research.google.com/archive/papers/dapper-2010-1.pdf

[54] T. Reddy.K, A. Johnston, P. Matthews, and J. Rosenberg, “Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN),” IETF, Tech. Rep. RFC 8656, Feb. 2020. [Online]. Available: https://datatracker.ietf.org/doc/rfc8656

[55] B. Ford, P. Srisuresh, and D. Kegel, “Peer-to-peer communication across network address translators,” in *Proceedings of the annual conference on USENIX Annual Technical Conference*, ser. ATEC ’05. USA: USENIX Association, Apr. 2005, p. 13.

[56] M. Seemann, M. Inden, and D. Vyzovitis, “Decentralized hole punching,” Online, 2022. [Online]. Available: https://research.protocol.ai/publications/decentralized-hole-punching/seemann2022.pdf

[57] libp2p, “Hole Punching,” 2023. [Online]. Available: https://docs.libp2p.io/concepts/nat/hole-punching/

Appendices

Appendix A

Further Background

A.1 Advantages and Challenges of Overlay Networks

See table A.1 and A.2.

A.2 NAT Traversal Techniques

uPnP (Universal Plug and Play) allows devices within a local network to discover and establish connections automatically. It facilitates automatic port configuration and allows routers to forward incoming traffic to the correct local device, enabling applications and devices to connect to remote peers. uPnP can present a security risk if insecure devices (such as IoT devices) are exposed and allowed to connect to remote attackers.

STUN and TURN, part of the Interactive Connectivity Establishment protocol, were introduced to help address NAT related networking challenges.[54] STUN (Session Traversal Utilities for NAT) helps devices to determine information about the network layout and NATs present, such as discovering public IP addresses and port numbers, which can aid in establishing connections.[55] TURN (Traversal Using Relays around NAT) is a protocol for use when direct connections are blocked by network configurations, it allows the use of a intermediate host to act as a relay, facilitating connection between two peers. Other 'hole-punching' techniques exist either based on or serving similar functionality to STUN, TURN and ICE, which generally leverage 'open' nodes (not located behind a firewall or NAT) to act as relays to establish connectivity for unreachable peers. Examples of these techniques are outlined in detail in [55, 56, 57].

As the primary role of NAT is to conserve the limited IPv4 address space by allowing multiple devices to share a single address, it should be noted that the introduction of IPv6 obviates the need for NAT by expanding the usable address space. The challenges and complexities associated with NAT traversal are minimised as each device on a network can have a distinct, globally-accessible, IPv6 address.[6, p.149]

Table A.1: *Advantages and Challenges of Unstructured Networks*

Advantages	Challenges
• Can be simpler to design and implement in the short term • Small world networks and organic networks exhibit strong inter-node connectivity, without complex topology management or re-assigning of resources. • Small world network topologies closely match social networks in that most social connections are based around small groups yet are still interconnected by peers which are in multiple groups. • Can be both effective and efficient at small scale • The lack of rigorous structure allows networks to break up into isolated separate networks if required without complex restructuring	• As there may not be guarantees on the structuring of nodes, resource discovery is generally less efficient • Message propagation in unstructured overlays can tend towards exponential complexity as the nodes increase • Performance can be difficult to estimate asymptotically, no way to prove how long a search will take • Inefficient at large scale without controls to manage peer connections etc. • Certain parameters must be optimized, e.g., message TTL and rebroadcasting, to prevent excessive redundancy and message looping.

Table A.2: *Advantages and Challenges of Structured Networks*

Advantages	Challenges
• Quicker performance for resource search. As the structure determines item location. • Can offer lower messaging overhead, compared to Breadth First or Depth First searching. • Likewise, generally fewer peer hops are required to fulfil requests. • Performance and scalability are deterministic and can be mathematically proved. • More resilient to routing/spoofing attacks, if the network relies on random distribution and hashing. • The amount of required peer to peer connections can be minimised, and resources can be evenly distributed across the network.	• Can be more complicated to design and implement • Usually require a minimum number of nodes to perform effectively • Keys and resources must be redistributed. • Overlay must be restructured if nodes join or leave. • Hash-based structured networks are not suitable for content or keyword based search. • Participation in the network requires a higher potential commitment of client resources • Similar or related resources are distributed randomly across the network, rather than stored together

Appendix B

Implementation Details

B.1 Item Schema Tables

Table B.1: *Document*

Document		
type	Types.Document	The item type
key	String sha-256 hash in base64 encoding	The unique id of the item
owner	String sha-256 hash in base64 encoding	The item creator's id
timestamp	Date object as ISO string	Time created
title	String	User provided title of the document.
content	String	User Provided content of the document.
tags	Array of String	User provided descriptive tags
signature	String 64-byte long base64 signature	The user generated signature for this item.

Table B.2: *User*

User		
type	Types.User	The item type
key	32-byte key in base64 encoding	The unique id of the user (ed25519 public key)
nickname	String	Optional user provided nickname
lastAddress	Object { address, port }	The last known address of user (if any)
lastSeen	Date object as ISO String	The time user was last seen at lastAddress
lastFeed	String sha-256 hash in base64 encoding	The latest feed item for this user
signature	String 64-byte long base64 signature	The user generated signature for this item.

Table B.3: *Feed*

Feed		
type	Types.Feed	The item type
key	String sha-256 hash in base64 encoding	The unique id of the item
owner	String sha-256 hash in base64 encoding	The item creator's id
timestamp	Date object as ISO string	Time created
documents	List of sha-256 hash keys	The collection of document keys for the user
signature	String 64-byte long base64 signature	The user generated signature for this item.

Table B.4: *UserProfile*

UserProfile		
type	Types.UserProfile	The item type
key	32-byte key in base64 encoding	The unique id of the user (ed25519 public key)
secretKey	Encrypted secret key object	The encrypted ed25519 secret key of the user
userObject	Nested User Object	The user object
following	List of 32-byte keys in base64	Collection of the followed user keys

B.2 External Libraries

Table B.5: *Main External Libraries Used*

websocket (ws)	Websockets are used for all inter-node communication to aid simplicity and reliability.
better-sqlite3	To allow storage of application data in a local sqlite database,
dht-rpc	To provide an auxiliary overlay network based on Kademlia DHT. To reduce project complexity of developing and testing a bespoke DHT implementation.
commander	To facilitate parsing and extraction of user provided command line arguments.
debug	To manage the display of debugging information from the different modules.
joi	Used for verification of the user, document and feed schema.
object-hash	To ensure a consistent and uniform hash generation of stringified JSON objects.
tweetnacl	Cryptographic functions for user authentication and item verification.