ST. PÖLTEN UNIVERSITY
OF APPLIED SCIENCES

Redesigning the Network

An assessment of peer-to-peer techniques for blockchain data synchronization

Master thesis

for attainment of the academic degree of

Diplom-Ingenieur/in

submitted by

Jung Clemens, BSc

51825061

in the

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

Supervision

Advisor: FH-Prof. Dipl.-Ing. Dr. Martin Pirker, Bakk.

Assistance: -

Declaration

Title: Redesigning the Network

Type of thesis: Master thesis

Author: Jung Clemens, BSc

Student number: 51825061

I hereby affirm that

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

I hereby declare that

☒ I have used a Large Language Model (LLM) to proofread the thesis.

☒ I have used a Large Language Model (LLM) to generate portions of the content of the thesis. I affirm that I have cited each generated sentence/paragraph with the original source. The LLM used is indicated by a footnote at the appropriate place.

☐ no Large Language Model (LLM) has been used for this work.

Date *Signature*

Kurzfassung

Diese Masterarbeit befasst sich mit dem Entwurf und der Implementierung eines Peer-to-Peer-Netzwerks für eine nicht-öffentliche[1] Distributed Ledger Technologies (DLT)-Anwendung. Die Motivation für diese Arbeit ergibt sich aus der steigenden Verbreitung von Kryptowährungen, die auf Trust und Kryptographie basieren. Während die Konsensmechanismen und Smart Contracts von DLTs bereits intensiv erforscht wurden, wurde dem zugrunde liegenden Peer-to-Peer-Netzwerk weniger Aufmerksamkeit geschenkt. Diese Arbeit zielt darauf ab, diese Lücke zu schließen, indem sie eine Grundlage zur Implementierung eines robusten und effizienten Netzwerks für nicht-öffentliche DLT-Anwendungen schafft.

Der in dieser Arbeit verfolgte Ansatz beinhaltet eine gründliche Überprüfung der Herausforderungen und Probleme, die mit Peer-to-Peer-Netzwerken verbunden sind. Zudem wird eine Analyse der bestehenden DLT-Netzwerke durchgeführt. Ziele der Arbeit waren niedrige Latenzzeiten, eine einfache Einrichtung, Widerstandsfähigkeit und Zukunftssicherheit. Diese Faktoren haben die Auswahl der Technologien für die Netzwerkimplementierung geleitet.

Die Ergebnisse dieser Arbeit umfassen die Implementierungsvorgaben für ein Peer-to-Peer-Netz einer permissioned DLT sowie eine Argumentation einer Technologieauswahl. Der Hauptbeitrag besteht in der Entwicklung eines umfassenden Netzwerkdesigns, das diese Herausforderungen anspricht sowie eine Leitlinie für zukünftige Projekte bietet.

[1] nicht-öffentlich meint in dem Kontext eine DLT-Anwendung welche in der englischsprachingen Fachliteratur als „permissioned“ bezeichnet wird

Abstract

This master's thesis focuses on the design of a peer-to-peer network for a permissioned Distributed Ledger Technologies (DLT) application. The motivation for this work stems from the increasing adoption of cryptocurrencies, which are built on trust and cryptography. While much research has been conducted on consensus mechanisms and smart contracts of DLTs, the underlying peer-to-peer network has received less attention. This thesis aims to address this gap by providing guidance on implementing a robust and efficient network for permissioned DLT applications.

The approach taken in this work involves a thorough review of the challenges and problems associated with peer-to-peer networks, as well as an analysis of existing DLT networks. The key success factors include low latency, easy setup, resilience, and future-proofness. These factors guided the selection of technologies for the network implementation.

The results of this thesis include implementation requirements for a peer-to-peer network. The network topology, data placement, and latency were also identified as key areas of concern. The main contribution of this work is the development of a comprehensive network design that addresses these challenges and provides guidance for future projects.

Contents

1 Introduction

1.1 Motivation

This thesis delves into the intricate challenges and innovative solutions within the realm of permissioned Distributed Ledger Technologies (DLT) in peer-to-peer networks. In our observation, a large body of work is focused on public DLT applications while reduced emphasis is placed on permissioned DLT applications. We also observe that the characteristics of a public DLT application differ greatly from a permissioned DLT application. In our perception these differences in characteristics are often reflected in the consensus mechanism in use, leaving the peer-to-peer network out of the question. Therefore we explored the challenges and technologies of DLT and peer-to-peer networks and reflected on their applicability in permissioned DLT networks.

1.2 Problem Statement

The primary issue addressed in this thesis pertains to the challenges and inefficiencies in peer-to-peer networks, particularly in the context of permissioned DLT. Libraries for permissioned and public DLT applications exist, but as highlighted in this thesis they do not fulfill all criteria outlined in this work. Literature analysis shows problems such as non-determinism in peer-to-peer networks [1], lack of verifiability of programming languages used in smart contracts [2], insufficient protection against attacks [3]–[5], or too fast-paced development [6]. We also notice that many ideas on hardening peer-to-peer against known attacks exist, but are not yet integrated into many solutions. As detailed in the scenario (chapter 3) we are looking for a vastly extensible peer-to-peer network where transport methods (such as TCP/UDP/QUIC) as well as cryptographic primitives can be changed.

1.3 Importance of Peer-to-peer Networks in Distributed Ledger Technologies

Nowadays the wide usage of the terms “cloud computing” and “cryptocurrency” can be observed[1]. Since one of the major selling points of cryptocurrencies is providing trust, a lot of effort is put into the development of consent mechanisms and smart contracts, and less into the design of the underlying network.

This drastically reduced *design* focus of the underlying peer-to-peer network, which is powering the cryptocurrency, opens room for attacks [8] and impacts the performance [9]. In cryptocurrencies, the delay introduced by a bad network design is somewhat acceptable. In a use case where real-time is important this delay becomes a problem.

A large body of work is focused on cryptocurrencies and even less research is focused on permissioned blockchains, therefore, very little is known about the design best practices of permissioned DLT applications.

1.4 Public vs. Permissioned Ledger

As Kolb *et al.*[10] explains, distributed ledgers are differentiated in who can access the peer-to-peer network. Literature differentiates between “public” (or “permissionless” [11]) and “permissioned” (also referred to as “consortium blockchain” and “federated blockchain”) [1], [10], [11]. Permissioned peer-to-peer networks where a centralized authority manages the consent are called “private” blockchains [10], [11].

Permissioned networks have very different approaches to public chains, whereas the research for public peer-to-peer networks applies only partially, and more research on that topic is needed.

1.5 Proposed Solution

The proposed solution in this thesis focuses on developing a guide for implementing a peer-to-peer network for a permissioned distributed ledger application.

We propose an API and implementation guidance that is built on ideas of existing frameworks extending them with various measures to tackle existing problems. We consider this guidance as

[1] As can be seen on Google trends, especially on YouTube search queries [7]

a partial solution since no source code is provided.

1.6 Contributions

This thesis investigates established peer-to-peer networks (e.g.: Gnutella, Napster, Kademlia) and compares their challenges and solutions to the peer-to-peer networks of Distributed Ledger Technologies (e.g.: Bitcoin, Ethereum, IPFS). Guidance on building a state-of-the-art network is given by providing a criteria-based evaluation of technologies.
This thesis makes several key contributions to the domain of DLT and peer-to-peer networks:

- **Fostering resilience:** The proposed solution aims to enhance the network's resistance against failure and attacks. This is achieved through the integration of measures against attacks.
- **Minimizing complexity:** A major contribution of this thesis is the critical reflection on the complexity of the introduced components.
- **Reducing attack surface:** Despite the inevitable increase in the code base due to the introduction of new features, this thesis prioritizes techniques that minimize the additional attack surface. This includes avoiding significant exposure to the network, such as refraining from using distributed hash tables or unsafe serialization techniques.
- **Resistance against misbehaving peers:** A critical aspect of the contributions is the inclusion of auditing the degree boundaries[2] of peers to identify misbehaving peers.

These contributions collectively advance the state-of-the-art in DLT and peer-to-peer networking, offering practical solutions to some of the most pressing challenges in the field.

1.7 Structure of the Thesis

This thesis is meticulously organized to explore the intricate aspects of DLT and peer-to-peer networks. Starting with **chapter 2**, the thesis lays the foundational groundwork by providing an overview of essential technologies and concepts in networking that are crucial for understanding peer-to-peer networks and DLT. It ensures that readers are familiar with fundamental networking

[2] Since a peer-to-peer network can be viewed as a graph the term "degree" is used from the domain of graph theory in literature (examples given of literature using the term in this context: [12]–[14]). Degree boundaries refer to lower and upper limits on the number of connections per peer.

technologies, models, and data structures.

When delving deeper into the background chapter readers find a literature overview addressing the problems and challenges of peer-to-peer networks in **section 2.7** which ends in a summary of known attacks on peer-to-peer networks in **subsection 2.7.14**.

We also consider general knowledge about some established peer-to-peer networks as outlined in **subsection 2.8** as relevant knowledge to understand the problem description and to follow the argumentation of the proposed solution.

Afterwards, in **chapter 3** the reader is presented with a detailed description of the scenario this thesis tackles and **chapter 4** outlines how we approach it. Referring to the approach we continue with outlining some noteworthy existing frameworks in **chapter 5** before we start in **chapter 6** our evaluation of technologies we then use in **chapter 7** to present our guide for implementing a peer-to-peer network.

Finally, the thesis culminates in **chapter 8**, where we summarize the key findings, discuss the broader implications of our work, and propose potential avenues for future research.

Chapter 9 encapsulates the essence of this thesis, tying together the various threads of research presented in earlier chapters.

2 Background

2.1 Assumptions

We assume that the reader of this thesis is aware of basic networking technology such as IPv4, IPv6, TCP, UDP, and DNS. We further assume that the reader is familiar with the ISO/OSI model [15] of network layers. We also expect the reader to be acquainted with basic data structures such as hash tables [16, Chapter 11]. Furthermore, we expect the reader to know the meaning of terms such as cryptographic hashes and encryption.
Since this work tackles problems on the networking layer of DLT core concepts and terms of DLT are only sparsely explained in section 2.2, for further explanation of DLT terminology please refer to the work of Kolb *et al.* [10].

2.2 Core Concepts of Distributed Ledger Technologies (DLT)

2.2.1 Transaction

In the context of DLT, a transaction refers to an action on the ledger (e.g.: sending funds). Transactions are fundamental to DLT technology as they represent the transfer or modification of assets, information, or value within the network. Whether it involves transferring financial assets, recording data, or executing smart contracts, transactions play a central role in the functioning of blockchain systems. [17]

2.2.2 Ledger

In the field of DLT technology, a ledger is a crucial component that functions as an append-only log for data storage [10]. This structure ensures that new entries can only be added to the end of the ledger, preventing any prior entries from being altered. DLT systems use this ledger to accurately

track the ownership and transfer of digital assets, with each transaction representing a new ledger entry.

Depending on the DLT in question these transactions may be bundled into blocks or used directly and stored in one of many data structures (such as blockchains) as explained by El Ioini *et al.* [11]. In simplified terms, the ledger is this data structure that stores all the transactions (complexity is added through implemented optimizations, e.g.: Bitcoin has mechanisms to reclaim disk space [18]). We observe that many people use the term "*blockchain*" when referring to the (distributed) ledger itself. We assume that this is based on the popularity of Bitcoin.

In addition to asset transfers, the ledger can document a wide range of data and transactions, including those with complex, application-specific logic such as smart contracts found in e.g. Ethereum [10]. In these situations, the ledger entries include not only asset exchanges but also programmable operations and data transformations.

The decentralized nature of a DLT system is a crucial advantage, ensuring that no single entity can unilaterally manipulate the data. However, centralizing a ledger in a single location presents challenges, such as the potential for a single point of failure or system bottlenecks. [10]

2.2.3 Smart Contract

A smart contract is a self-executing programmable transaction logic [19]. These contracts run distributed (on distributed ledger technology) and automatically enforce the code. [20]

A detailed explanation of Smart Contracts in Ethereum can be found in subsection 2.8.6.3.

2.2.4 Proof of ...

Due to the distributed nature of DLT, the ledger needs a form of consensus mechanism to ensure data consistency across nodes. We observed that the name of these consensus mechanisms often starts with "Proof of". Proof of Work (PoW) is explained by [10], [18], Kolb *et al.* further explaines Proof of Stake (PoS), Proof of Elapsed Time (PoET) and Proof of Authority (PoA) [10]. A consensus is also designed to withstand adversaries and attacks [10], [21].

Bitcoins' Proof of Work (PoW) in simplified terms operates by nodes called "*miners*" [22]. Miners collect open transactions to form a block. The data of the block (previous hash, a nonce, and transactions) is hashed. To generate a valid block a cryptographic puzzle that involves the nonce and the resulting hash needs to be solved. The hash needs to satisfy the "*difficulty*". The difficulty

is a parameter for the block creation which is adjusted automatically by the network protocol[1] to maintain a constant rate at which blocks are created. In Bitcoin, the difficulty describes the amount of zeros at the beginning of the blocks' hash [18]. To change the cryptographic hash of the given block miners exhaustively search for an appropriate value for the nonce field. Therefore, for a miner to append a block to the ledger a nonce must be found, which allows the hash of the block to satisfy the difficulty. Due to the nature of the cryptographic hashes in use, miners can only search exhaustively for a matching nonce, which costs a lot of computing power or "work". [10], [21]
We cover Ethereum's transition from PoW to Proof of Stake (PoS) in paragraph 2.8.6.5.3.

2.2.5 The Term Distributed Ledger Technologies (DLT)

Many technologies using peer-to-peer networks to distribute a common ledger employing a consensus protocol have emerged. Often these technologies are called Blockchain or Directed Acyclic Graph (DAG) due to the data structure of the ledger. In literature the umbrella term Distributed Ledger Technologies (DLT) prevailed[2]. In short, we summarize DLT as DLTs that combine block/DAG technology with peer-to-peer networks and a consensus algorithm.
DLT represents a significant shift in the way information is gathered and disseminated. It involves replicating and distributing a ledger across multiple entities, thereby decentralizing ledger control. This methodology enhances the ledger's resilience and security by preventing any single entity from independently altering the ledger's state, addressing problems inherent in systems where the ledger is maintained by a single entity. In a DLT environment, each participant must maintain an exact replica of the ledger. Any amendments to the ledger require consensus among the majority of the participants. This framework offers several benefits over centralized systems, including uninterrupted operation during limited failures, enhanced performance through load distribution across various replicas, and protection against manipulations by malicious entities. While this introduces new complexities, such as the upkeep of consistent ledger copies and the need for consensus on each ledger modification, it also offers the benefit of universal participation.
As previously mentioned DLT applications can be public. These public DLTs are designed so that participants without mutual trust can participate. These systems enable any individual to propose

[1] The difficulty is a crucial aspect of PoW systems, aiming to ensure a relatively constant rate of block creation despite changes in the overall network hash rate. [23]

[2] e.g. [11] or [24] use the abbreviation DLT, many other publications also refer to this technology as a distributed/decentralized ledger, some examples are given in [1], [20], [25]

a transaction, maintain their version of the ledger, and contribute to transaction processing for ledger updates. Public blockchains are challenging to govern and manage due to the absence of a central authority and inherent distrust among participants. Any changes to the blockchain's functionality requires consensus among the majority of the network's miners (who need to update the software), which can be difficult to achieve. This decentralized and trustless characteristic is in stark contrast to private blockchains and traditional distributed databases. In these cases, participant authentication is mandatory, and a consensus mechanism is essential to establish the sequence of events. [10]

2.3 QUIC

QUIC[3] is a UDP-based transport with major improvements over TCP and UDP such as multiplexing and security. As it has been standardized very recently (2021), we cannot assume that readers are familiar with the standard. Security in this context is not limited to transport security but also addresses issues like amplification attacks. [26] defines "The Transport Layer Security (TLS) Protocol Version 1.3" [27] to be used as the protocol for transport layer security. For more information, please refer to "*RFC 9000 QUIC: A UDP-based multiplexed and secure transport*". [26]

2.4 Peer-to-peer Networks

A peer-to-peer network is in general terms a network of multiple independent computers communicating with each other utilizing a protocol. A peer-to-peer network in a technical context is somewhat similar to a peer network in a social context in a way that coequal subjects communicate with each other. This technique in an information technology context is not new whereby the first mention originates from the year 1991 [28]. A program executing the network protocol is called node or client [29].

Peer-to-peer networks often exist as an overlay network, such as a file-sharing network based on top of TCP/IP. In mass media, peer-to-peer networks typically refer to (illegal) file-sharing [30]. However, in this work, we are referring solely to the underlying technology.

In terms of connections, peer-to-peer networks can be seen as a graph. In this analogy, the nodes are referred to as vertices. Connections correspond to the term edges and the degree of a vertex

[3] according to the standardizing RFC 9000 QUIC is not an abbreviation [26]

indicates the number of connections. With that translation in mind, the field of graph theory can be applied to peer-to-peer networks. Vyzovitis *et al.* [31] state that the desired degree of a network is the number of peers that a node holds a direct connection to. In GossipSub the degree is also called the “amplification factor” [31]. As shown by Singh *et al.* [32] the degree of overlay nodes, which help clients get established in peer-to-peer networks, needs to be carefully controlled to defend against Eclipse Attack (subsubsection 2.7.14.2).

2.5 Distributed Hash Table

2.5.1 DHT in General

According to Maurer *et al.* [33] multiple forms of hash tables exist, all of which provide an efficient data structure for key-value pairs. The key is hashed and used for indexing the data, resulting in access times that are independent of the amount of data. [33]
As Tiendrebeogo *et al.* [34] describes, DHTs provide technology comparable to a primitive hash table, commonly used in many applications. DHTs distribute the data of a hash table across multiple nodes. Multiple implementations for such DHT exist, examples are Kademlia[4], CAN[5], Chord[6], and Pastry[7]. DHTs are used in many DLT applications like Ethereum (see subsection 2.5.2) [40] and also used by common (mostly highly distributed filesharing) solutions like BitTorrent [41].
S/Kademlia [42] addresses some of the attacks against Kademlia including but not limited to signing messages, introducing crypto puzzles, and introducing a minimum number of siblings.

2.5.2 Application of DHT in DLT

Ethereum’s execution layer utilizes Kademlia and a set of hardcoded bootstrap nodes for node discovery. Kademlia’s “FIND_NODE” call is used by Ethereum to query information about the other nodes peer list. Constructing the Kademlia overlay and adding a handshake aswell as encryption is done by Ethereum’s RLPx layer. [43], [44]
In Ethereum 2.0 the protocol used for the node discovery is a modified form of Kademlia [45].

[4] Kademlia was created 2002 by Maymounkov *et al.* [35] and it’s implementations are often referred to as KAD [36]
[5] Content-Addressable Network (CAN) was created by 2001 by Ratnasamy *et al.* [37]
[6] Chord was created 2001 by the Massachusetts Institute of Technology (MIT) (Stoica *et al.*) [38]
[7] Pastry was created 2001 by Rowstron *et al.* [39]

The InterPlanetary File System (IPFS) acknowledged the vulnerabilities introduced by using standard Kademlia[8] and instead proposed the usage of S/Kademlia [46]. Henningsen *et al.* [4] was not able to confirm that S/Kademlia is used at all in IPFS.

2.6 Serializing Data in Communication Protocols

2.6.1 Need for Serialization

Programs hold information in a computer's working memory. Serialization is a well-known term in programming describing how the data from the working memory is written onto a medium, to be read later[9]. In peer-to-peer networks nodes need to exchange information over the network, therefore, data such as transactions needs to be serialized. Since messages that are part of a communication protocol usually are not static strings but also include data, we consider these messages as objects to serialization. We also consider messages of a communication protocol as part of the serialization concept, because these messages are sent over the network and reassembled and interpreted by another node exactly as described by the concept of serialization. Since the deserialization involves data originating from a different source, one must treat them cautiously. Two main approaches to serialization exist: serializing data from the memory into a human-readable stream of text as discussed in subsection 2.6.2, or the serialization into a non-human readable but very compact stream of binary data as discussed in subsection 2.6.3.

2.6.2 Serialization into Text

2.6.2.1 XML

Extensible Markup Language (XML) is widely used in various applications, including office tools to store documents (such as *.docx), XML-based databases (such as MarkLogic, eXist) and web protocol standards (such as SAML, SOAP) [48].
Furthermore, XML documents can undergo validation against a schema to ensure adherence to pre-defined standards regarding structure, data types, and values. This validation, often carried out using tools like XML Schema Definition (XML) or Document Type Definition (DTD), serves to

[8] Attacks on Kademlia are discussed in Baumgart *et al.* proposal for S/Kademlia [42]

[9] [47] describes the concept of serialization as "Object serialization is the process of writing the state of an object to a stream. Deserialization is the process of rebuilding the stream back into an object."

confirm that the document is well-formed and aligns with the guidelines outlined in the schema. Ultimately, this process enhances the security and reliability of XML data handling processes.
According to Späth *et al.* [48], the XML parser is a critical component with a history of known vulnerabilities dating back to 2002. These vulnerabilities include susceptibility to attacks like the so-called Billion Laughs and XML External Entity (XEE) attacks. Specifically, the Billion Laughs attack exploits internal General Entities of XML to create an exponential entity attack, relying on nested but limited levels of entity recursion. This vulnerability can lead to a file as small as 200 kilobytes growing to several gigabytes during parsing, exposing the system to various threats such as Denial-of-Service (DoS), Server Side Request Forgery (SSRF), and File System Access (FSA).

2.6.2.2 JSON

JavaScript Object Notation (JSON), a data serialization format that employs the object notation of JavaScript, is gaining popularity as a more efficient alternative to XML due to its simpler syntax and reduced file size. It typically results in smaller files compared to equivalent XML encodings. Its comprehensibility and ease of use surpass those of XML, and it has gained prominence as a preferred language for data representation in various applications. [49].
While JavaScript Object Notation (JSON) is a lightweight data format mainly utilized in web-based data exchanges, particularly in Application Programming Interface (API) communications, its popularity has grown due to its simplicity and readability for both people and machines. In recent years, a schema language called JSON Schema has been proposed and formalized to promote seamless data exchange between different systems [50]. In addition, JSON serves as a self-describing data serialization format that incorporates metadata about the data being transmitted into the message payload, making it easier to parse for both human readers and automated systems [51].
JSON parsers are susceptible to some forms of attacks as detailed by Muñoz *et al.* [52].

2.6.3 Serialization into Binary

2.6.3.1 Direct Binary Serialization

Binary serialization is the process of storing data or a data structure from the working memory as a stream of bytes. With direct serialization, we mean writing a data structure's content as is to a storage medium or network (e.g. 4 bytes version number followed by 32 bytes for the previous' blocks header followed by a 4 bytes none, ...) without any context or field identifier. While direct

binary serialization creates no overhead in terms of the volume of data that needs to be transported, there are several drawbacks to this technique.
Due to a lack of header information, structure, or length of the data, the program receiving the data has to know this information during the deserialization process. In addition, errors during the transmission of data are not as easily detectable. Future changes to the message structure are not possible without changing the program.
Albeit a crude option for data serialization, it is used by Bitcoin [53].

2.6.3.2 TLV

The Tag-Length-Value (TLV) encoding scheme is a well-known method used in binary-based serialization. It is a sequence of Type, Length, and Value fields. In this method, the Type field indicates the kind of information being encoded, the Length field specifies the size of the information field in octets, and the Value field encapsulates the information itself. [54]
This structure facilitates concise and secure data representation, allowing messages to be parsed quickly and efficiently, a necessary feature to optimize data transfer in performance-critical applications. [55]
However, a drawback of this method is that the format is not self-describing, making it dependent on specialized tools for interpreting and displaying message content [54]. The system remains extensible, an attribute essential for a robust encoding technique [55].
In our understanding, the Recursive-Length Prefix (RLP) used in Ethereums' RLPx layer is a simplified version of TLV where the Type and Length field are merged [56].

2.6.3.3 ProtoBuf

Protocol Buffers [57], commonly referred to as ProtoBuf, is a binary-based, language- and platform-neutral mechanism for serializing data. ProtoBuf's ecosystem consists of descriptions of the data structure in ".proto" files as well as a compiler that generates code for the desired programming language that handles the serialization, deserialization, and data structures. It generates native language bindings and provides a serialization format for packets of typed, structured data. Protobuf messages and services are described by engineer-written ".proto" files. [57]
However, one downside is that data sent in a binary format cannot be parsed unless the receiver has the ".proto" file or the necessary include files for that programming language, making it less

adaptable in certain situations [58].

We do not consider Protobuf messages as fully self-describing, because they do not include field names or types in their encoded form. Instead, Protobuf uses a separate schema to define the structure of the data, from which the code is derived. This makes Protobuf output smaller and faster than text-based serialization formats.

In terms of security, Protobuf is designed to be immune to buffer overflows and similar attacks. The format is also defined to be deterministic, meaning that encoding the same data multiple times should result in the same encoded output. Protobuf is widely adopted and used for inter-server communication as well as for archival storage of data on disk.

Although Protobuf is extensible, adding new fields to messages requires changes to the schema and recompilation of the code. This can make extending Protobuf messages more difficult than other serialization formats. Protobuf also defines how edge cases, such as repeated fields, are handled. For example, repeated fields are encoded as a repeated sequence of their type, and string encoding is defined as utf8. Despite its advantages, Protobuf is not immune to vulnerabilities. For example, a deserialization vulnerability has been found in Protobuf that could allow an attacker to execute arbitrary code. [57]

2.6.3.4 BSON

Binary JSON (BSON) is a compact binary encoding format. Litt *et al.* [59] demonstrated that BSON can significantly improve the performance of schema-less document storage in systems such as PostgreSQL. [59]

BSON was designed to have three main characteristics: lightweight, to keep spatial overhead to a minimum; traversable, to be easily traversed; and efficient, so encoding data into BSON and decoding from BSON can be done very quickly in most languages. [60]

Research by Litt *et al.* [59] shows that queries on BSON document keys are more than eight times faster than those on JSON documents, demonstrating BSON's ability to provide a significant performance boost in data retrieval operations. As a result, BSON continues to be a preferred choice for efficient data serialization, offering both speed and storage efficiency. It enables faster traversal of JSON-like documents compared to JSON, thanks to its transformation into a binary blob for storage, expediting the data serialization process.

Unlike JSON, which stores numbers as character strings, BSON simplifies storage by treating numbers as 32- and 64-bit integers or 64-bit double-precision floats, effectively optimizing memory us-

age and operational efficiency, with a restriction on maximum document size of 16MB. This design reduces computation time during traversal by using encoded metadata to bypass elements instead of a character-by-character key search and minimizes I/O time by offering more compact storage for data types. [59]

2.6.3.5 CBOR

The Concise Binary Object Representation (CBOR) has emerged as a prominent binary serialization specification, first documented by Bormann *et al.* [61] in an Internet Engineering Task Force (IETF) paper in 2013[10].

Designed primarily for Internet of Things (IoT) applications, it is notable for its schema-less structure and resource efficiency, making it particularly suitable for use in IoT and other resource-constrained environments. In addition to minimizing code and message size, it also generates a compact representation that easily conforms to JSON and various other data formats [62].

CBOR is recognized for its standardized approach, which is rigorously described in an IETF RFC document after extensive technical review, making it a more reliable choice for binary serialization than others. It supports a wide variety of data types, including integers, floats, booleans, strings, arrays, maps and tags. In addition, it facilitates cryptographic operations, message authentication codes and encryption, all of which contribute to its versatility in serializing and transmitting data [63].

One notable application of CBOR can be seen in the European Union (EU) DigitalGreenPass Certificates during the COVID-19 pandemic, where CBOR's efficient data representation became critical in handling large amounts of data related to COVID-19 vaccination authentications [64]. As well as providing a platform for signature development and management, it seamlessly integrated with the CBOR Object Signing and Encryption (COSE) framework to further enhance security functionality [62].

Despite its commendable features, CBOR is not without vulnerabilities, particularly concerning potential resource exhaustion attacks. Such challenges can manifest themselves as attackers tricking a decoder into allocating large data elements or exhausting stack depth by creating deeply nested elements, potentially exploiting integer overflow vulnerabilities, or remotely crashing a node. To counteract these problems, it is imperative to establish robust resource management within the decoder, introduce limits on the size of allocatable data items, and develop protocols that restrict

[10] notice that the 2013 version of this RFC ([61]) is superseded a version from 2020 ([62])

the possibility of multiple interpretations of a CBOR data item, thereby effectively mitigating security risks [62].

2.7 Problems and Challenges

2.7.1 Problems known already in Literature

Since the topic of challenges and problems in peer-to-peer networks is a broad field and may use different taxonomies. This section reviews literature from which we extract and differentiate problems (see problem definitions in subsection 2.7.2) which are later discussed in more detail (see subsection 2.7.3 to 2.7.14).
As for risks and or problems regarding peer-to-peer networks as used in DLT the literature acknowledges the following:

- Yamashita *et al.* [1] cites problems regarding the throughput of the Hyperledger Fabric network and identifies non-determinism from language instructions and non-determinism arising from outside the blockchain as problems.
- Daswani *et al.* [65] addresses the peer-to-peer topics of network topology, data placement, message routing, expressiveness, comprehensiveness, autonomy, efficiency, quality of service, robustness and anonymity.
- Shalini *et al.* [3] summarizes attack vectors in DLT environments such as double spending, transaction adaptability, attacks on the network, selfish mining, fork after withholding attacks and block withholding attacks.
- Vranken [66] demonstrates implications of computing speed-dependent algorithms such as Bitcoin's PoW when orders of magnitude faster hardware are available.
- Krishna Ramanathan *et al.* [12] published work tackling problems finding "good" peers in a network whereas this is determined by means of how to connect to the group, searching the network, connections to intermediate peers, the peer's importance and stability.
- Henningsen *et al.* [4] encountered missing security proposal implementations in the underlying Kademlia DHT as well as littler and more asymmetrical connections on NATed nodes while mapping the IPFS network.
- Tigelaar *et al.* [67] provides a literature overview of information retrieval in peer-to-peer networks in 2012, citing efficient resource usage, acceptable service quality, robustness, data availability, anonymity, churning peers, free-riding peers, malicious peers, lack of simula-

tion frameworks and a lack of standardized tests as challenges of peer-to-peer networks, as well as quoting latency, the freshness of the data index, and the need for common evaluation frameworks as future peer-to-peer challenges and compiling some key focus areas for designing future networks.

- Baumgart *et al.* [42] cites vulnerabilities in distributed hashtables and not only provides security improvements to the Kademlia DHT, but an overview of the routing layer attacks such as Eclipse, Sybil, Churn, Adversarial routing, Denial-of-Service (DoS) and attacks on the data storage.
- Brotsis *et al.* [2] points out various problems in the hyperledger fabric framework, such as vulnerabilities in smart contracts and the specification, network security and privacy issues.
- Wallach [5] addresses the issue of non-secure routing primitives in peer-to-peer networks as well as the closely related problem of assigning node ids and fair resource sharing.
- Henningsen *et al.* [68] demonstrates the execution of Eclipse attacks on the Ethereum network.
- Liu *et al.* [69] proves that even with knowledge of the physical topology and all peering nodes finding an optimal topology is NP-hard. Furthermore, Liu *et al.* summarizes topology challenges and message duplications.
- The work of Singh *et al.* [32] addresses the Eclipse and Sybil attacks in structured and unstructured overlay networks.
- Lua *et al.* [14] did a literature survey of peer-to-peer network schemes and compared the overlay schemes based on their solution to the problems of decentralization, their architecture, lookup (including latency), routing performance, routing scalability, churn, security and reliability (including free riding).
- Vu *et al.* [70] forms a taxonomy after reviewing and documenting multiple peer-to-peer protocols and characterizing them.
- Paavolainen *et al.* [71] summarizes the Ethereum protocol and client types as well as a definition of adversary attacks and attack scenarios resulting in an assessment of the security properties of light clients.
- Triantafillou *et al.* [72] highlights the challenges of peer-to-peer (content-sharing) networks and proposes a new solution based on clustering and fairness. In the literature reviewed by Triantafillou *et al.* challenges such as query routing, object location, load balancing, response time, hop count, and the system architecture (e.g. pure peer-to-peer or networks with central

elements) are addressed.

- Kuhn *et al.* [73] not only provides a great literature overview of the problems emerging from peers joining and leaving networks also known as churn. Naik *et al.* [9] revisits the churn aspect of modern and newer networks demonstrating the improvement made between 2010 and 2020, while still acknowledging that the presented churn-resistant protocols need "*more practical implementations to be widely employed*" [9].
- Marcus *et al.* [8] not only demonstrated that there is a significant lack of research addressing the underlying peer-to-peer network of blockchain technologies but also demonstrated the implications of a MiTM attack utilizing BGP such as an Eclipse Attack. Marcus *et al.* [8] also showed the attack surface of time, demonstrating that a client can be eclipsed or erased from the network when its (network) time source is compromised. As Vyzovitis *et al.* [31] pointed out in a sidenote: avoiding NAT issues is still a relevant topic in peer-to-peer networks.

2.7.2 Problem Definitions

Based on this literature analysis we summarize the problems of peer-to-peer networks as the following:

Non-determinism of chain code : As described by [1] non-determinism refers to the lack of verifiability of the programming languages in use for smart contracts, which could lead to inconsistent states.

Non-determinism from outside the peer-to-peer network : As described by [1] non-determinism may arise from external web services, system commands or file access.

We consider many of the aspects related to light clients as demonstrated by [71] as non-determinism from outside the network. We do not consider interlinked networks[11] as non-determinism from outside the network.

Our reasoning for this is that both interconnected networks share the same peer base with the same latency and security restrictions, while calls to external web services are likely to take different routes through the Internet than the peer-to-peer network itself.

Network topology : We summarize multiple terms as "network topology". Daswani *et al.* [65] barely discusses network topology but clearly demonstrates the author's understanding of it and its relevance in different scenarios. Triantafillou *et al.* [72] provides a more normative approach sharing the author's understanding of the term "topology" and mentions super peers

[11] e.g. the usage auf DHT in DLT as described in subsection 2.5.2

and IP-based clustering as an example [72]. Following the distinction made by Triantafillou *et al.* in [72] we only considered the overlay topology for this criterion, excluding the physical network topology. We argue this decision by the as-is state of the physical topology and security considerations outlined by [32] and [8] suggest that peers shall be treated by their connection degree (hence the author concludes that considering the physical location may enable an attacker to perform eclipse attacks). Clustering as an architectural choice is also mentioned by Tigelaar *et al.* in [67] and is considered as "network topology" by the author. The "Decentralization" aspect highlighted in [14] is also considered as "network topology" by the author since Lua *et al.* uses it to refer to structured or unstructured networks, including the super-peer architecture. We exclude the "Architecture" mentioned in [14] since Lua *et al.* used it more commonly for aspects related to message routing. We also consider the "basic structure" as referred to by Kuhn *et al.* in [73] as network topology.

Data placement : As highlighted by multiple sources placing data in a peer-to-peer network (especially in a DHT) is a challenge [65], [67], [72], [74]. One can assume that certain aspects of "*availability*" addresses data placement. Since the term *availability* is used in many publications with ambiguous meanings we split the term into multiple sub-challenges such as data placement. We understand all availability aspects arising from the lack of accessibility of data or insufficient distribution of data across the network as related to the problem of data placement. We also account for availability in a data placement context when used to describe proactively adding content replicas to mitigate churn effects. We do not understand any availability problems arising from (unexpected) churn or overloads such as DoS.

Latency : As many sources demonstrated latency is an ongoing challenge in a peer-to-peer network [1][53]. However, the reader may encounter different wordings for *latency*. We understand latency as described by Tigelaar *et al.* with the related effects [67]. Tigelaar *et al.* use the term in multiple contexts in [67] including the following description of latency we like to quote: "*Long paths [in a networking context] are expensive in terms of latency, and slow network links and machines worsen this*." [67]. We also assume that the underlying problem tackled by *delay tolerant networks* such as described by Chau *et al.* is latency [13]. Based on Lua *et al.* we also assume that latency is taken into account when designing protocols [14]. We also assume that solutions based on caching (e.g. [46]) are very closely linked to latency, even without being explicitly stated. We also encountered "propagation delay" as a commonly used phrase when referring to latency [72][53].

However, we do not understand latency in the context of delay introduced by accessing mass storage devices [75], nor do we understand latency as in some arbitrary bottleneck of the network chosen by the developer (e.g. self-regulating difficulty values in bitcoins protocol to ensure a constant amount of blocks generated per hour [18]). We argue that decision because we find that slow peripherals are more related to computing power rather than the peer-to-peer network inherently. We also argue that some programmed constants are not dependent on the network's size, capabilities, or state and therefore are not related to our understanding of latency.

Free riding : Many network designs assume that all nodes contribute resources, but as shown by Tigelaar *et al.* [67] this is not true.

Broadcasting : We define broadcasting as delivering a message to all peers in the network as described by [31], [53], [76]. We only consider methods that attempt to reach all nodes in the network as broadcasts. We made a distinction between *searching* and *broadcasting* since searching is inherently a different problem, although we acknowledge that this problems are very closely linked as pointed out by Naik *et al.* [9].

Searching : Searching is one of the key problems in file-sharing networks [14]. We only understand searching as the mechanism required to retrieve information (as exemplified by the elaboration concerning the index structure in [70]). We notion querying a database such as the global state of Ethereum as *searching* [10]. In the context of this thesis we also notion problems such as expressiveness of the query language or completeness of the query language as synonymous with the problem of searching a peer-to-peer network (e.g. [65] distinguished those sub-problems).

Churn : Churn in peer-to-peer networks is closely related to the meaning of the English word. As a non-native English speaker, the expression churn was unknown to some of us, hence we want to point out that churn is equivalent to stirring, agitating, or being turbulent. We understand churn in peer-to-peer networks as the rate of peers (re-)joining and leaving the network [9]. Daswani *et al.* [65] describes churn as "robustness", while Vu *et al.* [70] uses the term "availability" to describe the availability of data when peers leave the network. Since does not use "availability" in the context of proactively spreading the content in the network we assume churn as the underlying problem [70].

Message routing : Message routing in a peer-to-peer network is one of the most cited challenges (e.g. [4], [5], [14], [24], [32], [65], [67], [77] address this topic). For this work, we stick to the

simple explanation given by Daswani *et al.* [65] "*message routing defines how messages are propagated through the network*" [65]. We summarize the term "query routing" into "*message routing*" (e.g. as used in [67]).

Network Address Translation : Network Address Translation (NAT) is a problem in peer-to-peer networking where nodes can initiate a connection on their own but other nodes may witness this node as unconnectable. This one-way connectivity presents a problem to peer-to-peer networks, as mentioned by [30], [31], [68].

Bootstrap : We understand the bootstrap process as a means that enables a node to join the network e.g. by using hardcoded DNS pointers [29].

Processing power : We see processing power as one of the challenges in peer-to-peer networking. We understand many aspects as *processing power*. In summary, we summarize necessary computations as processing power. Examples of these necessary computations are mining in PoW consensus [20] or the calculation of erasure codes [75]. We also see a close relation between processing power and Jafari Navimipour *et al.* definition of bottleneck, as we assume that the absence of processing power results in a bottleneck as described in [78]. We do not consider architectural improvements that increase the transaction throughput as "processing power" although they are often described with the term "Performance" (e.g.: [79]).
We only consider a lack of processing resources as a processing issue. We understand that the overall performance of the peer-to-peer network is not just limited by the processing capabilities. Therefore we assume that limits of transaction processing that can not be explained by any other problem listed above originate in a lack of processing power.

2.7.3 Topology of peer-to-peer networks

Vu *et al.* [70] provides a very good overview of the structures of peer-to-peer networks:

Centralization peer-to-peer networks exist in a centralized or decentralized fashion. Centralized networks are easier to orchestrate and bootstrap, but at the same time, more error-prone since a single point of failure is introduced.
A centralized peer-to-peer network has some nodes that differ from the rest, Vu *et al.* [70] exemplifies Napster or BOINC as a representative example. In BOINC (Berkeley Open Infrastructure for Network Computing) [80] a project is identified by a single "master URL" where servers provide tasks to be done by participating clients. In contrast to centralized networks, a decentralized network is characterized by peers having equal rights and responsibilities

as stated by Vu *et al.* [70]. Daswani *et al.* [65] quotes Gnutella as an example of a fully decentralized network.

Triantafillou *et al.* [72] refers to fully decentralized networks as "*pure peer-to-peer*" networks.

Hierarchy peer-to-peer networks can have a sort of hierarchy (e.g. flat or 2-tier structure). Hierarchy can be assigned static or established over time (e.g. good-behaving nodes can be promoted to supernodes). Statically assigned super-peers are somewhat a compromise between centralized and decentralized but also inhibit the same problems and benefits as a centralized solution. Promoting nodes to supernodes introduces complexity and therefore attack surface. Triantafillou *et al.* [72] further distinguishes between a hybrid approach (e.g., superpeers) and node clustering. Tigelaar *et al.* [67] also mentions clustering as an architectural choice in various implementations. In reference to [67], we consider assigning nodes different roles as a hierarchical topology.

The structure of the network greatly affects the behavior under churn, executing (search-)queries and propagation of information.

Krishna Ramanathan *et al.* [12] describes a method for (re-)organizing a peer-to-peer network by constantly monitoring neighboring peers and changing neighbors to be closer to "interest" groups, while Kuhn *et al.* [73] organizes peers in "hypercubes".

A peer-to-peer network can be viewed as a graph and hence share some key indicators. One of the most cited is the *degree*. Wang *et al.* [81] analyzed the degree on Gnutella and Kuhn *et al.* [73] suggests that a sufficiently high degree equals resilience towards churn. Magharei *et al.* [82] studied *degree* and concluded that "increasing peer degree improves the connectivity among peers but reduces the value of bandwidth-per-flow (or) for each connection" and that the lower bound for peer degree can be assumed as 6 and the upper bound is limited by bandwidth [82].

The structure of a peer-to-peer network itself is listed as an open problem since no "perfect" structure is known and each approach implies other strengths and weaknesses. Therefore picking a structure contributes a lot to the overall characteristics of the network such as behavior under churn, or bootstrapping. The DLT world has not come to a conclusion (e.g. Bitcoin/BTC is fully decentralized, Ripple/XRP uses centralized validating servers, and both networks host a cryptocurrency) [83].

2.7.4 Data Placement

Peer-to-peer networks usually need some form of data to operate. Placing that data in the network is a challenge and design decision. As Daswani *et al.* [65] points out data/index placement can impact query speed and as Tigelaar *et al.* [67] points out, placing the data affects availability under failure or churn situations [65], [67]. Tigelaar *et al.* [67] lists the following approaches to data placement:

- *Centralized inidces* (e.g. Napster, see centralized peer-to-peer networks in subsection 2.7.3)
- *Distributed indices* such as DHT (see section 2.5)
- *Strict local indices* (not distributed)
- *Aggregated local indices* (e.g. super peers which aggregate data from their leaf nodes)

Pro-active approaches of creating duplicates exist, such as Filecoin, which "*works as an incentive layer on top of IPFS*" [74]. In DLT it is usual to eliminate the problem by making all nodes keep a copy of the ledger [18] (nodes that do not keep a full copy of the ledger are referred to as *light client* [71]).

We summarize approaches of data placement as:

- Centralized index
- (Fully) Distributed index
- Local index (not distributed)
- Aggregated using topology
- Incentivized duplication
- Fully replicated to all nodes
- Delegated replication (light clients)

2.7.5 Latency

Latency is a telecommunication problem, mainly cited in the context of Mobile Ad hoc Network (MANet) and Delay-Tolerant Networking (DTN) but is not directly applicable to peer-to-peer networks, because MANet and DTN mostly focus on routing whereas latency is often referred to as an unmitigable problem in peer-to-peer networks (e.g. measured an addressed but rarely solutions are proposed). In peer-to-peer networks, latency also affects the choice for the consensus and its stability [79]. Chau *et al.* [13] describes scenarios and formalizes the problem. Vyzovitis *et al.* [31] presents GossipSub that combines a push and pull approach and points out that Bitcoin uses

flooding to compact malicious nodes, message delays or packet loss. Benet [46] uses caching (in combination with a merkle-DAG for removing duplicate blocks) to combat the connection delay or failure of a node. Tigelaar *et al.* [67] proposes content-based data placement to reduce latency by exploiting locality on a logic level. Mischke *et al.* [84] cites approaches that use a multi-tier structure for latency reduction utilizing high bandwidth peers with low latency as a landmark. Budhkar *et al.* [85] uses a similar approach whereas peers are selected by upload capacity and round trip time. Kalogeraki *et al.* [86] proposes replication along the edges of the network closer to the user. Naik *et al.* [9] cites that La-Nice uses clusters and peer selection to reduce overall message delays.

2.7.6 Free Riding

Free riding (often also referred to as leeching) is a state where nodes only use resources from the network without contributing back to the network. Peer-to-peer networks rely on the contribution of each node to function properly. Therefore, counter methods for free riding evolved:

Monetary : Paying for a service is a very simple but effective approach. Its effect is cited in various papers as explained in [87], [88]. In some cases, the whole monetary incentive layer for other systems is created as cryptocurrency: "*Filecoin works as an incentive layer on top of IPFS*" [74].

Reciprocity : Peers can keep track of the behavior of their direct neighbors and penalize misbehaving peers or prioritize good peers. This can also be described as a bartering economy and is described in [87][89][88].

(Global) **Reputation** : Peer behavior can be tracked and each node can query the reputation of its peers. A prominent example of this is BitSwap. [87][89]

2.7.7 Message Broadcasting

True broadcasting happens when all receivers are on the same physical medium as the sender. The most prominent example is FM radio where all devices receive the same physical signal from the transmission tower [90], [91].

Peer-to-peer networks exist on a logical level by connecting peers from different networks. Therefore in a peer-to-peer network broadcasting needs to be solved by logic instead of physics.

Few enabling technologies for broadcasting in peer-to-peer networks are currently known:

Flooding : A receiving node sends a previously unseen package to every other node (except the

node it received the package from).

Gossip : (or epidemic) A node sends a previously unseen package to a defined amount of randomly selected nodes. This approach is also known as *gossip* or *random network coding*. See subsubsection 2.7.7.3 for clarification.

(Random) Walk : As described in subsubsection 2.7.7.4, a single message traverse the network randomly.

In peer-to-peer networks, the whole package or just the info about the availability of a new package is transmitted to other nodes. These approaches are known as pull and push. Hybrids are also known.

Decker *et al.* [53] provides a good overview of the communication technology used by Bitcoin.

2.7.7.1 Flood

The most straightforward approach to broadcasting is to send the message to all other peers [92]. According to [93], this is the approach *Gnutella* is using. The description found on the Bitcoin-wiki "*When someone sends a transaction, they send an inv message containing it to all of their peers*" implies that Bitcoin works in a similar way [94].

Bitcoin's exact broadcast mechanism has changed over the years from a "flood" like mechanism to a random walk [95].

2.7.7.2 Tree-Based

As described by Dayanand *et al.* [92] networks exist where broadcasting is based on a tree, in a way that each node decides how many copies are forwarded to each neighbor in an attempt to utilize the available bandwidth as best as possible without creating duplicates.

2.7.7.3 Gossip

At the time of writing gossip/random walks/epidemic broadcast is widely used in peer-to-peer networks. Examples are Hyperledger [19], IPFS [96], and Ethereum 2.0 [31]. Arunachalam *et al.* [97] describes the gossip process in detail and Cuenca-Acuna *et al.* [98] describes it in more simple terms. The gossip process, in general, can be described as peers detecting changes in the world state and pushing these "*rumors*" to n randomly chosen peers. Due to the possibility that rumors

may die out, peers poll other randomly chosen peers for changes. Cuenca-Acuna *et al.* demonstrates the scalability of gossip and its capabilities as a content-addressing network.

2.7.7.4 Random Walk

As described by Arunachalam *et al.* in [97] a random walk sends the message to a single randomly selected peer while Gossip sends the message to multiple randomly selected peers. These random walks perform as well and in some use cases even better than flooding as Gkantsidis *et al.* [99] has proven.

2.7.8 Searching

Searching peer-to-peer networks is one of the most referred problems in literature. Example given: Schlosser *et al.* tackles the problem by organizing the nodes as "hypercube" [76], Kalogeraki *et al.* tackles the problem by extending Gnutella's protocol [86], and Jafari Navimipour *et al.* compares resource discovery techniques in peer-to-peer networks [78].
However, DLT is not affected by this problem since all participants hold a copy of the ledger (see also [10, p 18]) and therefore can search locally. Because of performance or hardware restrictions light clients are used which "uses another node for state management and retrieval" [71]. As described by Paavolainen *et al.* subprotocols are used for the communication between these clients and (multiple) network nodes allowing *light clients* to operate on the chain and verify the chain by tracing down to the chain's root [71]. Since *light clients* query the node and do not perform searches on the peer-to-peer network, they do not address the problem of searching a peer-to-peer network, and thus the claim that DLT is unaffected holds true.

2.7.9 Churn in peer-to-peer Networks

Churn describes the effect of the permanent joining and leaving of peers in a network. Stutzbach *et al.* observed that most peers have a short uptime [100]. As Stutzbach *et al.* describes a churn effect should be gracefully handled and long-lived peers can and should be identified [101][77]. Kuhn *et al.* proposed organizing nodes as "hypercube"[12] to get resilient to churn [73]. Meng proposes to combat churn by utilizing a hierarchy (super-peers) to guess and handle churn [102]. Suto *et*

[12]Schlosser *et al.* describes building and maintaining hypercubes in [76] in more detail while emphasizing building, broadcasting, and searching in this ontology.

al. proposed a system composed of super peers and interconnected leaf-nodes [103]. Liu *et al.* analyzed churn in tree and mesh-based peer-to-peer networks and concluded that a higher peer degree results in more resilience against churn [104].

The impact of churn depends on the type of peer-to-peer network. E.g. in a DHT where data is stored distributed amongst the peers churn can cause outages if not handled properly. In a pure DLT environment, where all the data should be available on every other node data loss is not of concern.

Churn is a relevant problem for DLT applications:

- Ethereum is known for a lot of churn [105] and Kiffer *et al.* describes churn as a problem for DLT with these words: "*Churn can run the risk of disconnecting a network or making it difficult to quickly propagate information throughout it, and challenging to estimate the size of the network*" [105]
- Churn in Bitcoin correlates between subnets [106]. Imtiaz *et al.* [106] also confirms that churn highly affects block propagation times.

2.7.10 Message Routing/Forwarding

As discussed in subsection 2.7.9 peers can be offline. Therefore fault resistant message routing can be a problem. This is less of an issue in DHT-networks since the hash-table can act as a letterbox[13].

Wei *et al.* describes the problem of message forwarding in a formal approach as follows: each node has bandwidth[14] and memory constraints, while each link that has a probability of reaching the target and messages requires memory. Each node communicates its available memory and bandwidth. Therefore the forwarding set can be considered as a multi-knapsack, problem where the sacks are bound by bandwidth and memory, with the optimizing goal of reaching the message. Multi-knapsack is considered NP-hard by Wei *et al.*, therefore we deduct that optimal message forwarding in a peer-to-peer network is also NP-hard. [108]

Approaches to message forwarding according to our conducted literature analysis:

- **DHT** based approaches (e.g.: [107])
- **Forwarding to peer closest to destination**. Esnault *et al.* used a "best tail selection al-

[13] this is a very simplified depiction of an approach of multiple authors, an example is [107]

[14] Wei *et al.* refers to it as "energy", but we deduced that bandwidth and computational was meant but since we consider computational power as non-issue we only refer to it as "bandwidth"

gorithm" to forward the message to the "best" available peer by ranking peers using bloom filters to weed out detours in the peer-to-peer network [109]. Wei *et al.* realized this idea with a more complex approach by acknowledging that peers have bandwidth and memory constraints, deriving the probability that the message reaches the target when sent over a specific link, therefore constructing multi-knapsacks to optimize message forwarding in a constraint environment [108]

- Chau *et al.* studied **random walks** for message forwarding, by tuning the weight of each node with the availability [13]
- Since a **broadcast** reaches every peer, we deduct that when one-to-all communication (broadcasting) is solved this implicitly solves one-to-one communication (with a lot of overhead).

2.7.11 Network Address Translation

Network Address Translation (NAT) is a technology widely adopted, especially in IPv4 networks. It is often coupled with Port Address Translation (PAT) that NAT became a colloquialism for NAT/PAT to such a degree that the english Port Address Translation Wikipedia page redirects to Network Address Translation at the time of writing.

NAT maps internal network addresses to external network addresses[15]. Since it became obvious that the IPv4 address space is running out [111] NAT became an option to only provide active clients dynamically with a public address for more efficient usage of the remaining addresses [112].

In the same way as NAT maps network addresses Port Address Translation (PAT) maps port addresses. By combining NAT and PAT it is possible to "masquerade" a whole network behind a single IPv4 address.

While every client on the inside can connect to any device on the outside, the other direction is heavily dependent on the specific form of implementation of the address translation. Most NAT/PAT implementations work the same way as a firewall by allowing sessions to be established from the inside to the outside but not the other way around.

This leads to problems in peer-to-peer networking since some peers can not be reached by others. Most solutions do not tackle this problem and rely on the user to create a "port forwarding" which is an option available in most NAT/PAT solutions. This option reserves an "external" network address and port combination to be translated to a specific internal network address and port combination. This requires user interaction and technical experience to some extent.

[15] internal and external are the terms used in [110]

Technical solutions without user interaction can be summarized to the umbrella term “NAT traversal”. Various NAT traversal methods are in use nowadays:

STUN : The UDP-based *Session Traversal Utilities for NAT* (STUN) [113], is a tool for handling NATs. It enables an endpoint to ascertain the IP address and port assigned by a NAT to match its private IP address and port. Additionally, it provides a method for an endpoint to sustain a NAT binding. STUN alone is not a solution for NAT traversal. Instead, it should be utilized as a tool within the framework of a STUN traversal solution. Along with certain extensions, the protocol is capable of performing connectivity checks between two endpoints, as well as transmitting packets between them. STUN is designed to be employed within one or more NAT traversal solutions, also referred to as STUN usages. Each usage details how STUN is implemented to achieve NAT traversal solutions. Typically, a use case specifies when STUN messages are sent, which optional attributes to include, which server to use, and which authentication mechanism to apply. Interactive Connectivity Establishment (ICE) represents one use case of STUN, while SIP Outbound represents another. In certain scenarios, a use case may demand extensions to STUN. These extensions may take the form of new methods, attributes, or error response codes. [114]

However, while STUN has proven effective for traversing non-symmetric NAT scenarios, it fails to address the traversal of progressive symmetric and random symmetric NAT, which require accurate prediction of open ports.[115]

TURN : The *Traversal Using Relays around NAT* (TURN) protocol enables peer-to-peer communication by allowing a connection to be established when one or both sides are unable to establish a direct peer-to-peer connection. Nevertheless, an automated method for discovering TURN servers does not exist. Consequently, clients need to be explicitly set up with recognized TURN servers. To enable TURN applications to function without manual configuration across different networks, an auto-discovery mechanism for TURN services is crucial. Three discovery methods maximize the chances of detection, depending on the network where the TURN client is located: Service Resolution, DNS Service Discovery, and Anycast. [116]

The protocol operates through a public relay server, enabling communication between devices enclosed behind NAT. TURN represents an advancement over the *Session Traversal Utilities for NAT* in that it is unrestricted by the category of NAT deployed. TURN accommodates every type of NAT, including symmetric NAT. Nonetheless, the fundamental downside to TURN is that locating the destination IP address and port of the peer can be challenging,

particularly when they are concealed behind symmetric NATs. [117]

ICE : Interactive Connectivity Establishment (ICE) is a protocol allowing NAT traversal for UDP-based communication. ICE employs STUN and TURN protocols for discovering network addresses and port mappings. STUN aids ICE in identifying the public IP address and port of the NAT, while TURN provides a relay service when direct communication is not feasible. By employing both STUN and TURN protocols, ICE is able to discern the most optimal communication route between two devices, guaranteeing a swift and dependable connection setup. ICE offers numerous benefits, such as compatibility with various types of NATs, straightforward integration, and versatility with multiple networks and protocols. [118]

The limitations include the increased complexity of implementations that require both STUN and TURN protocols, bandwidth limitations, increased latency, and server costs that scale with the number of users behind symmetrical NAT. [115]

Hole punching : Hole punching is a peer-to-peer communication technique that allows direct communication between hosts on different private networks in the presence of NAT. It is a straightforward technique for NAT traversal that enables peers to create communication sessions via both User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) protocols. The technique functions through exchanging public address and port information between the peers, then utilizing low TTL values[16] to "*punch holes*" in the NAT devices along the path. This effectively signals to the NATs that the communication sessions are solicited and should be accepted. [119]

The method is outlined and examined in the work "*Peer-to-Peer Communication Across Network Address Translators*" by Ford *et al.* The paper illuminates the pivotal elements, regarding both application and NAT behavior that enable hole punching to function. Ford *et al.* determines that roughly 82% of the NATs tested allow hole punching for UDP, and about 64% allow hole punching for TCP streams. [120]

[16] Time-to-live (TTL) referring to the maximum hop count of the IP packet. Therefore a low hop count ensures that the packet dies after traversing the NAT. This creates a situation where no traffic reaches the destination (which saves bandwidth) while the NAT translator considered the session to be open.

2.7.12 Bootstrap

In order to become a functioning part of a peer-to-peer network, a node needs to connect to the network. In some networks (e.g. Chord[17]) a node needs to position itself in the network to become a functioning part.

The term "join" refers to the process of becoming a part of a network, such as establishing connections and positioning in the network [14].

Some methods for bootstrapping are in common use: [121]

- List of known nodes in various facets
  - Config file
  - DNS
  - Known from previous sessions
- Out-of-band methods (like websites where known nodes are published over IRC-Channels)
- Utilize other overlay-networks (e.g. a DHT) as [122] and [123] also promote
- Scanning the internet (Gauthier Dickey *et al.* analyzed the actual feasibility of this approach [124])
- Passive listening for incoming connections

Knoll *et al.* uses Dyn-DNS for the bootstrap process and re-assigns the dynamic DNS entry if the current bootstrap node is unavailable [125]. Cirani *et al.* [126] proposes a multicast approach while acknowledging that multicast IPv4 is not widely supported on the public internet.

As Karbhari *et al.* highlights that in Gnutella peer info is collected by "ultra-peers" and published on the internet [127].

In the context of bootstrapping, we want to include some security considerations aswell: One of the most prevalent attacks on peer-to-peer networks is the Eclipse attack [32] and as Marcus *et al.* [8] has established: The bootstrapping overlay protocol is the most vulnerable part of modern peer-to-peer networks like Ethereum [8]. Therefore, a hardening and LangSec[18] approach reducing the attack surface is preferable. As Henningsen *et al.* demonstrated, clients listening and accepting *all* incoming connections can make the client fall victim to the *false friends attack* and become eclipsed [68].

[17] In Chord routing is done in a ring-structure [38]

[18] Referring to the "*Language-Theoretic Security*" approach where all possible inputs are defined as formal grammar and processed by a validated parser. See glossary entry LangSec for a definition of the abbreviation.

2.7.13 Processing Power

We assume that the readers who conform to the assumptions stated previously (see section 2.1) intrinsically understand what we mean when referring to processing power. For a more detailed explanation please refer to section 2.7.2, where various synonyms from the reviewed literature are enumerated. As stated we assume the reader is familiar with UDP, therefore we refer to Chen *et al.* when pointing out that the access time of erasure codes is dependent on the processing power [75].

As for processing power in consensus mechanisms in DLT, please visit the work of Kolb *et al.* [10]. As stated in [10], for permissioned blockchains requirements for participating in the consensus protocol can be more relaxed than for public blockchains. [10] further describes and classifies (processing) requirements of Proof of Work (PoW), Proof of Stake (PoS), Proof of Elapsed Time (PoET), Proof of Authority (PoA) or other Byzantine Fault-Tolerant (BFT) consensus protocols.

We summarize the means to reduce needed processing power:

Consensus : As commonly known[19] in DLT the choice about the consensus has an impact on the performance.

Super peers : Jafari Navimipour *et al.* [78] states that the topology of peer-to-peer networks influence their performance in handling queries. We believe that the topology would also affect DLT applications performance (information would reach the hosts creating the block faster). We categorize that as a push to hybrid networks with a super-peer structure.

Pure peer-to-peer : Jafari Navimipour *et al.* [78] also states that peer-to-peer networks are often used to eliminate bottlenecks a centralized service would experience. We categorize that as a push to fully distributed networks.

2.7.14 Attacks on peer-to-peer Networks

2.7.14.1 Sybil Attack

Based on [128], we summarize a Sybil Attack as an identity-based attack where an adversary can create many fake identities so that all connections of the victim will be established with the attacker. The threat posed by a Sybil attack to peer-to-peer networks is significant, as it involves the use of fake identities to deceive victims and enable double-spending. [128]

[19] and stated by Kolb *et al.* in [10]

This type of attack is especially alarming in completely decentralized systems, where there is no central authority controlling the number of node IDs an attacker can obtain [42]. Thus, attackers can take control of a significant portion of the nodes in the network by joining with multiple node IDs [42]. This vulnerability poses a challenge to preventing Sybil attacks entirely, as shown by Douceur. However, effective methods exist to impede it. [129]

These include honest nodes that only forward accurate information to the genuine network while disregarding misleading data propagated by attackers. The Bitcoin protocol implements a range of measures to improve Sybil resistance. Nevertheless, the effectiveness of these measures cannot be entirely guaranteed, as they rely heavily on maintaining a functional connection to the network, which aims to reduce associated risks. It is important to differentiate between the Sybil attack and the Eclipse attack, where the latter only focuses on compromising identity without affecting the network as a whole.
In addition, executing a Sybil attack necessitates that the attacker generates or controls enough Sybil nodes to significantly influence the choices made by honest nodes. [128]
As summarized in [42] and [129], a Sybil attack in pure peer-to-peer networks can only be impeded by restricting the rate at which IDs can be generated.

2.7.14.2 Eclipse Attack

The Eclipse attack is a sophisticated and targeted attack on peer-to-peer networks that aims to isolate specific nodes or groups of nodes from the wider network by monopolizing connections to a victim node and controlling the flow of information. The attacker manipulates the routing tables of the targeted nodes or hijacks their IP addresses, exploiting the peer discovery logic used by the network, which is susceptible to "*false friends*" attacks. [68] [3]
These attacks permit the blocking of long-term, remote nodes controlled by victims by silently introducing antagonistic nodes with deliberately chosen IDs. The attacker requires only two hosts located on different /24 subnets to trigger an eclipse, and just one Sybil node in each bucket of the neighbor table to guarantee that solely antagonistic nodes are sent back to the peer management, thus altering the victim's perception of the network and the blockchain. [68] Once the nodes have been isolated, the attacker is able to launch multiple subsequent attacks, for example double-spending, transaction censorship or stubborn mining. [68] [3]
These manipulations pose a significant threat to the security and reliability of peer-to-peer networks.

To reduce the potentially catastrophic outcomes of Eclipse attacks, various countermeasures have been proposed. These countermeasures include incorporating resilient routing protocols into the network and using monitoring tools to identify and prevent such attacks. Additionally, the adoption of reputation-based systems could locate and isolate malevolent nodes, thus ensuring the network's reliability and operation. [3] Notably, even after implementing security upgrades, platforms such as Ethereum's Go Ethereum (Geth) are still susceptible to Eclipse attacks. [68]

2.7.14.3 Double Spending

A significant reliability and security issue in the area of digital currencies, including decentralized networks using PoW consensus mechanisms such as Bitcoin is double-spending. This occurs when a malicious user spends a unit of digital currency more than once, by orchestrating two conflicting transactions using the same input. This problem arises due to the fact that the blockchain is maintained by a decentralized network of nodes. Transactions are not added immediately to the chain, instead, they are initially broadcasted to the network and added to an unconfirmed transaction pool. [3] [21]

In a standard scenario, an attacker - who may be a customer in the Bitcoin network - initiates one transaction for goods or services to a merchant and almost immediately orchestrates another to a different node to be added to the blockchain. This intruder then competes to have their fraudulent transaction recorded on the blockchain faster than the legitimate transaction, thereby obtaining goods or services without truly spending the digital currency unit. [21]

In order to prevent double spending, Bitcoin has a built-in mechanism where transactions are recorded in a unique ledger that is replicated by each Bitcoin node. Once a transaction has been confirmed and integrated into blocks, it is protected from double-spending through digital signing and agreement facilitated by distributed timestamp resources. Bitcoin's PoW-based timestamp mechanism helps to mitigate the risk of attacks, taking around ten minutes to validate a rapid transaction. During this time period, there exists a vulnerability where attackers can advance a longer blockchain. This enables them to perform duplicate spending on fast payments before necessary verifications are completed, leading to the issuance of goods and services without requisite confirmations, which in turn leads to double-spending attacks. These may become more complicated, necessitating the engagement of multiple nodes in the network. Detecting double-spending within the Bitcoin network can be achieved through a detailed analysis of the network's infrastructure, which facilitates the creation of effective techniques. [3]

2.7.14.4 Churn Attack

Churn attacks are a frequent cause of denial-of-service attacks on peer-to-peer networks. These attacks result from the continual and rapid arrival and failure of a significant number of participants in the system. Such attacks can cause extremely stressful network conditions, increasing host loads, and blocking a significant fraction of normal insert and lookup operations in the peer-to-peer system. Because churn attacks can be instigated by sources typically considered non-malicious, they are considered a threat. Attackers may incite churn in order to launch a distributed denial-of-service attack, which could potentially disable the sharing mechanism and cause participating machines to experience high levels of traffic. [130]

2.8 Established Networks

2.8.1 Gnutella

2.8.1.1 Overview to Gnutella

Gnutella, established in the early 2000s, is recognized as a decentralized, unstructured peer-to-peer network used predominantly for file sharing purposes [77]. The network's operational model begins with client connections to public peers, as listed by a central server. These clients then employ "*ping*" messages to further expand their network reach by discovering additional nodes [38]. Central to Gnutella's functionality is its search mechanism, a flood search, which entails the broadcasting of file requests to directly connected neighbors. This method, while extensive in reach, often leads to increased network traffic and redundant requests, resulting in a heavier network load [131]. To mitigate these issues, strategies like Time-to-live (TTL) and expanding ring searches are employed, particularly effective for files that are frequently shared. However, these strategies exhibit limitations in managing less common files and fail to fully resolve the problem of duplicate messages [46]. The network's structure, characterized by its random node connections, enhances its resilience. Nonetheless, this structure presents challenges in efficient information dissemination, necessitating the exploration of more effective search mechanisms [9]. Further analysis in the context of Gnutella's broadcasting operations reveals the influence of network latencies on the efficiency of flood broadcasts, particularly highlighting the "short-circuiting" phenomenon. This phenomenon indicates a failure of messages to reach extensive segments of the network, thus raising concerns regarding the scalability and effectiveness of current peer-to-peer broadcasting

techniques within such networks [132].

Despite these limitations, Gnutella has facilitated the creation of extensive networks. [46]

2.8.1.2 Bootstrap

To connect to the Gnutella network, peers need to undergo a bootstrap process and require the IP addresses of online peers. The GWebCache system serves as a decentralized repository for this information and permits peers to question caches for a roster of online peers. In Gnutella, all nodes act as clients and servers and are referred to as servent. In subsequent executions, servent implementations experiment with other techniques, such as keeping local registries of hosts from prior runs. The implementation of the bootstrapping process varies significantly over different servent implementations. These differences lead to substantial differences in performance for the users. The GWebCache system plays a crucial role in comprehending bootstrapping algorithms. Comprised of voluntary-operated caches, it maintains a list of peers who accept new incoming connections. [127]

The GWebCache protocol entails updating the cache with IP addresses and port numbers. While clients transmit messages to the web server via HTTP requests, GWebCaches do not possess online host information beyond their IPs and ports. The information contained in these caches is supposed to be updated by the servents as the Ultrapeers(Gnutella) or Superpeers (Gnutella2) do not communicate with each other at any time. [127]

2.8.1.3 Security

Bellovin [133] identified various security concerns within Gnutella and addressed them in his research. An instance of this was the possible exploitation of the “push” function to enable DDoS attacks.

Wallach [5] examines security concerns related to peer-to-peer networks, such as Distributed Denial of Service (DDoS) attacks and address spoofing. Due to the absence of a central authority to certify node identities, it is challenging to determine who can be trusted. The article proposes solutions that involve the use of micro-cash to foster reputation building. Furthermore, the text addresses problems with Napster and Gnutella, such as privacy breaches and the challenge of restricting access to these networks via firewalls.

2.8.1.4 Gnutella2

Several improvements that were made during the development of this project led to the creation of the follow-up project Gnutella2 [30]. One significant change in the newer version is its topology, which is not fully decentralized, as it includes super-peers that act as hubs and bundle connections from various clients, and they maintain many connections to other super-peers. Furthermore, these super-peers store search indexes for all clients that are directly connected to them. This change enables a new method of file searching that eliminates the need for flooding the network. Instead, a client sends a search query to a super-peer, which consults its index and then forwards the request to the appropriate client or another super-peer. If the search is unsuccessful, the request will be forwarded to additional super-peers. The primary distinction in this method lies in the sequential handling of search requests to various peers with Gnutella2, in contrast to the original Gnutella's approach of flooding the network from the outset. [30]

2.8.2 Napster

Napster, a centralized peer-to-peer file-sharing network, relied on a central server that hosted a "*centralized global index*" for users to share MP3 files. This server housed an extensive database of metadata and other pertinent information about shared files and online peers [70].
The actual acquisition of files, or parts of files, was decentralized by downloading directly from peers. [67]

The peers, upon joining, would submit metadata about the files they wanted to share to this central entity, which would then add it to its central index, thus acting as a broker between the peers [70]. Although the system made resource discovery fast and reliable, it had a glaring weakness - a significant scalability limitation and a potential single point of failure, increasing the risk of malicious attacks and privacy breaches [125].
Due to Napster's combination of decentralized content and centralized index, it only partially fulfilled the requirements for distributed control of peer-to-peer systems [134].

2.8.3 Kademlia

2.8.3.1 Overview to Kademlia

Kademlia [35], a DHT algorithm, plays a pivotal role in decentralized data storage and retrieval within computer networks. Its widespread application in p2p networks and DApps, including BitTorrent[20] and DLT systems, highlights its significance in modern network architectures (refer to subsection 2.5.2 and [41]). The design of Kademlia, as introduced by Maymounkov *et al.*, emphasizes scalability and fault tolerance, eliminating the need for centralized control or indexing in expansive networks [35].

Kademlia stands out as a meticulously crafted DHT protocol, incorporating XOR metrics, routing tables, and redundancy strategies to establish a robust framework for data distribution and retrieval in peer-to-peer environments [135]. Its implementation extends across various p2p applications, ranging from file sharing and decentralized cryptocurrencies to distributed storage systems, demonstrating its versatility and effectiveness in diverse network settings [35].

2.8.3.2 Structured Scalability

In the field of large-scale distributed systems, Kademlia's structured scalability is a crucial feature. This scalability is achieved through the organization of nodes and data within a structured binary-tree network, a design that is fundamental to managing the expansion of the network in terms of nodes and data [136]. The nodes in this network are assigned unique Node IDs, typically in the form of large numbers or hashes, which play a crucial role in the system's scalability [35].

Kademlia's use of DHTs and the XOR metric is instrumental in organizing the network into this binary tree structure [136]. This organization is not merely for structural integrity but also for optimizing the routing process. Specifically, Kademlia maintains a logarithmic count of routing hops, a feature that becomes increasingly beneficial as the network scales [35].

This logarithmic scaling is essential for handling a large number of nodes, ensuring that the computational and communication costs associated with queries remain manageable even in extensive networks [136].

Moreover, when a node initiates a query for data or another node, Kademlia's architecture allows it to quickly narrow down the search to a limited subset of nodes. These nodes are selected based on their Node IDs, which are closest to the target of the query, thereby significantly reducing

[20] The extend of BitTorrent's Kademlia usage is described in [41]

the computational and communication overhead [35]. This efficiency in data retrieval and node querying is a testament to Kademlia's structured scalability, making it a robust solution for large-scale distributed systems [136].

2.8.3.3 Decentralization and Autonomy

The second fundamental characteristic of Kademlia, as highlighted by Maymounkov *et al.*, is its strong emphasis on decentralization [35]. This decentralization means the absence of any central authority or control within the network, with all nodes operating on an equal footing and collectively contributing to the network's functionality. In Kademlia's framework, there is no central server or index to oversee data storage or node management. Instead, each node independently determines the data it stores and its role in routing queries, enhancing the network's robustness, resilience, and resistance to censorship [135]. This decentralized nature is in line with the principles of peer-to-peer networks, where nodes work collaboratively without depending on a singular, potentially vulnerable point of authority [35].

Moreover, Kademlia's architecture, which organizes nodes and data into a structured binary-tree network, supports its scalability and efficient management of an increasing number of nodes and data as the network grows. This structure, where nodes are assigned unique Node IDs represented as large numbers or hashes, allows Kademlia to maintain a logarithmic count of routing hops even in extensive networks [137]. Such a system enables a node to quickly narrow down its search to a small subset of nodes, closest in Node IDs to the target, thereby significantly reducing the computational and communication costs of queries. This aspect of Kademlia is crucial for handling large numbers of nodes while ensuring efficient data retrieval [35].

2.8.3.4 Fault Tolerance and Adaptability

According to Maymounkov *et al.*, Kademlia's third key characteristic is fault tolerance. Fault tolerance refers to a system's ability to continue operating correctly despite the presence of failures, errors, or network disruptions [35]. This fault tolerance in Kademlia is multifaceted, with one crucial aspect being its iterative and parallel lookup processes. When performing a lookup, hosts contact peers with progressively smaller eXclusive OR (XOR) distances (XOR-based routing is explained in the next subsection [2.8.3.5]) to the target ID through prefix matching, which not only facilitates efficient data location but also enables dynamic updating of routing information. Nodes in Kademlia

automatically learn and update their routing tables based on received queries, making this update mechanism an automatic outcome of regular interactions within the network [138].
To enhance network reliability, Kademlia nodes periodically refresh their routing tables by sending ”ping” messages to other nodes, ensuring up-to-date information about the network’s topology. Additionally, Kademlia’s design incorporates data redundancy by storing data across multiple nodes, thereby reducing the risk of data loss due to node failures. The re-publishing requirement for stored data, particularly in file-sharing contexts, addresses the issue of stale data [35].
Moreover, Kademlia employs data replication to enhance system resilience. This involves storing key-value pairs corresponding to resources at multiple nodes, particularly those whose IDs are closest to the resource’s key in the identifier space. Such redundancy ensures continuous access to critical data, even in the event of node failures, thus preventing data loss and ensuring uninterrupted operation [138]. These features, including efficient and self-updating routing protocols, establish Kademlia as a resilient system within the DHT-based algorithm landscape, capable of adapting to changes and failures in the network [35] [138].

2.8.3.5 Efficient XOR-Based Routing

Kademlia’s fourth critical attribute, as outlined by Maymounkov *et al.*, is its adeptness in routing, primarily utilizing the XOR metric [35]. Routing, in this context, is the method of determining the optimal path from a source node to a destination node within the network. Kademlia employs the XOR distance metric to ascertain the “closeness” of Node IDs, where nodes with smaller XOR distances are considered closer. This approach ensures that when a node seeks to find data or another node, it prioritizes nodes that are nearest in terms of XOR distance to the intended target [14]. Consequently, this strategy directs the request through nodes that are progressively closer to the destination, effectively reducing the number of hops and thereby lessening both computational and communication burdens. This proficient routing mechanism is fundamental to Kademlia’s ability to rapidly locate data within a decentralized network [35].

2.8.3.6 Data Structure

The Routing Table is a crucial data structure at the core of Kademlia’s network. It simplifies the process of discovering nodes and retrieving data by organizing nodes into distinct ranges based on their Node IDs. This table consists of a series of ’buckets’, each representing a unique range of

Node IDs . Nodes in the Kademlia network are identified by Node IDs, which are usually hashes or large numbers. These IDs are arranged in a binary tree structure, where each level corresponds to a bit position in the Node ID, creating a hierarchical organization and defining node proximity [135]. Each bucket within the Routing Table is aligned with a specific range of Node IDs, mirroring a level in the binary tree. The initial bucket encompasses the entire key space, ranging from 0 to 2^{b-1}, where b represents the bit count in the Node ID. The subsequent buckets cover progressively smaller areas, with each bucket covering half the area of its predecessor. The total number of buckets, determined by the number of bits in the Node ID, determines the depth of this tree-like structure [35].

Nodes within each bucket are cataloged based on their Node ID range and ordered by the latest contact. The nodes are ordered based on the recency of contact, with those that were contacted less recently placed at the beginning [14]. These nodes are not only identifiers but also contain important network information such as IP addresses and ports, as well as metadata that improves routing efficiency [135]

Kademlia's design enables the dynamic adjustment of the Routing Table by splitting or merging buckets. When a bucket reaches its capacity limit, it splits into two, each covering a more specific Node ID range. Conversely, underpopulated buckets can merge with adjacent ones to span a broader range. The selection of the appropriate bucket for a Node ID is guided by the XOR metric, which calculates "closeness" by performing a bitwise XOR between Node IDs. The bucket with the smallest XOR distance to a target Node ID is chosen, thereby enabling efficient routing to nodes that are "closer" in the key space [35].

2.8.4 InterPlanetary File System (IPFS)

The IPFS was designed to future-proof the internet by storing (web-)resources in a distributed Directed Acyclic Graph (DAG) and sharing them on demand via a peer-to-peer network [46]. By caching resources highly distributed the network becomes a Content Delivery Network (CDN) where the required resource can be fetched from the closest peer. Therefore, if humanity colonizes another planet the internet of today would become terribly slow without the usage of IPFS.

IPFS on a technical level uses: [46]

- S/Kademlia to identify nodes and authenticating messages ([4] could not verify this!)
- A DHT for resolving node names to network addresses and querying peers for a specific object

- BitSwap as data exchange protocol
- A Merkle DAG for linking related blocks
- ICE NAT traversal

Henningsen *et al.* [4] crawled the DHT of IPFS in 2020 and shed some light on the actual specifics in use in contrast to the whitepaper ([46]). Key findings of [4] include the absence of S/Kademlia and that a huge proportion of nodes are not reachable (which questions the usage of ICE NAT traversal as proposed by the whitepaper).

2.8.4.1 BitSwap

BitSwap is a protocol proposed with IPFS [46]. The gist behind BitSwap is that nodes advertise the block they have and which they want and if a node offers a needed block they "swap". These transactions are recorded in a ledger. If a node is too much "in debt" others stop sharing blocks and the node starts helping its peers to find the blocks they want in order to repay the favor. A more in-depth analysis of the protocol was done by De la Rocha *et al.* [24].
Our perception of a DLT is more focused on broadcasting blocks to all nodes in the network. Therefore the "on-demand" approach of BitSwap makes it not very suited for our use case.

2.8.4.2 Lib peer-to-peer (libp2p)

2.8.4.2.1 Overview : Originating from the BitTorrent DHT implementation, libp2p has become a widely used library supporting peer-to-peer networking in many projects [40]. Such projects include the well-known InterPlanetary File System (IPFS) and the Ethereum network [139]. The latter especially in the beacon chain after the merge upgrade. libp2p has been widely adopted by the Ethereum community, as evidenced by its compatibility with Ethereum 2's networking requirements. The integration of libp2p into the Ethereum mainnet, led by the collaboration between Protocol Labs and the Ethereum community, represents a major achievement in this field. [140]

2.8.4.2.2 GossipSub : A key protocol in the libp2p network stack, the GossipSub protocol, is an extensible publish/subscribe protocol designed to efficiently and securely distribute pubsub messages over the libp2p network [141]. The protocol benefits largely from libp2p's functionalities, which include secure connections of peers, traversal of NATs, support for several transport protocols, and peer discovery and routing. It notably utilizes GossipSub for an effective publish/subscribe mechanism, allowing a sender to transmit messages to various recipients without requiring

knowledge of their identities. This promotes scalable and efficient message distribution in P2P networks [142]. Moreover, the libp2p network houses GossipSub implementations in several programming languages, improving interoperability across diverse platforms and providing developers with greater flexibility in their applications [139].

2.8.4.2.3 Routing protocols : On the technical side, libp2p presents three routing protocols for content routing: Multicast DNS (mDNS), Kademlia DHT (KAD), and Publish-Subscribe (pubsub). These protocols provide functionalities for peer routing and discovery. mDNS broadcasts queries to find peers with specific content, while KAD uses the DHT to locate peers containing the required content. The publish-subscribe protocol is implemented in two ways. FloodSub and GossipSub are both utilized in libp2p [143].

While libp2p progresses towards enhanced functionality, security, and performance, it is important to recognize possible vulnerabilities. Vigilant resource management is necessary to prevent potential attacks, such as resource exhaustion and the exploitation of integer overflow vulnerabilities, and to ensure a secure and robust network infrastructure [139].

2.8.5 Bitcoin

Bitcoin [18] was the first popular cryptocurrency but was not the first blockchain technology. The first blockchain technology was published by Chaum [144] in 1982 according to Sherman *et al.* [145].

Decker *et al.* provided a very detailed overview of the technical specifics of bitcoins network layer [53]. Drawbacks of Bitcoins design are the slow block generation rate[21], the huge power consumption of the PoW-algorithm [66][146] and various attack vectors [3].

We acknowledge Bitcoin as a needed step towards distributed ledger technologies, but nowadays bitcoin is more like an object of financial speculation rather than a usable fiat currency or a high-performance distributed system.

[21] At time of writing at approximately 10 minutes per block; Time taken from `https://bitinfocharts.com/comparison/bitcoin-confirmationtime.html`

2.8.6 Ethereum

2.8.6.1 Overview

Ethereum [147], created by Vitalik Buterin in 2013 and launched in 2015, is a decentralized and open-source blockchain platform. It provides a secure and transparent environment for executing and verifying smart contracts. The platform has expanded beyond financial transactions to serve various industries, such as finance (Decentralized Finance (DeFi)), supply chain management, and gaming [148]. Ethereum's rise to become the second-largest cryptocurrency by market capitalization underscores its substantial impact in the digital domain. This is especially noticeable in its flourishing ecosystem for decentralized applications (DApp) and DeFi, which are enabled by its Turing-complete programming language, showcasing the platform's technical sophistication and flexibility [43].

2.8.6.2 Core Architecture

Ethereum's operations are based on an intricate architecture that combines a network of interconnected nodes. These nodes communicate and synchronize through the Ethereum Virtual Machine (EVM), enabling the implementation of smart contracts. [71]
Ethereum's core architecture demonstrates its innovative approach to blockchain technology. The EVM is at the core of Ethereum's architecture, acting as the execution engine for the platform and enabling the implementation of smart contracts and DApps.[71]. This intricate network of interconnected nodes, which communicate and synchronize through the EVM, ensures a secure and transparent environment for executing and verifying smart contracts. Furthermore, the platform's peer-to-peer network, utilizing protocols like RLPx and DEVp2p, manages node discovery and communication. This diverse node structure significantly impacts the network's efficiency and block propagation, presenting both challenges and opportunities for optimization in Ethereum's network infrastructure [43].

2.8.6.3 Smart Contracts

Smart contracts are programs embedded in the Ethereum blockchain that execute automatically when predefined conditions are met. They enable secure and transparent interactions and transactions among network participants. The smart contracts are executed by a decentralized network

of nodes, ensuring that the results are immutable, verifiable, and securely distributed across the network. [148].

Ethereum's native cryptocurrency, Ether (ETH), is essential to this ecosystem as it facilitates transactions and incentivizes network participants [43]. Solidity, Ethereum's programming language, enhances the platform's capabilities by enabling the execution of complex contracts and applications directly on the blockchain. This has several advantages, including decentralization, immutability, and resistance to fraud, which enhances the security and reliability of the Ethereum network [147].

2.8.6.4 Developmental Potential

In addition to its native cryptocurrency, Ether (ETH) [43], Ethereum offers a platform for the development of dezentralized application (DApp)s utilizing its Solidity programming language [43]. This language enables developers to create custom smart contracts and advanced applications, utilizing the decentralized computing capabilities of the Ethereum network [148]. The platform's programming language is Turing-complete, providing flexibility in development. This facilitates the creation of DApps and enables the design of custom smart contracts and advanced applications that exploit the decentralized nature of the network [43].

Ethereum's impact on the digital realm is highlighted by its position as the second-largest cryptocurrency by market capitalization [148]. The platform's success in fostering a thriving ecosystem for DApps and DeFi demonstrates its developmental potential. Ethereum's potential to transform industry operations and interactions within a decentralized, blockchain-oriented framework is underscored by its versatility in accommodating a diverse range of applications [148].

2.8.6.5 ETH 2.0

2.8.6.5.1 Conensus Layer The system referred to as Ethereum 2.0 (ETH 2.0), denominated as the "*consensus layer*", aims to address the challenges related to scalability, security, and energy efficiency within the Ethereum network [146]. ETH 2.0 incorporates various significant modifications and features intended to enhance the overall performance and usability of the platform.

As outlined by the ethereum.org-team, the previously used terms Ethereum mainnet (ETH 1.0) and ETH 2.0 have been superseded in favor of labeling "ETH 1.0" as the "execution layer" and "ETH 2.0" as the "consensus layer". Collectively, these components are collectively referred to as Ethereum [149]. Despite the term "consensus layer" carrying inherent significance within the

(DLT) context, we maintain the usage of ETH 1.0 to denote the state preceding “the merge”, and ETH 2.0 to denote the state after the transition’s completion. This decision is motivated by our belief that the term “consensus layer” adequately describes the shift from PoW to PoS, but falls short in comprehensively characterizing the transition to libp2p.

2.8.6.5.2 The “Merge” The term “*the merge*” is often used to refer to the transition from the (ETH 1.0) to ETH 2.0, where the ETH 1.0 chain became a shard within the ETH 2.0 network. [31] The Merge, executed on September 15, 2022, was the joining of Ethereum’s original execution layer with its new Proof of Stake consensus layer, the Beacon Chain. Its implementation eliminated the energy-intensive mining mechanism and facilitated the network’s safeguarding through staking [140].

2.8.6.5.3 Proof of Work (PoW) to Proof of Stake (PoS) The transition from a PoW consensus mechanism to a PoS consensus mechanism is the most significant change in ETH 2.0 [140]. In the ETH 1.0 network, miners compete to solve complex mathematical problems to validate transactions and create new blocks. This process requires a significant amount of computational power and energy consumption. ETH 2.0 relies on validators who hold and lock a specific amount of ether (ETH) as a stake in the network to establish consensus. Validators are randomly selected to propose and validate new blocks based on their ETH holdings and reputation scores. Additionally, ETH 2.0 implements shard chains, which can process their transactions and smart contracts. This implies that the network can be scaled up and its capacity can be increased significantly, allowing for a greater number of transactions to be executed concurrently. [31]

2.8.6.5.4 Sustainability An analysis comparing ETH 2.0 to the preceding Ethereum network, reveals significant advancements in terms of scalability and energy efficiency [150]. The transition to a PoS consensus mechanism and incorporation of shard chains means ETH 2.0 is capable of processing a higher number of transactions simultaneously whilst reducing energy consumption when compared to the resource-intensive PoW mechanism used in ETH 1.0. [31]

2.8.6.5.5 Network Layer The Ethereum network relies on various components to facilitate decentralized communication and consensus [43].
Kim *et al.* describes the networking layer of the original Ethereum network in great detail including RLPx as a Kademlia-based node discovery layer, DEVp2p as a session layer, followed by the “eth”

subprotocol as indicated through DEVp2p. Kim *et al.* also shows visually how many nodes operate on the different sub-networks in Ethereum's peer-to-peer network. [43]
Starting with ETH 2.0 libp2p is "merged" into the networking layer [140].

2.8.6.5.6 Security Security in Ethereum, initially like Bitcoin, was maintained using PoW, which provided robust protection as long as the majority of participants were honest. The security of PoW is predominantly dependent on the substantial computing power required to generate a block [151]. However, with the evolution of hardware and rising concerns over sustainability and throughput, there has been a shift in the blockchain community towards alternatives to PoW. As a result, Ethereum's security mechanism transitioned from PoW to PoS following the merge [10].
This change, while reducing the energy consumption associated with PoW, brought about new security concerns in PoS, particularly its vulnerability to monopolies or centralization and a heightened risk of attacks and fork issues due to the lower cost of mining blocks. A significant challenge within the PoS framework is the 'nothing at stake' problem, where validators might be incentivized to support multiple blockchain histories, potentially compromising network security [152]. The adoption of PoS in Ethereum represents a significant step in addressing the limitations of PoW, such as high energy consumption and vulnerability to forks and selfish mining, but it also introduces unique challenges that necessitate careful consideration in maintaining the security and integrity of the network [10].

2.8.6.5.7 Summary Ethereum has become a pivotal decentralized and open-source blockchain platform. Its core architecture, including the EVM, supports immutable and verifiable smart contracts, contributing to its growth in various sectors, such as DeFi [148][43].
"The merge", marking Ethereum's transition from ETH 1.0 to ETH 2.0, shifted its consensus mechanism from PoW to PoS. This transition aimed to improve energy efficiency and scalability but introduced new security challenges, including the 'nothing at stake' problem and potential risks of centralization [140][152]. ETH 2.0 also implements shard chains, enhancing transaction processing capabilities [31].
Overall, Ethereum's evolution into ETH 2.0 signifies major strides in blockchain technology, balancing scalability, security, and energy efficiency, while also presenting new challenges in maintaining a secure and decentralized network [150][10].

3 Scenario

3.1 Overview

The scenario this thesis tackles is the theoretical creation of a non-public distributed ledger application with user interaction. Our goal is to design a peer-to-peer network capable of powering this distributed ledger application. In contrast to other peer-to-peer networks participants in our network are authenticated. In our scenario, the distributed ledger technology is not used for financial transactions, instead, it should be capable of performing distributed computing tasks. We assume that these computing tasks are not intended as background jobs and moreover, we assume these computing tasks are interactive programs like the countless established applications in use on non-distributed systems.

3.2 Problem Identification

The primary problem this thesis addresses is the lack of available guidance on implementing peer-to-peer networks for permissioned DLT applications. Despite the potential of DLT for decentralized data management and transaction processing, as outlined in chapter 1, it faces significant challenges. These include vulnerability to network attacks (chapter 2.7.14), scalability issues as the network grows (chapter 2.7), and maintaining data integrity and consensus across a distributed network. Additionally, the integration of various technologies poses compatibility and interoperability challenges, as highlighted in chapter 2.8.

We perceive a lack of research about non-public networks. We induce that this is because public networks are easier to study and that public networks due to their excess amount of peers are more relevant. We see this lack of research and therefore software frameworks as a problem.

3.3 Interaction with Environment

Central to this discussion is how the network responds to and is influenced by external factors, such as network congestion, variable connectivity, and potential security threats, as discussed in chapter 2.7.14. The scalability of the network, a key aspect analyzed later in chapter 6, is particularly challenged by these environmental factors. Additionally, we examine the network's adaptability to changing technological landscapes, referencing insights from chapter 2.8. With the changing technological landscape, we refer to longtime tendencies such as the adoption of the QUIC protocol as HTTP/3 [153] or the ever-increasing amount of internet-connected devices [154].

3.4 Theoretical Constraints and Considerations

Central to the considerations are the inherent limitations of current technologies, as explored in chapter 6. This includes issues related to network bandwidth, latency, and the computational overhead associated with maintaining a distributed ledger. Available bandwidth and latency are a constraint no peer-to-peer application can tackle since they are part of the unchangeable working environment for software.

We also acknowledge that we do not have the computational resources to simulate networks on the scale of Ethereum and therefore can only base our implementation recommendations on argumentation.

4 Approach

As outlined in chapter 3 the goal of this thesis is the creation of a theoretical framework for a peer-to-peer network of a permissioned distributed ledger technology.
To create such a state-of-the-art peer-to-peer network we derive criteria from the presented scenario to evaluate possible solutions for their alignment with the scenario. To advance current peer-to-peer solutions we analyze potential solutions to open problems and challenges. These identified solutions are then evaluated using the defined criteria. Identified potential solutions to the scenario are discussed and evaluated. Based on this evaluation and knowledge from existing peer-to-peer networks the specifics of a new peer-to-peer network are formed.
We decided to use literature analysis to discover problems and challenges in peer-to-peer networks. We base this decision on the fact that peer-to-peer networks and blockchain, in general, are not novel technologies, therefore literature exists. We balanced a systematic literature review with the search for already existing literature surveys and concluded that looking for existing literature surveys that cover problems and challenges of peer-to-peer in general as well as problems, challenges, and attacks on distributed ledger technologies would yield the same results.
We decided to use literature analysis again to look for possible solutions. We base this decision on the fact that we perceive a great diversity of already existing approaches and therefore conclude that approaches from different already existing solutions can be put together to solve problems.
Potential solutions as well as problems and challenges in peer-to-peer networks are already established in section 2.7 of this thesis.
Readers already have a rough understanding of the success factors since the scenario has already been outlined in chapter 3.

5 Existing Frameworks

5.1 Hyperledger Fabric

5.1.1 Overview

Hyperledger is a community that writes frameworks for building blockchain solutions. One of these frameworks is the Hyperledger Fabric, which allows users to execute arbitrary code as "smart contracts" to build private blockchains. [25]. As highlighted by Brotsis *et al.* Hyperledger Fabric lacks post-quantum security, smart contracts can be hazardous, and the "Membership Service Providers"[1] can develop into a Single Point of Failure [2].

We ranked Hyperledger Fabric as relevant because of its modular design, customizability, intended for use in private/permissioned ledgers, and open source while backed by a Foundation that is supported by well-known members[2] and performs audits and penetrationtests[3]. Considering the long legacy [4] we perceived the technology as stable for production use.

Brotsis *et al.* [2] addresses several issues with the Hyperledger Fabric framework, including vulnerabilities in smart contracts and the specification, network security, and privacy issues. In particular, it highlights the fact that few publications cover the architectural risks associated with the security and privacy of Hyperledger fabric. In addition, the document discusses the vulnerabilities in the Hyperledger fabric blockchain implementation that can lead to inefficiencies and potential security risks. The analysis of the Membership Service Providers (MSP) indicates that the appropriate safeguards are not provided to mitigate the network risks associated with fabric, potentially resulting in a single point of failure. Fabric's smart contracts were designed to be written in high-level

[1] the Membership Service Providers is a term used by Hyperledger Fabric to describe a certificate issuing instance

[2] at the state of writing Hyperledger claims to have 135 members with total funding of 11B USD,

see `https://dltlandscape.hyperledger.org/members?grouping=category`

[3] Most recent pentest listed on the website at the point of writing is from 2021[155]

[4] at the point of writing the GitHub repository lists releases back until 2017,

`https://github.com/hyperledger/fabric/releases?page=8`

languages, like Java, to reduce the learning burden on developers. [2]

5.1.2 Considerations regarding the scenario

We consider Hyperledger Fabric as a relevant work for multiple reasons. One of the reasons is that the goals of the Fabric project align with ours.
An analytical review, as demonstrated by Brotsis *et al.*, illustrates notable weaknesses inherent in the Hyperledger Fabric architecture, significantly compromising its suitability for this venture. Particularly, the identified shortcomings in network security and privacy mechanisms could potentially lead to worsened latency issues, which goes against our aim of reducing latency. [2]
Furthermore, vulnerabilities in the Fabric's smart contracts as described in "On the Security and Privacy of Hyperledger Fabric: Challenges and Open Issues" [2], not only pose potential security risks but also suggest a likely single point of failure. This aspect contrasts sharply with our goal of promoting resilience by eradicating as many points of failure as possible, favoring a decentralized approach that improves reliability and decreases the risk of failure. The identified shortcomings in the MSP aggravate these concerns as it lacks essential precautions to efficiently alleviate network hazards linked to Fabric.
This thesis emphasizes a progressive viewpoint, concentrating on solutions capable of embracing future developments, such as IPv6 and UDP support, which are not currently adequately fortified in the Hyperledger Fabric framework.
Given the significant differences between the goals stated in this thesis and the constraints inherent in Hyperledger Fabric, as identified by [2], it is reasonable to conclude that this framework may not be a suitable basis for our prototype DLT.

5.2 libp2p

5.2.1 Overview

Originating from the BitTorrent DHT, libp2p has evolved into a significant library facilitating peer-to-peer networking in several high-profile projects, including IPFS and the Ethereum network, as it is integrated in Ethereum 2.
A crucial component of the libp2p network stack is the Gossipsub protocol. This adaptable publish/subscribe protocol enables developers to create flexible applications with secure and efficient

message distribution across various platforms. Notwithstanding the ongoing improvements to enhance functionality, security and performance, it is important to acknowledge that libp2p being an open-source project is continuously developing to address potential vulnerabilities.

5.2.2 Considerations regarding the scenario

Upon reviewing “libp2p” we find certain potential limitations that could impact its suitability for our specific requirements.

Firstly, despite the advancements and its widespread usage, as noted by [139], the libp2p project is still under continuous development to address potential vulnerabilities and enhance its functionality, security and performance. This ongoing developmental cycle, characteristic of open-source projects, might pose inherent risks of instability or incompatibility.

Secondly, we emphasize future-proof software, which necessitates a stable build environment and stable API. The current stage of libp2p being in continuous development could potentially entail complexities when libp2p implements changes. For example, the adaptable Gossipsub protocol, a significant component of libp2p, though flexible, does not appear to support hardware tokens or to change the cryptographic primitives as needed.

Finally, this thesis envisions a future-proof technology stack adaptable to emerging network protocols including comprehensive support for IPv4 and IPv6, along with mandatory UDP support. Already, during the writing of this thesis the compatibility, feature set, and interoperability of different implementations of libp2p changed drastically. At the current state, the QUIC transport is only available to the Go implementation, while discovery protocols differ vastly by implementation [6].

In light of the above considerations, it appears that libp2p may be a great library in the future, but is currently too unstable to use in production, and for our consideration, the ongoing development is too fast-paced to integrate into slow-moving software products.

6 Selection of Technologies

6.1 Criteria for Selection

6.1.1 Overview

The **criteria** below are based on the previously stated success factors.

6.1.2 Low Latency

Whether a technology introduces latency or is able to prevent latency.

This criterion has been divided into the following sub-criteria:

No addition of latency : Whether a technology or approach introduces additional latency.

- ✔ Means no or close to none latency is introduced.
- ~ Means that some latency will definitely be added.
- ✗ Denotes a relevant impact on latency.
- N/A Means that this criterion is not applicable.

Prevention of latency : Whether a technology or approach is able to reduce latency (e.g. by caching).

- ✔ Means that this technique can reduce latency.
- ~ States that latency will be indifferent.
- ✗ Denotes that the technology won't improve latency.
- N/A Means that this criterion is not applicable.

Compared latency : When multiple technologies introduce latency we want to compare them depending on their impact.

- ✔ Denotes little impact.
- ~ Denotes some impact, and in comparison to the alternatives.
- ✗ Denotes the most impact.
- N/A Means that this criterion is not applicable.

6.1.3 Easy Setup

Whether a technology requires user interaction on startup or the technology requires a proficient user or administrator.

We broke this criterion down into the following sub-criteria:

Does not require user interaction : Whether a technology or approach requires user interaction.

- ✔ Means that a user can run this technology without interaction.
- ~ Describes that the user will probably need to interact with the tool but will be able to resolve it easily.
- ✗ Denotes will require some interrupting interaction.
- N/A Means that this criterion is not applicable.

Does not require proficient user interaction : Whether a user needs to be trained to use the technology or an approach properly.

- ✔ Means that a user can run this technology without proficiency.
- ~ Some users are expected to face problems.
- ✗ Denotes that users are expected to get trained.
- N/A Means that this criterion is not applicable.

Does not require admin interaction : Whether a technology or an approach may need a network or system administrator to function properly (either skill or training, as well as administrative privileges. e.g.: setting up Domain Name System () records).

- ✔ Means that no additional privileges are required to set up or operate this technique.
- ~ Some privileges like firewall rules.
- ✗ Denotes that administrative power is needed.
- N/A Means that this criterion is not applicable.

6.1.4 Resilient

Whether a technology introduces a single point of failure, vastly increases complexity/attack surface, if the technology is affected by churn in the network, or if the technology can be vastly affected by some misbehaving peer.

Not introducing failure points : Whether a technology or approach may introduce a (single) point of failure.

Fosters resilience : Whether a technology or an approach may increase the network's resistance against censorship/failure/data loss.

Not introducing complexity : Whether a technology or an approach may increase complexity on an algorithmic level. E.g. increase the total amount of states/phases or different blocks.

Little additional attack surface : We acknowledge that adding something will inevitably increase the code base and therefore, increase the attack surface. Nonetheless, different techniques leave a different footprint on the project. We also consider increasing the exposure to the network (e.g. by opening additional ports) as a huge increase in the context of the attack surface. We therefore compare the alternatives by their impact and use the symbols to indicate the relative impact, just as described in the criterion *compared latency*.

Resistance against misbehaving peers : Whether a technology is dependent on all peers working correctly and fair.

Churn resistant : Whether the technology is able to withstand high churn rates.

6.1.5 Future Proof

Whether technology is likely to last or it may become obsolete in a foreseeable amount of time.

IP versions : Whether a technology or approach or framework is able to utilize IPv4 and IPv6.

TCP and UDP : Whether a technology or an approach or framework is able to work with TCP and UDP.

Usage as intended : Whether a technology or an approach or framework is used as intended. e.g. UDP is used in amplification attacks why new standards like QUIC or DNS over HTTPS (DoH) are created. We consider the unrestricted use of technology that is currently actively exploited (see UDP based amplifications explained in the recent publication [156]) as downside and not future proof.

6.1.6 Vast Extensibility

If it's possible to tailor each relevant part to our needs (e.g. whether a library allows to switch out the used cryptography or consensus mechanism) and the technology is free of charge (e.g. being free of charge in general and changing something is free of charge).

6.1.7 Disqualifiers

Combining our literature review with these criteria, we deduce the following **disqualifiers**:

No DHT : Since DHT is most affected by churn and inhibits huge attack surfaces, we consider it an unsafe option for bootstrapping a peer-to-peer network. Therefore, the presence of a DHT will greatly affect our resilience rating.

Authentic bootstrap methods : Since eclipse attacks are the most important threats in DLT networks nowadays potentially unsafe bootstrap mechanisms like broadcasting to the local network or fetching non-authenticated out-of-band data should not be incorporated.

6.1.8 Optional

Since comparable NAT-traversal is not tackled by other products (e.g. IPFS) this still is a great usability bonus, but not a required criterion.

6.2 Topology of peer-to-peer Networks

6.2.1 Overview

	No Addition of Latency	Prevention of Latency	Compared Latency	Does not require User Interaction	Does not require Proficient User Interaction	Does not require Admin Interaction	Not introducing Failure Points	Fosters Resilience	Not introducing Complexity	Little Additional Attack Surface	Resistance Against Misbehaving Peers	Churn Resistant	Future Proof	Vast Extensibility	No DHT	Authentic Bootstrap Methods	NAT Traversal
Centralized	✔	~	✔	✔	✔	✔	✗	✗	✗	~	✔	~	~	✔	✔	✔	✔
Pure p2p	~	✔	~	~	~	✔	✔	✔	✔	~	✗	~	✔	~	✔	~	~

Table 6.1: How we classify the options for topology choices for peer-to-peer networks as detailed in subsection 2.7.3 compared to the criteria outlined in section 6.1, detailed explanation about the reasoning can be found in subsection 6.2.2.

6.2.2 Reasoning (see Table 6.1)

Based on the literature on Gnutella and Napster we conclude that a hybrid/centralized p2p network will experience less latency since this central entity can form the network and offload bandwidth or computing-intensive tasks. A well-known example are BitTorrent Trackers. Starting from a fully decentralized approach, we assume that the latency of end users is more impactful than from a central entity.
We conclude that central entities do not scale as cost-efficient as decentralized networks, therefore we state that pure peer-to-peer networks prevent by-design latency while central entities will suffer increased load.
We conclude that having a central entity would provide a more user-friendly bootstrap process.
We conclude that having central entities will inevitably become failure points and are prone to censorship.
We assume that a pure peer-to-peer system is comprised of just one type of peer, therefore a centralized/hybrid peer-to-peer solution introduces more complexity by having more than one type of peer.
Having a centralized entity with authoritative power makes the network less susceptible to malicious peers.
Based on the increase in internet-enabled devices[1] we estimate that protocols that are future-proof will need to handle more requests than an average central entity can handle.
We assume that centralized networks can perform network-wide updates/extensions more easily than pure peer-to-peer networks.
We assume that a central entity can way more easily authenticate the bootstrap process and can provide more options for NAT traversal.

6.2.3 Decision

We decided to use a pure peer-to-peer network where all peers have the same codebase and the same privileges (meaning there won't be a central index or super peers). We acknowledge that in a private ledger setting as described a central entity is very likely to provide servers running the software.

[1] we base this on common knowledge as well as documented statistics like [154]

6.3 Data Placement

6.3.1 Overview

	No Addition of Latency	Prevention of Latency	Compared Latency	Easy Setup	Not introducing Failure Points	Fosters Resilience	Not introducing Complexity	Little Additional Attack Surface	Resistance Against Misbehaving Peers	Churn Resistant	Future Proof	Vast Extensibility	No DHT	Authentic Bootstrap Methods	NAT Traversal
Centralized Index	✔	~	~	✔	✗	✗	~	✗	✔	✔	~	~	✔	N/A	✔
Distributed Index	✗	✔	✗	~	✔	✔	✔	✔	~	✗	✔	✔	✗	N/A	~
Local Index	~	✗	✗	~	✔	✗	~	~	~	✗	~	✔	✔	N/A	~
Aggregated Index	~	✗	~	~	~	~	✗	~	~	~	~	~	✔	N/A	~
Incentive Dup.	✔	✔	~	~	✔	✔	✗	✗	~	~	✔	~	✔	N/A	~
Fully Replicated	✔	✔	✔	~	✔	✔	✔	✔	✔	✔	✔	✔	✔	N/A	~
Delegated Rep.	✔	~	✔	✔	~	~	~	~	✗	~	✔	~	✔	N/A	~

Table 6.2: How we classify the data placement options as detailed in subsection 2.7.4 compared to the criteria outlined in section 6.1. Detailed explanation about the reasoning can be found in subsection 6.3.2.

6.3.2 Reasoning (see Table 6.2)

Based on the necessary query connections, local indices can provide high speed when the required data is available. However, distributed indices may require multiple requests before identifying the desired target. Aggregated indices need several lookup iterations but are less intense than distributed indices.

Due to bundling requests against centralized resources with limited capacity, larger use cases demand powerful centralized components to avoid performance degradation. Although distributed indices offer more potential peers with the required data, the presence of more entries limits the potential of latency prevention. Conversely, local indices have limited storage, which reduces the chance of finding an entry in a similar scenario. We conclude that aggregate indices still depend on the nodes underneath, which may increase latency. Delegating requests does not address the scaling issue, as the counterpart still needs to perform the query.

The compared latency is listened in the following orders, based on our expectation of the respective latency:

1. Fully replicated

   Since all required data is already present, there is no need for communication to other peers.
2. Delegated replications

   It is essentially the same case as with fully replicated, albeit with one additional connection to the replication.
3. Centralized index

   Since all clients need to communicate with this instance, it is more likely to have a higher latency than a delegated replication that can be replicated for the desired clients.
4. Aggregated index
5. Incentive Duplication
6. Local index
7. Distributed index

A centralized approach is easy to set up because there are fewer dependencies. Other approaches require more work. However, this does not affect the end user.

Since there is a single instance with all the required data, this instance becomes a single point of failure. If connections to other nodes are required, those nodes are also potential points of failure.

A centralized index is classified with the lowest resilience since taking over the central entity causes network-wide issues. However, a centralized index can heal itself if it only receives bad information from one source and other peers provide correct information. A local index poses the threat that information may not be tested for plausibility and altered if required.

Maintaining a centralized index places a high demand on synchronization. Aggregating updated information from multiple sources also adds complexity. An incentive layer adds complexity not only for the data itself but also for the incentive. Delegated replication also adds complexity by

requiring an API for the clients using the replication.

The different options provide different additional attack surfaces: A centralized index carries the risk that a successful attack will affect all peers, whereas the information can be verified and attacks can be neutralized if only one of a sufficient number of peers provides malicious information. In the case of incentive duplication, the additional layer and the ability to receive rewards directly increase the attack surface. Both aggregation code and local indexes require data to be parsed. This parsing can also lead to security problems.

Since both a centralized approach and a fully replicated approach hold all the necessary data, these possible solutions are more resilient to misbehaving peers. Conversely, the other solutions depend on other nodes, resulting in a higher risk of experiencing problems when misbehaving peers occur.

A centralized index and fully replicated indexes do not depend on clients that may or may not leave the network, resulting in good churn resistance. Other indexing methods rely on peers, resulting in significantly reduced churn resistance.

We conclude that both fully distributed (fully replicated and distributed) and local systems can be easily modified without disrupting the operation of existing nodes.

Data placement is not considered relevant for authentic bootstrapping.

NAT traversal is simplified by having a single point of contact with the node hosting the centralized index.

6.3.3 Decision

We decided on a peer-to-peer network that fully duplicates the whole ledger to all nodes. Since the intended solution is a protected chain with a form of Proof of Authority (PoA), we also authenticate the client using a Public Key Infrastructure (PKI) and therefore shall mitigate the risk of misbehaving peers.

We deduce that a peer that is authoritative in an PoA system cannot be a misbehaving peer, since it or the victim would be able to publish that information into the network and it would get accepted.

	No Addition of Latency	Prevention of Latency	Compared Latency	Easy Setup	Not introducing Failure Points	Fosters Resilience	Not introducing Complexity	Little Additional Attack Surface	Resistance Against Misbehaving Peers	Churn Resistant	Future Proof	Vast Extensibility	No DHT	Authentic Bootstrap Methods	NAT Traversal
No Treatment	✔	~	✗	✔	✔	✗	✔	✔	✔	✔	✗	N/A	N/A	N/A	N/A
Consensus	✔	~	✗	~	~	✔	✗	~	✗	N/A	✔	✗	N/A	N/A	N/A
Push & Pull	✔	✔	✔	✔	~	✔	~	~	✔	✔	✔	N/A	N/A	N/A	N/A
Flood Broadcast	~	✔	~	✔	✔	~	✔	✔	✔	✔	✗	N/A	N/A	N/A	N/A
Caching	✔	✔	~	✔	~	~	~	✔	✔	N/A	✔	N/A	N/A	N/A	N/A
Deduplication	✔	✔	~	✔	~	✔	✗	✔	✔	N/A	✔	~	N/A	N/A	N/A
Logical Locality	✔	✔	✔	✔	✗	~	~	✗	✗	✔	✔	N/A	N/A	N/A	N/A
Topology	✔	✔	✔	✔	~	✔	✗	~	✗	✗	✔	N/A	N/A	N/A	N/A
Select by Capacity	✔	✔	✔	~	✗	✔	~	~	~	✗	✔	N/A	N/A	N/A	N/A
Peer Clustering	✔	✔	✔	✔	✔	✔	✗	~	~	✗	~	N/A	N/A	N/A	N/A

Table 6.3: How we classify options for reducing latency as outlined in subsection 2.7.5 compared to the criteria outlined in section 6.1. Detailed explanation about the reasoning can be found in subsection 6.4.2.

6.4 Latency

6.4.1 Overview

6.4.2 Reasoning (see Table 6.3)

Considering the existence of latency and limited throughput, flooding a peer-to-peer network can increase latency when the needed throughput exceeds the node's capacity.

We conclude that not taking latency into account does change the latency, hence the classification

as indifferent. We also conclude that no consensus mechanism can actively prevent latency.
We rank technologies designed to reduce latency based on their potential to reduce latency, therefore no treatment and changing consensus get ranked lowest because they do not affect latency. Flooding is ranked medium because it might cause more latency but also might improve the user experience since it reaches all nodes. Caching and deduplication are ranked medium because we think they are more like a hotfix than a solution. We reason that changing topology, selecting high-capacity peers, or clustering will scale and have a great potential to reduce latency.
Changing the Consensus may be complicated for the user (e.g. freezing tokens for a PoS procedure). We also deduce that estimating the correct capacity per node might be challenging and might need some configuration.

Lack of adequate measures leads to failure exposition such as flooding, clustering, and capacity abusiveness. Flooding introduces latency beyond the operational capacity of individual nodes, characterized by the uncontrolled and indiscriminate dissemination of data by nodes. This latency can have a detrimental effect on the network's ability to maintain timely and efficient communications. Furthermore, without proper mechanisms to curb such behavior, peer-to-peer networks are vulnerable to simple flooding attacks, where malicious actors deliberately overload the network with excessive data. In addition, the lack of effective clustering strategies and capacity utilization policies can lead to a sub-optimal allocation of resources, which reduces the overall efficiency of the network. In the context of distributed systems, the lack of proactive measures to address these points of failure can undermine the robustness of the network, leaving it vulnerable to disruption and misuse.

Passive inaction in response to network outages or vulnerabilities can compound problems by leaving the network without mitigation strategies, leaving it vulnerable to various threats. Push and pull mechanisms, when used judiciously following an outage, can facilitate a more controlled process of data dissemination compared to broadcasting alone, and help to ensure that network functionality is restored in an efficient manner.

While clustering based on logical locality increases network efficiency, it can create a vulnerability to cluster failures. If nodes within the same logical locality are clustered together, then a failure in one part of the network can lead to a cascade of failures that affect the entire cluster.

Complexities are intrinsic to the design and operation of distributed ledger technologies and peer-to-peer networks. This complexity arises from the subtleties of consensus mechanisms, the need for low latency optimization, the challenges of efficient deduplication, and the diverse strategies

required for effective network topology and cluster management. The ability to navigate through this complexity is critical to ensuring that they are robust and effective.
Introducing locality, complexity in consensus mechanisms, using push and pull logics, and managing network topology, capacity and clusters all increase attack surfaces in distributed systems. To ensure the security and integrity of such systems, understanding and mitigating these vulnerabilities is paramount.
Misbehaving peers, especially in the context of consensus, locality and topology management, pose a significant challenge in distributed systems. To effectively deal with such misbehavior, robust consensus protocols are required. Capacity estimation and clustering, although essential for network efficiency and organization, require varying degrees of cooperation among nodes and must be carefully managed to mitigate the risks of misbehaved peers.
The impact of churn in peer-to-peer networks varies, with some nodes unaffected and others, particularly those heavily dependent on neighbors, suffering disruption. Highlighting the importance of churn management for network resilience and stability, proactive strategies and proximity-based interactions can help mitigate the effects of churn.
A “future-proofed” peer-to-peer network should prioritize optimizing latency, scalability, and adaptability. Long-term viability can be improved by minimizing unnecessary traffic and promoting a unified network fabric. It is also essential to maintain extensibility to accommodate changing requirements and technologies as the network evolves, especially in critical areas such as consensus and deduplication. In an ever-changing landscape of technology and user requirements, this holistic design approach ensures that the peer-to-peer network remains resilient and relevant.

6.4.3 Decision

We decided on using push and pull for the network design and will enable caching by adding a validity period to messages where possible. We intend on embedding relative time sources (e.g. Lamport timestamps) into the protocol to aid the consensus while allowing the consensus mechanism to be changed.

6.5 Free Riding

6.5.1 Overview

	No Addition of Latency	Prevention of Latency	Compared Latency	Easy Setup	Not introducing Failure Points	Fosters Resilience	Not introducing Complexity	Little Additional Attack Surface	Resistance Against Misbehaving Peers	Churn Resistant	Future Proof	Vast Extensibility	No DHT	Authentic Bootstrap Methods	NAT Traversal
Monetary	✔	✔	✔	~	~	✗	✗	✗	✗	✔	N/A	N/A	N/A	N/A	N/A
Reciprocity	✗	✔	~	✔	✔	✔	✔	✔	~	~	N/A	N/A	N/A	N/A	N/A
(Global) Reputation	~	✔	~	✔	✔	~	~	~	✔	✗	N/A	N/A	N/A	N/A	N/A

Table 6.4: How we classify methods of reducing free-riding as outlined in subsection 2.7.6 compared to the criteria outlined in section 6.1. Detailed explanation about the reasoning can be found in subsection 6.5.2.

6.5.2 Reasoning (see Table 6.4)

Some reasons for the given matches:

- We consider monetary incentives as a mismatch/interference with a transparent consensus, because non-paying peers may not be able to publish to the ledger.
- Promoting monetary incentives also does not increase security/resilience. Misbehaving peers would have an advantage in the peer-to-peer network if the threat actor spends enough money.
- Penalizing peers can exclude peers which could lead to a situation where a peer is eclipsed and not be able to participate in the consensus protocol.

- Penalizing peers can exclude peers which may put peers at a disadvantage or discourage participation which may add delay on a global scale but not on an individual scale.
- Penalizing peers as a concept may be susceptible to the digital equivalent of bad-mouthing.
- Since a newly joined peer is neither good nor bad, incentivizing good peers may either put existing connections or new connections at a disadvantage. Hence adding the probability of adding latency on joining.
- Keeping a global record for reputation may emerge the need for another overlay network (e.g. DHT) or may need a second consensus on how contradicting reputation is handled. Overlay networks like DHT are susceptible to churn.

6.5.3 Decision

Since we expect free riding in a private ledger a subpar problem we opt for the more easy-to-implement reciprocity approach. Because with that approach, the problem is addressed without the creation of a monetary incentive layer or a complex system for managing global reputation.

6.6 Message Broadcasting

6.6.1 Overview

	No Addition of Latency	Prevention of Latency	Compared Latency	Easy Setup	Not introducing Failure Points	Fosters Resilience	Not introducing Complexity	Little Additional Attack Surface	Resistance Against Misbehaving Peers	Churn Resistant	Future Proof	Vast Extensibility	No DHT	Authentic Bootstrap Methods	NAT Traversal
Flood	~	~	~	✔	✔	✔	✔	✔	✔	✔	✗	~	N/A	N/A	✔
Tree-based	✔	~	✔	✔	~	~	✔	✔	~	~	~	✔	N/A	N/A	~
Gossip	✔	✔	✔	✔	✔	~	~	~	✔	✔	✔	✔	N/A	N/A	~
Random Walk	✗	✗	✗	✔	~	✗	✔	✔	~	✗	✗	✔	N/A	N/A	~

Table 6.5: How we classify methods of message broadcasting in peer-to-peer networks as detailed in subsection 2.7.7 compared to the criteria outlined in section 6.1. Detailed explanation about the reasoning can be found in subsection 6.6.2.

6.6.2 Reasoning (see Table 6.5)

We conclude that using different paths simultaneously is faster. Therefore, random walks are not an efficient solution due to their unpredictability in terms of required steps and thus time. Flooding is not efficient and wastes resources because it has no significant search logic.

As mentioned above, flooding utilizes a high bandwidth whereas both tree-based approaches and random walks may be terminated before finishing.

Tree-based broadcasting can be susceptible to interference. This is especially true if the tree structure is disrupted by node failures or malicious activity. Likewise, RW broadcasting can be vulnerable to interference, as the random selection of neighbors can lead to inefficient message

propagation. These schemes may still have limitations in their resilience to interference, although they have been studied and improved over time. When designing a peer-to-peer communication system, it's important to consider the trade-offs between different broadcasting schemes and their susceptibility to network interference.

It's important to clarify that while flooding in peer-to-peer networks provides high resilience to message propagation, it comes at the cost of scalability and network efficiency. Flooding sends messages indiscriminately to all nodes. This can lead to excessive network traffic and congestion, potentially overwhelming the network and causing inefficiencies. However, it is not necessarily the most efficient or sustainable option with respect to network utilization and scalability. The tradeoff between resilience and resource efficiency should be given careful consideration in the choice of message delivery method in peer-to-peer networks.

Compared to simpler methods such as flooding or tree-based broadcasting, gossip-based protocols are generally more complex. This increased complexity is due to the iterative and probabilistic nature of gossip. Nodes randomly choose their neighbors and engage in multiple rounds of message propagation. This complexity requires more sophisticated coordination and maintenance mechanisms, although it offers benefits such as reduced network traffic and improved resilience.

A malicious peer can potentially increase message delay by withholding or selectively providing information, thereby disrupting the normal flow of updates, in a network where pull mechanisms are available. Conversely, in a network without pull mechanisms, nodes may miss updates if a malicious peer deliberately withholds or delays the forwarding of messages. This can lead to inconsistencies and potential disruptions in the network.

Because flooding and gossiping rely on less structured network topologies, allowing messages to propagate through multiple paths, they are typically more resilient to churn. In contrast, tree-based methods may have difficulty in forming and maintaining their structures in the presence of frequent node churn, potentially hindering message delivery. Random walk can also be affected by churn because it relies on randomly selecting neighbors, which can become less efficient as nodes join and leave the network.

NAT is typically a challenge for flooding because it involves the indiscriminate propagation of messages, which can be made more difficult by NAT devices. NAT devices often limit incoming connections, making flooding difficult to reach nodes behind NATs by blocking multiple connections from different peers.

Gossip-based protocols appear to be a robust choice for the future as they offer a good balance

between message efficiency and resilience, making them well-suited to scalable and decentralized networks. On the other hand, as networks grow and resource constraints become more severe, flooding may not be sustainable in the long run due to its excessive overhead. While random walk schemes are inherently resilient, they can be less efficient in terms of message propagation.

6.6.3 Decision

We decided to use a gossip-based approach.

6.7 Searching

Since in DLT every peer keeps a copy of the whole ledger, searching is not an issue of the peer-to-peer network because all the data is already available locally.

6.8 Churn in peer-to-peer Networks

6.8.1 Overview

	No Addition of Latency	Prevention of Latency	Compared Latency	Easy Setup	Not introducing Failure Points	Fosters Resilience	Not introducing Complexity	Little Additional Attack Surface	Resistance Against Misbehaving Peers	Churn Resistant	Future Proof	Vast Extensibility	No DHT	Authentic Bootstrap Methods	NAT Traversal
Accepting	✔	~	✔	✔	✗	✗	✔	✗	N/A	✗	✗	✔	N/A	N/A	N/A
Tiering	✔	✔	✔	✔	✔	✔	~	✔	N/A	✔	✔	✔	N/A	N/A	~
Peer Degree	✔	✔	✔	✔	~	✔	✔	✔	N/A	✔	✔	✔	N/A	N/A	N/A

Table 6.6: How we classify methods to handle churn as described in subsection 2.7.9 compared to the criteria outlined in section 6.1 in Peer-to-peer networks. Details explanation about the reasoning can be found in subsection 6.8.2.

6.8.2 Reasoning (see Table 6.6)

Neglecting the issue of latency can result in a sub-optimal network that fails to meet user expectations and operational requirements, which underlines the importance of proactive strategies to prevent latency in peer-to-peer networks.

Inadequate node degree or connectivity can make nodes more susceptible to failure, reducing network resilience. Conversely, excessive connectivity at certain nodes can inadvertently create semi-centralized entities, potentially raising network centralization concerns or indicating the presence of an ongoing eclipse attack, where a subset of nodes disproportionately controls communication. For network robustness, security and equitable participation, achieving a balanced degree distribution is critical.

Resilience requires proactive measures to be taken to prepare for, respond to and recover from adverse events, disruptions and failures. Inaction or neglect of potential vulnerabilities and challenges can compromise the network's ability to withstand and adapt to changing conditions. With the proliferation of decentralized and peer-to-peer networks, it is anticipated that churn rates may increase in the future due to a number of factors such as device mobility, changes in user behavior and network scaling. Failure to develop robust churn management strategies and mechanisms can be a barrier to network adaptability and sustainability. To be future-proof, peer-to-peer networks must prioritize churn management as a core component of their design and operation. This means implementing robust protocols, efficient resource management and adaptive strategies that thrive in dynamic churn environments.

6.8.3 Decision

We decided to manage the peer's degree by enforcing an upper and a lower connection limit following [32] recommendation. We also try to connect with peers whose peer list has the least overlap with our already established peers. This approach should help to make the network more interconnected without creating clusters.

6.9 Message Routing/Forwarding

6.9.1 Overview

	No Addition of Latency	Prevention of Latency	Compared Latency	Easy Setup	Not introducing Failure Points	Fosters Resilience	Not introducing Complexity	Little Additional Attack Surface	Resistance Against Misbehaving Peers	Churn Resistant	Future Proof	Vast Extensibility	No DHT	Authentic Bootstrap Methods	NAT Traversal
Letterbox	✔	✔	✔	✔	✗	✗	✗	✗	✗	✗	✔	✔	✗	N/A	N/A
Close Peer [108]	~	✔	✔	✔	✔	✔	~	✔	~	✔	✔	✔	✔	N/A	N/A
Random Walk	✗	✗	✗	✔	✗	✔	✔	✔	✗	✗	~	✔	✔	N/A	N/A
Broadcast	✔	~	~	✔	✔	✔	✔	~	✔	✔	✔	✔	✔	N/A	N/A

Table 6.7: How we classify the approaches to message forwarding as outlined in subsection 2.7.10 compared to the criteria outlined in section 6.1. Detailed explanation about the reasoning can be found in subsection 6.9.2.

6.9.2 Reasoning (see Table 6.7)

Having a rendezvous point that can be located by both parties in a constant amount of time (O(1)) can indeed facilitate ideal low latency communication by enabling fast and efficient connections. However, finding the shortest path for communication through the network can be complex due to routing challenges, especially in highly dynamic peer-to-peer environments. While random walk methods can eventually find the destination, they may not be as efficient for low-latency communications, introducing additional latency due to potential loops and lack of a guaranteed path. Broadcasting, due to its ability to reach multiple receivers at the same time, can be an effective and low-latency method of communication, especially if it is used judiciously within the constraints

of the network.
Strategies such as letterboxing or using efficient routes and close peers can help maintain low latency as the network scales because they focus on minimizing unnecessary hops and ensuring that messages reach their destinations efficiently. Conversely, actively flooding the network with messages or allowing messages to wander endlessly can significantly increase latency with more nodes by causing inefficient resource utilization and congestion. Therefore, in the context of latency prevention in peer-to-peer networks, the adoption of targeted and efficient routing methods is essential to ensure a sustainable and low-latency communication environment, especially as the network grows in size and complexity.
Peer discovery in a distributed system is a non-trivial task that often requires sophisticated algorithms and data structures such as distributed hash tables (DHTs). This additional layer of complexity can have an impact on system efficiency, resource utilization and overall manageability.
Random walks are inherently based on the random selection of neighbors, and as networks grow in size, the efficiency of such methods can decrease. In larger networks, there is an increased chance that a random walk method will encounter nodes that are far away from the destination, potentially resulting in longer paths and higher latencies. Looking ahead, it's important to recognize the scalability limitations of random walks and consider alternative routing and forwarding strategies that are more suited to the growth and complexity of larger peer networks.

6.9.3 Decision

Since we plan on using Gossip for broadcast we want to incorporate some ideas from [108] and cache messages based on probabilities along the path through the network.

6.10 Bootstrap

6.10.1 Overview

	No Addition of Latency	Prevention of Latency	Compared Latency	Does not require User Interaction	Does not require Proficient User Interaction	Does not require Admin Interaction	Not introducing Failure Points	Fosters Resilience	Not introducing Complexity	Little Additional Attack Surface	Resistance Against Misbehaving Peers	Churn Resistant	Future Proof	Vast Extensibility	No DHT	Authentic Bootstrap Methods	NAT Traversal
Config File	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	~	✔
DNS	✔	✔	✔	✔	✔	✗	✗	✔	~	✔	✔	✔	✔	~	✔	✔	✔
Prev. Conn.	✔	✔	~	✔	✔	✔	✔	✔	✔	✔	~	✔	✔	✔	✔	✔	✔
Overlay	✗	~	✗	✔	✔	✔	~	~	✗	✗	~	~	✗	✗	✗	✔	✗
Authentic OOB	✔	~	~	✔	✔	✔	✗	✗	~	✔	✔	✔	✔	✔	✔	✔	✔
OOB	✔	~	~	✔	✔	✔	✗	✗	✔	~	✔	✔	✗	✗	✔	✗	✔
Scanning	✗	✗	✗	✔	✔	✔	✔	✔	✗	✔	✔	✔	~	✔	✔	✔	✗
Passive Listen	✗	✗	✗	✔	✔	✔	✔	✗	~	✔	✔	✔	✔	✔	✔	✔	✗
DynDNS	✔	~	~	✔	✔	✔	✗	✗	✗	~	✔	✔	✔	~	✔	✔	✔
Global Multicast	✔	✔	✔	✔	✔	✔	~	✔	~	✗	~	~	✔	✔	✔	✔	~
Local Multicast	✔	✔	✔	✔	✔	✔	✔	✔	~	✗	~	✔	✔	✔	✔	✔	~

Table 6.8: How we classify bootstrapping methods as described in subsection 2.7.12 compared to the criteria outlined in section 6.1. Detailed explanation about the reasoning can be found in subsection 6.10.2.

We acknowledge that most peers will be discovered through exchanging peer lists, but we come to the conclusion that a peer needs to be joined to exchange peer lists, therefore it is not considered

a bootstrap method.

6.10.2 Reasoning (see Table 6.8)

We consider DNS as cachable and well-established, moving the scalability problem from one network to another does not solve it. Therefore, the delay of the additional load will occur, we also consider OOB as not scalable since it is a centralized service. We do not consider scanning as scalable due to the enormous resource usage on a global scale. We acknowledge the issues of IPv4 and the resulting adoption of NAT with the issue of connection problems. Therefore, we do not consider passive listening as scalable.
We conclude that DNS needs to be set up and an additional overlay may need assistance with the firewall. Scanning the internet should raise an alarm and therefore should be whitelisted by security personnel. Passive listening (e.g. accepting incoming connections) should need some administrative clearance and global multicast may provoke some security appliances.
We conclude that when everything is centralized, it may be prone to fail.
We argue that a future-proof technology shall not rely on other non-future-proof technologies and we do not consider inauthentic data future-proof. We consider behavior nowadays associated with malware or penetration testing as not future-proof since we expect it to interfere with alerting and we hope that with the rise of IPv6 issues with multicast/firewalls/NAT will resolve.
We do not consider a system we have no authority over as not in our control and therefore lacking extensibility.
We assume that another overlay will have the same issue as other overlays and therefore will not be able to traverse NAT. We argue that scanning and listening will face trouble with restricted access.

6.10.3 Decision

Starting from the evaluation above we will use the following bootstrap mechanisms:

Config file : Configuration file provided with all the information needed to bootstrap.

DNS : We believe that users expect things to have names. Also, we believe the administration power over a DNS-record will be available in a private chain setting.

Previous connections : A cache with previously used connections will be used. The cache may contain older nodes and or manipulated data, therefore, the cache shall be authenticated by the client itself and shall be used in a late stage of the bootstrap process.

6.11 Processing Power

6.11.1 Overview

	No Addition of Latency	Prevention of Latency	Compared Latency	Easy Setup	Not introducing Failure Points	Fosters Resilience	Not introducing Complexity	Little Additional Attack Surface	Resistance Against Misbehaving Peers	Churn Resistant	Future Proof	Vast Extensibility	No DHT	Authentic Bootstrap Methods	NAT Traversal
Consensus	N/A	N/A	N/A	N/A	✔	✔	~	✔	✔	~	✔	✔	N/A	N/A	N/A

Table 6.9: How we classify means on reducing the processing power needed to operate the peer-to-peer network as outlined in subsection 2.7.13, we excluded super-peers since as previously outlined we decided on a pure peer-to-peer solution. Means of reducing processing power are compared to the criteria outlined in section 6.1. Detailed explanation about the reasoning can be found in subsection 6.11.2.

6.11.2 Reasoning (see Table 6.9)

We point out that the topology was chosen beforehand and that the main driver for processing power can be the usage of PoW. In our private chain setup PoA shall be used.

7 Designing the peer-to-peer network

After discussing the pros and cons of technical decisions we now use them to design a peer-to-peer network.

7.1 Overview of the Design

The peer-to-peer network for a modern DLT solution is composed of the following building blocks:

Big Picture : The resulting network is a pure peer-to-peer network. IDs used by nodes are managed through an authority using Public Key Cryptography. As usual in DLT networks all nodes keep a copy of the complete ledger and can be contacted by light clients who do not keep the full ledger. We believe in extensibility and understandability - thus messages sent over the network shall be encoded using COSE+CBOR.

Auditing : Peers are audited by their peers. In intervals defined in the configuration file peers ask randomly chosen peers for their peer list and verify it by asking listed peers for their list as well. The degree of the peer is also audited.

Establish Connections : Each peer in a network needs to establish an initial connection somehow as highlighted in section 2.4. The means "*our*" DLT solution is considering state of the art are described in section 7.2 while section 7.3 describes how connections are handled.

Authenticate Nodes : In permissioned/private DLT solutions authentication needs to happen somehow, to check if a node is permissioned for this DLT. Section 7.4 will elaborate the intended solution.

Message Broadcasting : For efficient communications, a DLT needs means to broadcast/flood-/forward messages to all nodes. These messages are, for instance, the broadcast of new blocks.

Consensus : DLT applications are distributed, hence, they need a mechanism for consensus. The consensus mechanism for this DLT is out of the scope of this thesis and detailed in another

separate publication. The reader can assume it is a form of PoA. Section 7.6 describes what protocol properties are contributed to the consensus mechanism.

Communication Protocol : A network needs a protocol. Section 7.7 will discuss the different message types and states. As explained in subsection 2.6.1, we consider messages of a network protocol as objects to serialization. To try to minimize the attack surface we decided to use already established serialization techniques. We expect established techniques to be already studied for DoS, buffer overflows, or non-deterministic behavior.

7.2 Establish Initial Connections

Our solution will rely on a list of peer IP addresses and DNS names provided by a configuration file, as well as stored ones from previous communications for the bootstrapping process. After these initial connections, peer lists are exchanged, and additional connections are established. Some of the available (shortlived) connection slots perform this connect and exchange process continuously.

As detailed in section 6.10, we do not use local multicast like mDNS because we consider it as an attack vector to eclipse a node.

7.3 Connection Handling

Each node has a configurable lower- and upper-degree bound (how many connections it will accept/how many connections it tries to establish). This amount of connections is divided into 3 lists: long-lived connections, shortlived connections, and bootstrap connections. Peers are expected to reply to the peer they received a broadcast from. The sending peer will monitor these replies and will order the peers by a metric consisting of the round-trip time, whether the message contained news to the peer and the success rate of the transmission. Transactions where the peer received a message are treated the same way, the round trip time will be taken from the last peer list exchange with that node. The long-lived list reserves a configurable quota for peers of which more messages were received than transmitted (peers who spread news vs. peers who receive news). Good-performing peers are moved from the short-lived list into the long-lived list when their metric exceeds the worst metric of the long-lived list or a connection is dropped.

Connections in the short-lived lists are intended to churn. In configurable time intervals, a peer gets

replaced by a new peer. The peer is chosen by querying peer lists from some peers and inspecting their peer lists for overlap. The peer whose peer list has the least overlap with the node will be contacted. This should ensure that the peers expand their mesh. Since selecting peers, choosing from their peer list, and inspecting these nodes again is very resource intensive, the amount of peers selected and the amount of peers inspected from their list is configurable.

The third pool of connections serves bootstrapping only. Therefore, only a single query for the peerlist in a configured time window is allowed. This should enable the network to allow to constantly churn with an upper-bound degree.

When a client offers multiple ways of communication, we prefer QUIC over TCP over UDP. Since authentication is done in each message a UDP connection can be trusted as soon as a reply is seen. The amount of unacknowledged UDP packets tolerated before disconnecting the host is configurable.

Connections on the link layer are secured with TLS1.3. The exact cipher suites can be configured in a config file. The messages themselves are secured using CBOR Object Signing and Encryption (CBOR).

Readers may assume that this will result in encrypting twice, but COSE is used for signing only and works without encryption, therefore no CPU time is wasted on double encryption. Assuming that TLS1.3-based cipher suites enforce a MAC that serves similar goals then cryptographically signing each message and therefore performing integrity check twice is correct. We concluded that a low-level MAC e.g. in QUIC/UDP outweighs the costs, also as we decided the protocol to be stateless, we cannot trust this MAC as sufficient for authenticating messages.

7.4 Authentication

From a language security view, less complicated is preferred. Therefore, the protocol should not represent a state machine where a single handshake establishes trust. Instead, each message should be signed cryptographically. This results in each message being trustworthy by itself. Since each message is signed individually, only this message and a certificate store must check permissions. This certificate store can be implemented by placing the certificates in the DLT and thereby distributing each certificate to each user of the DLT. Permitting new users can thus be achieved by letting another (privileged) user place the certificate for the other user on the data ledger. The cryptographic primitives shall be enforced by the tool using this library. Hence, the library will not

limit the possibility of the cryptographic IDs in use.

7.5 Message Broadcast

As detailed in subsubsection 2.7.7.3, Gossip is a well-researched, versatile option. In our scenario, latency reduction is a key goal. As other Gossip modifications propose, the probability of selecting a node is tweaked: for this work, known peers are split into groups and a message should be delivered to at least one node in each group, as outlined in the Connection Handling explanation. As detailed in section 7.7, peers are monitored for their packet loss, stability and bandwidth. Our node selection is therefore based on selecting nodes per formed category, the amount of nodes picked from long-lived and short-lived connection pools is configurable. Based on the aggregated probability of failure (threshold is up to user configuration) messages are cached and transmitted again later (see section 6.9). We combine a hybrid of push and pull, therefore in a configurable time interval a randomly chosen node is pulled/queried for updates.

7.6 Consensus

The exact mechanism is out of the scope of this thesis. Readers may assume a type of PoA. To aid the consens mechanism each message contains a relative time stamp. We will use Lamport Timestamps.

7.7 Communication Protocol

The following message types are implemented: All messages should be treated stateless to keep protocol complexity low. A node may send a block to other nodes without being asked for or a node may not respond to queries. Therefore, nodes should drop the connections to other nodes when unwanted behavior is detected (e.g. receiving too many blocks that were not requested or being ignored by the other node).

Broadcast message : A message is broadcast via Gossip. The goal is low latency communication throughout the network rather than fault tolerance. This will be used to broadcast fresh blocks to all nodes. During the peer selection, the node picks peers from all three lists and transmits the message. As an extension to the Gossip protocol, a single acknowledgment is expected.

A flag can be set to indicate just a single recipient (referring to the cryptographic ID). Peers, who were connected to the recipient or are connected to the recipient will cache this message if the probability of that link failing (based on the monitoring of acknowledgments) exceeds a configured threshold. Peers will try to send after a configured amount of time and will drop the message after some retries.

Broadcast acknowledgment : A reply sent back to the peer a fresh block was learned from.

Query peerlist : Peers answer this query with their peer list. the list is shuffled to hide which peer is kept in which list by the node.

Peerlist acknowledgment : This is the answer to query peerlist.

Announce world state : Nodes send their known head of the ledger. This can be compared to the “inventory” message from Bitcoin. The peer shall not expect any response. If it would communicate something new to a peer it would reply with a query message.

Query message : This message indicates that a node missed the broadcast of a message and prompts the receiving peer to re-broadcast it.

8 Discussion

As this field is a matter of ongoing development, additional time and resources proved to be especially valuable. Due to this adaptation of the original time schedule, several important developments, such as the merge in Ethereum, could be taken into consideration for this work.

However, due to the extended timeframe and the additional information taken into account, the abundance of information slowed the progress of this work. This was especially the case with fast-paced projects like libp2p.

The initial objective of this work was to generate an in-depth evaluation of different libraries in this context and combine the learnings from this step to propose a guideline of how to create a modern peer-to-peer network.

This original goal could not be fully realized due to limitations with the chosen libraries. Some of those libraries are aimed at the usage in public peer-to-peer networks, while the goal of this work was to create a private peer-to-peer network. Other libraries had the issue that internal components such as functions and cryptographic algorithms could not be altered or swapped in a way to include alternatives within the allocated time.

With additional resources in terms of developers, a possible workaround to this issue might be to re-implement big portions of the libraries in question and thus facilitate a more modular design or at least use other cryptographic algorithms. However, this approach was not deemed feasible within the scope of a master's thesis.

In turn, the main learning of the evaluation is that none of the analyzed libraries fulfills all the requirements brought forward in the initial phase. Especially in terms of PoA or code stability, significant shortcomings could be observed. While providing many of the desired features, libp2p ultimately could not be chosen as the library to use as it is still a matter of rapid development and has not reached a status of sufficient stability in several features required for the use case of this work.

For this issue, a further extension of the timeframe is thought as not likely to result in a significantly

better stability of the library. As the high degree of ongoing development of new features does not allow an estimate of when the stability issues causing an issue at this state would be resolved. Thus, the main outcome of this work is the aggregated knowledge on different implementations of protected chains apart from hyperfabric.

9 Conclusion

Based on the analysis of various sources from literature and induction, we can draw several conclusions regarding the challenges and considerations in building a peer-to-peer network for a Distributed Ledger Technologies (DLT) application.
After reflecting on various challenges and problems as well as their respective solution, we designed a peer-to-peer network with the following key improvements:

- purely distributed peer-to-peer network,
- messages sent over the network use CBOR encoding and are COSE signed for stateless authentic communication,
- modification to the Gossip-broadcast algorithm to incentivize participation,
- refrain from using Kademlia or mDNS as bootstrap methods,
- and audit degree bounds of neighboring peers.

Various alternatives to these technologies were discussed and evaluated.
In summary, this thesis provides a literature overview of problems and challenges in peer-to-peer networks as well as a literature overview of potential solutions, and finally a breakdown of technologies suitable for creating a peer-to-peer network for a permissioned distributed ledger application.

9.1 Future Work

We conclude that the vast topic of DLT is not yet fully explored. We see that new technologies such as ProtoBuf are adopted one by one. We expect adoptions like CBOR+COSE from other DLT solutions in the near future.
Therefore, future work is expected to evaluate the effect of the standardization of QUIC for the peer-to-peer landscape. We also expect that following the Ethereum “merge” event a follow-up to [43] will be done. we expect that this publication will shed new light on the shared libp2p-ecosystem of IPFS, ETH2, and many more.

List of Tables

Glossary

Membership Service Providers	A Membership Service Provider is a term used by Hyperledger to refer to a certificate issuing instance, and defines protocols and mechanisms to verify an identity in the Hyperledger fabric ecosystem.
IPv4	Internet Protocol version 4
API	Application Programming Interface
BFT	Byzantine Fault-Tolerant
BGP	Border Gateway Protocol
BOINC	Berkeley Open Infrastructure for Network Computing
BSON	Binary JSON
CAN	Content-Addressable Network; usually refers to the network presented by Ratnasamy *et al.*
CAN	Content-Addressable Network
CBOR	Concise Binary Object Representation
CDN	Content Delivery Network
COSE	CBOR Object Signing and Encryption
COVID-19	Coronavirus Disease 2019
DAG	Directed Acyclic Graph
DApp	dezentralized application
DDoS	Distributed Denial of Service

DeFi	Decentralized Finance
DHT	Distributed Hash Table
DLT	Distributed Ledger Technologies
DNS	Domain Name System; System based on UDP to resolve the human preferred domain name into an (IP-) address.
DNS	Domain Name System
DoH	DNS over HTTPS
DoS	Denial-of-Service
dp	data placement
DTD	Document Type Definition
DTN	Delay-Tolerant Networking
Dyn-DNS	Dynamic DNS; Like DNS but receiving dynamic updates. Mostly used by consumer internet user getting a dynamic IP from their service provider.
ETH 1.0	Ethereum mainnet
ETH 2.0	Ethereum 2.0
EU	European Union
EVM	Ethereum Virtual Machine
FM	Frequency Modulation
FSA	File System Access
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
I/O	Input/Output
ICE	Interactive Connectivity Establishment
IEEE	Institute of Electrical and Electronics Engineers

IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IPFS	InterPlanetary File System
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
LangSec	Language-Theoretic Security, an approach where all possible inputs are defined as formal grammar and processed only by a validated parser for the formal grammar. The description given by IEEE's LangSec workshop [157]: "*The LangSec approach posits that the only path to trustworthy computer software that takes untrusted inputs is treating all valid or expected inputs as a formal language and the collection of respective input-handling routines as a recognizer for that language. LangSec shows that broad classes of vulnerabilities in systems handling untrusted inputs result from disregard for fundamental computability principles in design of data formats and their parsers*."

libp2p	Lib peer-to-peer
MANet	Mobile Ad hoc Network
mDNS	Multicast DNS
MIT	Massachusetts Institute of Technology
MiTM	Machine in The Middle
MP3	MPEG-1 Audio Layer III
MPEG	Moving Picture Experts Group
MSP	Membership Service Providers
NAT	Network Address Translation
nonce	Term used in cryptography for a “number used once” [158]
NP-hard	non-deterministic polynomial-time hardness
OOB	Out Of Band
OS	Operating System
OSI	Open Systems Interconnection model
p2p	peer-to-peer
PAT	Port Address Translation
PKI	Public Key Infrastructure
PoA	Proof of Authority
PoET	Proof of Elapsed Time
PoS	Proof of Stake
PoW	Proof of Work
RFC	Requests for Comments, nowadays considered as publication
RLP	Recursive-Length Prefix

SIP	Session Initiation Protocol
SSRF	Server Side Request Forgery
STUN	Session Traversal Utilities for NAT
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TLV	Tag-Length-Value
TTL	Time-to-live
TURN	Traversal Using Relays around NAT
UDP	User Datagram Protocol
URL	Uniform Resource Locator; refers to a string locating a specified resource. Example: `https://google.com`
XEE	XML External Entity
XML	Extensible Markup Language
XOR	eXclusive OR
XSD	XML Schema Definition
XXE	External Entity

Bibliography

[1] Kazuhiro Yamashita, Yoshihide Nomura, Ence Zhou, Bingfeng Pi, and Sun Jun, "Potential risks of hyperledger fabric smart contracts," in *2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, 2019, pp. 1–10. DOI: 10.1109/IWBOSE.2019.8666486.

[2] S. Brotsis, N. Kolokotronis, K. Limniotis, G. Bendiab, and S. Shiaeles, "On the security and privacy of hyperledger fabric: Challenges and open issues," in *2020 IEEE World Congress on Services (SERVICES)*, 2020, pp. 197–204. DOI: 10.1109/SERVICES48979.2020.00049.

[3] S. Shalini and H. Santhi, "A Survey on Various Attacks in Bitcoin and Cryptocurrency," in *2019 International Conference on Communication and Signal Processing (ICCSP)*, Apr. 2019, pp. 0220–0224. DOI: 10.1109/ICCSP.2019.8697996.

[4] Sebastian Henningsen, Martin Florian, Sebastian Rust, and Björn Scheuermann, "Mapping the Interplanetary Filesystem," in *2020 IFIP Networking Conference (Networking)*, Jun. 2020, pp. 289–297. DOI: https://doi.org/10.48550/arXiv.2002.07747.

[5] Dan S. Wallach, "A Survey of Peer-to-Peer Security Issues," en, in *Software Security — Theories and Systems*, Mitsuhiro Okada, Benjamin C. Pierce, Andre Scedrov, Hideyuki Tokuda, and Akinori Yonezawa, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2003, pp. 42–57, ISBN: 978-3-540-36532-7. DOI: 10.1007/3-540-36532-X_4.

[6] *Libp2p: Implementations*, en-us, Snapshot of the page during writing online https://web.archive.org/web/20240114224246/https://libp2p.io/implementations/. [Online]. Available: https://libp2p.io/implementations/ (visited on 09/12/2023).

[7] Google Trends, *Topic cryptocurrency*, google, Ed., [Online; queried on 13-Janurary-2022], Jan. 2012. [Online]. Available: https://trends.google.de/trends/explore?date=all_2008&gprop=youtube&q=%5C%2Fm%5C%2F0vpj4_b.

[8] Yuval Marcus, Ethan Heilman, and Sharon Goldberg, "Low-resource eclipse attacks on ethereum's peer-to-peer network.," *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 236, 2018.

[9] Ashika R Naik and Bettahally N Keshavamurthy, "Next level peer-to-peer overlay networks under high churns: A survey," *Peer-to-Peer Networking and Applications*, vol. 13, no. 3, pp. 905–931, 2020. DOI: https://doi.org/10.1007/s12083-019-00839-8.

[10] John Kolb, Moustafa AbdelBaky, Randy H. Katz, and David E. Culler, "Core Concepts, Challenges, and Future Directions in Blockchain: A Centralized Tutorial," *ACM Comput. Surv.*, vol. 53, no. 1, 9:1–9:39, Feb. 2020, ISSN: 0360-0300. DOI: 10.1145/3366370. [Online]. Available: https://doi.org/10.1145/3366370 (visited on 08/26/2020).

[11] Nabil El Ioini and Claus Pahl, "A Review of Distributed Ledger Technologies," en, in *On the Move to Meaningful Internet Systems. OTM 2018 Conferences*, Hervé Panetto, Christophe Debruyne, Henderik A. Proper, Claudio Agostino Ardagna, Dumitru Roman, and Robert Meersman, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2018, pp. 277–288, ISBN: 978-3-030-02671-4. DOI: 10.1007/978-3-030-02671-4_16.

[12] M. Krishna Ramanathan, V. Kalogeraki, and J. Pruyne, "Finding good peers in peer-to-peer networks," in *Proceedings 16th International Parallel and Distributed Processing Symposium*, 2002, pp. 1–8. DOI: 10.1109/IPDPS.2002.1015499.

[13] Chi-Kin Chau and Prithwish Basu, "Analysis of latency of stateless opportunistic forwarding in intermittently connected networks," *IEEE/ACM Transactions on Networking*, vol. 19, no. 4, pp. 1111–1124, 2011. DOI: 10.1109/TNET.2010.2103321.

[14] Eng Keong 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, Conference Name: IEEE Communications Surveys Tutorials, ISSN: 1553-877X. DOI: 10.1109/COMST.2005.1610546.

[15] "Open Systems Interconnection – Basic Reference Model: The basic model," International Telecommunication Union – Telecommunication Standardization Sector, Standard, Apr. 1993. DOI: 11.1002/1000/2820. [Online]. Available: http://handle.itu.int/11.1002/1000/2820.

[16] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein, *Introduction to algorithms* (The MIT Press), 3rd ed. London, England: MIT Press, Jul. 2009, ISBN: 0262033844.

[17] Suyash Gupta and Mohammad Sadoghi, "Blockchain transaction processing," *CoRR*, vol. abs/2107.11592, 2021. arXiv: 2107.11592. [Online]. Available: https://arxiv.org/abs/2107.11592.

[18] Satoshi Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," en, p. 9, [Online]. Available: https://bitcoin.org/bitcoin.pdf.

[19] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, and et al., "Hyperledger fabric," *Proceedings of the Thirteenth EuroSys Conference*, Apr. 2018. DOI: 10.1145/3190508.3190538. [Online]. Available: http://dx.doi.org/10.1145/3190508.3190538.

[20] Till Neudecker, "Security and anonymity aspects of the network layer of permissionless blockchains," Ph.D. dissertation, Karlsruher Institut für Technologie (KIT), 2019, 176 pp. DOI: 10.5445/IR/1000089033.

[21] Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun, "On the security and performance of proof of work blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16, Vienna, Austria: Association for Computing Machinery, 2016, pp. 3–16, ISBN: 9781450341394. DOI: 10.1145/2976749.2978341. [Online]. Available: https://doi.org/10.1145/2976749.2978341.

[22] Sishan Long, Soumya Basu, and Emin Gün Sirer, "Measuring miner decentralization in proof-of-work blockchains," *arXiv preprint arXiv:2203.16058*, 2022.

[23] Shijie Lin, "Proof of work vs. proof of stake in cryptocurrency," *Highlights in Science, Engineering and Technology*, vol. 39, pp. 953–961, 2023. [Online]. Available: https://doi.org/10.54097/hset.v39i.6683.

[24] Alfonso De la Rocha, David Dias, and Yiannis Psaras, "Accelerating content routing with bitswap: A multi-path file transfer protocol in ipfs and filecoin," 2021. [Online]. Available: https://research.protocol.ai/publications/accelerating-content-routing-

with-bitswap-a-multi-path-file-transfer-protocol-in-ipfs-and-filecoin/delarocha2021.pdf.

[25] Christian Cachin, "Architecture of the hyperledger blockchain fabric," in *Workshop on distributed cryptocurrencies and consensus ledgers*, Chicago, IL, vol. 310, 2016. [Online]. Available: https://www.zurich.ibm.com/dccl/papers/cachin_dccl.pdf.

[26] Jana Iyengar and Martin Thomson, "Rfc 9000 quic: A udp-based multiplexed and secure transport," *Internet Engineering Task Force*, May 2021, ISSN: 2070-1721.

[27] Eric Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.3*, RFC 8446, Aug. 2018. DOI: 10.17487/RFC8446. [Online]. Available: https://www.rfc-editor.org/info/rfc8446.

[28] Stephen Simon, "Peer-to-peer network management in an IBM SNA network," *IEEE Network*, vol. 5, no. 2, pp. 30–34, Mar. 1991, Conference Name: IEEE Network, ISSN: 1558-156X. DOI: 10.1109/65.75839.

[29] Sehyun Park, Seongwon Im, Youhwan Seol, and Jeongyeup Paek, "Nodes in the bitcoin network: Comparative measurement study and survey," *IEEE Access*, vol. 7, pp. 57 009–57 022, 2019. DOI: 10.1109/ACCESS.2019.2914098.

[30] "Peer-to-Peer-Netzwerke in der Praxis," de, in *Peer-to-Peer-Netzwerke: Algorithmen und Methoden*, ser. eXamen.press, Peter Mahlmann and Christian Schindelhauer, Eds., Berlin, Heidelberg: Springer, 2007, pp. 243–248, ISBN: 978-3-540-33992-2. DOI: 10.1007/978-3-540-33992-2_13. [Online]. Available: https://doi.org/10.1007/978-3-540-33992-2_13 (visited on 09/08/2020).

[31] Dimitris Vyzovitis, Yusef Napora, Dirk McCormick, David Dias, and Yiannis Psaras, *Gossipsub: Attack-resilient message propagation in the filecoin and eth2.0 networks*, 2020. DOI: https://doi.org/10.48550/arXiv.2007.02754. arXiv: 2007.02754 [cs.NI].

[32] Atul Singh, Miguel Castro, Peter Druschel, and Antony Rowstron, "Defending against eclipse attacks on overlay networks," in *Proceedings of the 11th workshop on ACM SIGOPS European workshop*, ser. EW 11, New York, NY, USA: Association for Computing Machinery, Sep. 2004, 21–es, ISBN: 978-1-4503-7807-9. DOI: 10.1145/1133572.1133613. [Online]. Available: https://doi.org/10.1145/1133572.1133613 (visited on 09/02/2020).

[33] W. D. Maurer and T. G. Lewis, “Hash table methods,” *ACM Comput. Surv.*, vol. 7, no. 1, pp. 5–19, Mar. 1975, ISSN: 0360-0300. DOI: 10.1145/356643.356645. [Online]. Available: https://doi.org/10.1145/356643.356645.

[34] Telesphore Tiendrebeogo and Mamadou Diarra, “Big data storage system based on a distributed hash tables system,” *International Journal of Database Management Systems*, vol. 12, pp. 1–9, Oct. 2020. DOI: 10.5121/ijdms.2020.12501.

[35] Petar Maymounkov and David Mazières, “Kademlia: A peer-to-peer information system based on the xor metric,” in *Peer-to-Peer Systems*, Peter Druschel, Frans Kaashoek, and Antony Rowstron, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 53–65, ISBN: 978-3-540-45748-0.

[36] Moritz Steiner, Taoufik En-Najjary, and Ernst W. Biersack, “A global view of kad,” in *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC ’07, San Diego, California, USA: Association for Computing Machinery, 2007, pp. 117–122, ISBN: 9781595939081. DOI: 10.1145/1298306.1298323. [Online]. Available: https://doi.org/10.1145/1298306.1298323.

[37] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker, “A scalable content-addressable network,” *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 161–172, Aug. 2001, ISSN: 0146-4833. DOI: 10.1145/964723.383072. [Online]. Available: https://doi.org/10.1145/964723.383072.

[38] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan, “Chord: A scalable peer-to-peer lookup service for internet applications,” *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 149–160, Aug. 2001, ISSN: 0146-4833. DOI: 10.1145/964723.383071. [Online]. Available: https://doi.org/10.1145/964723.383071.

[39] Antony Rowstron and Peter Druschel, “Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems,” in *Middleware 2001*, Rachid Guerraoui, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 329–350, ISBN: 978-3-540-45518-9.

[40] Huawei Huang, Jianru Lin, Baichuan Zheng, Zibin Zheng, and Jing Bian, “When blockchain meets distributed file systems: An overview, challenges, and open issues,” *IEEE Access*, vol. 8, pp. 50 574–50 586, 2020. DOI: 10.1109/ACCESS.2020.2979881.

[41] Liang Wang and Jussi Kangasharju, "Measuring large-scale distributed systems: Case of bittorrent mainline dht," in *IEEE P2P 2013 Proceedings*, 2013, pp. 1–10. DOI: 10.1109/P2P.2013.6688697.

[42] Ingmar Baumgart and Sebastian Mies, "S/Kademlia: A practicable approach towards secure key-based routing," in *2007 International Conference on Parallel and Distributed Systems*, ISSN: 1521-9097, Dec. 2007, pp. 1–8. DOI: 10.1109/ICPADS.2007.4447808.

[43] Seoung Kyun Kim, Zane Ma, Siddharth Murali, Joshua Mason, Andrew Miller, and Michael Bailey, "Measuring Ethereum Network Peers," in *Proceedings of the Internet Measurement Conference 2018*, ser. IMC '18, New York, NY, USA: Association for Computing Machinery, Oct. 2018, pp. 91–104, ISBN: 978-1-4503-5619-0. DOI: 10.1145/3278532.3278542. [Online]. Available: https://doi.org/10.1145/3278532.3278542 (visited on 09/23/2020).

[44] Soo Hoon Maeng, Meryam Essaid, and Hong Taek Ju, "Analysis of ethereum network properties and behavior of influential nodes," in *2020 21st Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2020, pp. 203–207. DOI: 10.23919/APNOMS50412.2020.9236965.

[45] "Devp2p/discv5 at master · ethereum/devp2p," GitHub. (2020), [Online]. Available: https://github.com/ethereum/devp2p/tree/master/discv5 (visited on 01/10/2024).

[46] Juan Benet, "Ipfs-content addressed, versioned, p2p file system (draft 3)," *arXiv preprint arXiv:1407.3561*, 2014. [Online]. Available: https://doi.org/10.48550/arXiv.1407.3561.

[47] Marjan Hericko, Matjaz B. Juric, Ivan Rozman, Simon Beloglavec, and Ales Zivkovic, "Object serialization analysis and comparison in java and .net," *SIGPLAN Not.*, vol. 38, no. 8, pp. 44–54, Aug. 2003, ISSN: 0362-1340. DOI: 10.1145/944579.944589. [Online]. Available: https://doi.org/10.1145/944579.944589.

[48] Christopher Späth, Christian Mainka, Vladislav Mladenov, and Jörg Schwenk, "SoK: XML parser vulnerabilities," in *10th USENIX Workshop on Offensive Technologies (WOOT 16)*, Austin, TX: USENIX Association, Aug. 2016. [Online]. Available: https://www.usenix.org/conference/woot16/workshop-program/presentation/spath.

[49] Kazuaki Maeda, "Performance evaluation of object serialization libraries in xml, json and binary formats," in *2012 Second International Conference on Digital Information and Com-*

munication Technology and it's Applications (DICTAP), 2012, pp. 177–182. DOI: 10.1109/DICTAP.2012.6215346.

[50] Pierre Bourhis, Juan L. Reutter, Fernando Suárez, and Domagoj Vrgoč, "Json: Data model, query languages and schema specification," in *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, ser. PODS '17, Chicago, Illinois, USA: Association for Computing Machinery, 2017, pp. 123–135. DOI: 10.1145/3034786.3056120. [Online]. Available: https://doi.org/10.1145/3034786.3056120.

[51] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann, "Restful web services vs. "big"' web services: Making the right architectural decision," in *Proceedings of the 17th International Conference on World Wide Web*, ser. WWW '08, Beijing, China: Association for Computing Machinery, 2008, pp. 805–814, ISBN: 9781605580852. DOI: 10.1145/1367497.1367606. [Online]. Available: https://doi.org/10.1145/1367497.1367606.

[52] Alvaro Muñoz and Oleksandr Mirosh, "Friday the 13th JSON Attacks," en, 2017. [Online]. Available: https://blackhat.com/docs/us-17/thursday/us-17-Munoz-Friday-The-13th-JSON-Attacks-wp.pdf.

[53] Christian Decker and Roger Wattenhofer, "Information propagation in the Bitcoin network," in *IEEE P2P 2013 Proceedings*, ISSN: 2161-3567, Sep. 2013, pp. 1–10. DOI: 10.1109/P2P.2013.6688704.

[54] "IEEE Standard for Ethernet," *IEEE Std 802.3-2012 (Revision to IEEE Std 802.3-2008)*, pp. 1–3747, 2012. DOI: 10.1109/IEEESTD.2012.6419735.

[55] Nabeel Shahzad, "S.im.pl serialization: Type system scopes encapsulate cross-language, multi-format information binding," Ph.D. dissertation, Texas A&M University, 2012.

[56] Ethereum Foundation, *Recursive-length prefix (rlp) serialization*, https://ethereum.org/en/developers/docs/data-structures-and-encoding/rlp, Accessed: (12-02-2024), 2023.

[57] *Protocol Buffers*, en. [Online]. Available: https://protobuf.dev/ (visited on 09/12/2023).

[58] Audie Sumaray and S. Kami Makki, "A comparison of data serialization formats for optimal efficiency on a mobile platform," in *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*, ser. ICUIMC '12, Kuala Lumpur, Malaysia: Association for Computing Machinery, 2012, ISBN: 9781450311724. DOI: 10.

1145/2184751.2184810. [Online]. Available: https://doi.org/10.1145/2184751.2184810.

[59] Geoffrey Litt, Seth Thompson, and John Whittaker, *Improving performance of schemaless document storage in PostgreSQL using BSON*, en. [Online]. Available: https://www.geoffreylitt.com/resources/Postgres-BSON.pdf.

[60] Álvaro Luis, Pablo Casares, Juan J. Cuadrado-Gallego, and Miguel A. Patricio, "PSON: A Serialization Format for IoT Sensor Networks," *Sensors*, vol. 21, no. 13, 2021, ISSN: 1424-8220. DOI: 10.3390/s21134559. [Online]. Available: https://www.mdpi.com/1424-8220/21/13/4559.

[61] Carsten Bormann and Paul E. Hoffman, *Concise Binary Object Representation (CBOR)*, RFC 7049, Oct. 2013. DOI: 10.17487/RFC7049. [Online]. Available: https://www.rfc-editor.org/info/rfc7049.

[62] Carsten Bormann and Paul E. Hoffman, *Concise Binary Object Representation (CBOR)*, RFC 8949, Dec. 2020. DOI: 10.17487/RFC8949. [Online]. Available: https://www.rfc-editor.org/info/rfc8949.

[63] Thomas Rix, Kai-Oliver Detken, and Marcel Jahnke, "Transformation between xml and cbor for network load reduction," in *2016 3rd International Symposium on Wireless Systems within the Conferences on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS-SWS)*, 2016, pp. 106–111. DOI: 10.1109/IDAACS-SWS.2016.7805797.

[64] *Guidelines on Technical Specifications for EU Digital COVID Certificates*, vol. 2, eHealth Network, Jun. 2022.

[65] Neil Daswani, Hector Garcia-Molina, and Beverly Yang, "Open Problems in Data-Sharing Peer-to-Peer Systems," en, in *Database Theory — ICDT 2003*, Diego Calvanese, Maurizio Lenzerini, and Rajeev Motwani, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2003, pp. 1–15, ISBN: 978-3-540-36285-2. DOI: 10.1007/3-540-36285-1_1.

[66] Harald Vranken, "Sustainability of bitcoin and blockchains," en, *Current Opinion in Environmental Sustainability*, Sustainability governance, vol. 28, pp. 1–9, Oct. 2017, ISSN: 1877-3435. DOI: 10.1016/j.cosust.2017.04.011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877343517300015 (visited on 05/03/2021).

[67] Almer S. Tigelaar, Djoerd Hiemstra, and Dolf Trieschnigg, "Peer-to-Peer Information Retrieval: An Overview," *ACM Trans. Inf. Syst.*, vol. 30, no. 2, 9:1–9:34, May 2012, ISSN: 1046-8188. DOI: 10.1145/2180868.2180871. [Online]. Available: https://doi.org/10.1145/2180868.2180871 (visited on 08/31/2020).

[68] Sebastian Henningsen, Daniel Teunis, Martin Florian, and Björn Scheuermann, "Eclipsing Ethereum Peers with False Friends," in *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*, Jun. 2019, pp. 300–309. DOI: 10.1109/EuroSPW.2019.00040.

[69] Yunhao Liu, Li Xiao, A-H Esfahanian, and Lionel M Ni, *Approaching Optimal Peer-to-Peer Overlays*, en. DOI: 10.1109/MASCOTS.2005.15. [Online]. Available: https://www.researchgate.net/publication/221082875_Approaching_Optimal_Peer-to-Peer_Overlays (visited on 09/10/2020).

[70] Quang Hieu Vu, Mihai Lupu, and Beng Chin Ooi, "Architecture of Peer-to-Peer Systems," en, in *Peer-to-Peer Computing: Principles and Applications*, Quang Hieu Vu, Mihai Lupu, and Beng Chin Ooi, Eds., Berlin, Heidelberg: Springer, 2010, pp. 11–37, ISBN: 978-3-642-03514-2. DOI: 10.1007/978-3-642-03514-2_2. [Online]. Available: https://doi.org/10.1007/978-3-642-03514-2_2 (visited on 06/01/2021).

[71] Santeri Paavolainen and Christopher Carr, "Security properties of light clients on the ethereum blockchain," *IEEE Access*, vol. 8, pp. 124 339–124 358, 2020. DOI: 10.1109/ACCESS.2020.3006113.

[72] Peter Triantafillou, Chryssani Xiruhaki, Manolis Koubarakis, and Nikos Ntarmos, "(PDF) Towards High Performance Peer-to-Peer Content and Resource Sharing Systems," en, Jan. 2003. [Online]. Available: https://www.researchgate.net/publication/220988197_Towards_High_Performance_Peer-to-Peer_Content_and_Resource_Sharing_Systems (visited on 08/31/2020).

[73] Fabian Kuhn, Stefan Schmid, and Roger Wattenhofer, "Towards worst-case churn resistant peer-to-peer systems," en, *Distrib. Comput.*, vol. 22, no. 4, pp. 249–267, May 2010, ISSN: 1432-0452. DOI: 10.1007/s00446-010-0099-z. [Online]. Available: https://doi.org/10.1007/s00446-010-0099-z (visited on 06/01/2021).

[74] J Benet, *Filecoin: A decentralized storage network. protocol labs, 2017*, https://filecoin.io/filecoin.pdf, 2017.

[75] Yongle Chen, Hui Li, Kejiao Li, and Jiyang Zhang, "An improved P2P file system scheme based on IPFS and Blockchain," in *2017 IEEE International Conference on Big Data (Big Data)*, Dec. 2017, pp. 2652–2657. DOI: 10.1109/BigData.2017.8258226.

[76] Mario Schlosser, Michael Sintek, Stefan Decker, and Wolfgang Nejdl, "Hypercup — hypercubes, ontologies, and efficient search on peer-to-peer networks," in *Agents and Peer-to-Peer Computing*, Gianluca Moro and Manolis Koubarakis, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 112–124, ISBN: 978-3-540-45074-0. [Online]. Available: https://link.springer.com/chapter/10.1007/3-540-45074-2_11.

[77] Xing Jin and S-H Gary Chan, "Unstructured peer-to-peer network architectures," in *Handbook of peer-to-peer networking*, Springer, 2010, pp. 117–142. DOI: 10.1007/978-0-387-09751-0_5.

[78] Nima Jafari Navimipour and Farnaz Sharifi Milani, "A comprehensive study of the resource discovery techniques in peer-to-peer networks," *Peer-to-Peer Networking and Applications*, vol. 8, no. 3, pp. 474–492, May 2015, ISSN: 1936-6450. DOI: 10.1007/s12083-014-0271-5. [Online]. Available: https://doi.org/10.1007/s12083-014-0271-5.

[79] Javad Zarrin, Hao Wen Phang, Lakshmi Babu Saheer, and Bahram Zarrin, "Blockchain for decentralization of internet: Prospects, trends, and challenges," *Cluster Computing*, vol. 24, no. 4, pp. 2841–2866, 2021. DOI: 10.1007/s10586-021-03301-8.

[80] David P Anderson, "Boinc: A system for public-resource computing and storage," in *Fifth IEEE/ACM international workshop on grid computing*, IEEE, 2004, pp. 4–10. [Online]. Available: https://boinc.berkeley.edu/grid_paper_04.pdf.

[81] Fang Wang, Yamir Moreno, and Yaoru Sun, "Structure of peer-to-peer social networks," *Phys. Rev. E*, vol. 73, p. 036 123, 3 Mar. 2006. DOI: 10.1103/PhysRevE.73.036123. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.73.036123.

[82] Nazanin Magharei and Reza Rejaie, "Prime: Peer-to-peer receiver-driven mesh-based streaming," *IEEE/ACM Transactions on Networking*, vol. 17, no. 4, pp. 1052–1065, 2009. DOI: 10.1109/TNET.2008.2007434.

[83] Frederik Armknecht, Ghassan O Karame, Avikarsha Mandal, Franck Youssef, and Erik Zenner, "Ripple: Overview and outlook," in *International Conference on Trust and Trustworthy Computing*, Springer, 2015, pp. 163–180. DOI: 10.1007/978-3-319-22846-4_10.

[84] J. Mischke and B. Stiller, “A methodology for the design of distributed search in P2P middleware,” *IEEE Network*, vol. 18, no. 1, pp. 30–37, Jan. 2004, Conference Name: IEEE Network, ISSN: 1558-156X. DOI: 10.1109/MNET.2004.1265831.

[85] Shilpa Budhkar and Venkatesh Tamarapalli, “Two-tier peer selection strategy to minimize delay in p2p live streaming systems,” in *2016 Twenty Second National Conference on Communication (NCC)*, 2016, pp. 1–6. DOI: 10.1109/NCC.2016.7561203.

[86] Vana Kalogeraki, Dimitrios Gunopulos, and D. Zeinalipour-Yazti, “A local search mechanism for peer-to-peer networks,” in *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, ser. CIKM ’02, McLean, Virginia, USA: Association for Computing Machinery, 2002, pp. 300–307, ISBN: 1581134924. DOI: 10.1145/584792.584842. [Online]. Available: https://doi.org/10.1145/584792.584842.

[87] Murat Karakaya, Ibrahim Korpeoglu, and Özgür Ulusoy, “Free riding in peer-to-peer networks,” *IEEE Internet Computing*, vol. 13, no. 2, pp. 92–98, 2009. DOI: 10.1109/MIC.2009.33.

[88] Philippe Golle, Kevin Leyton-Brown, Ilya Mironov, and Mark Lillibridge, “Incentives for Sharing in Peer-to-Peer Networks | SpringerLink,” vol. WELCOM 2001, Springer, Berlin, Heidelberg, Oct. 2001, pp. 75–87. DOI: https://doi.org/10.1007/3-540-45598-1_9. [Online]. Available: https://link.springer.com/chapter/10.1007/3-540-45598-1_9 (visited on 06/10/2021).

[89] Q. Sun and H. Garcia-Molina, “Slic: A selfish link-based incentive mechanism for unstructured peer-to-peer networks,” in *24th International Conference on Distributed Computing Systems, 2004. Proceedings.*, 2004, pp. 506–515. DOI: 10.1109/ICDCS.2004.1281617.

[90] Anna-Lena Wirz, *Media-Streaming und Geoblocking: Eine urheberrechtliche Analyse der Werkverwertung durch On-Demand-Dienste*. Springer Fachmedien Wiesbaden, 2019, ISBN: 9783658260835. DOI: 10.1007/978-3-658-26083-5. [Online]. Available: http://dx.doi.org/10.1007/978-3-658-26083-5.

[91] Hans J. Kleinsteuber, *Radio*. VS Verlag für Sozialwissenschaften, 2012, ISBN: 9783531931005. DOI: 10.1007/978-3-531-93100-5. [Online]. Available: http://dx.doi.org/10.1007/978-3-531-93100-5.

[92] Naviya Dayanand and Amarsinh V. Vidhate, “Routing in delay tolerant network,” 2016. [Online]. Available: https://research.ijcaonline.org/icast2015/number2/icast3013.pdf.

[93] Zupeng Li, Daoying Huang, Zinrang Liu, and Jianhua Huang, “Research of peer-to-peer network architecture,” in *International Conference on Communication Technology Proceedings, 2003. ICCT 2003.*, IEEE, vol. 1, 2003, pp. 312–315. DOI: 10.1109/ICCT.2003.1209091.

[94] ”Bitcoin wiki, *Network - bitcoin wiki*, Online available: https://en.bitcoin.it/wiki/Network, Last-Access 1/15/2024., Jun. 2018.

[95] Giulia Fanti and Pramod Viswanath, “Deanonymization in the bitcoin p2p network,” in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 1364–1373. [Online]. Available: https://papers.nips.cc/paper_files/paper/2017/file/6c3cf77d52820cd0fe646d38bc2145ca-Paper.pdf.

[96] Dimitris Vyzoviti, Yusef Napora, Dirk McCormick, David Dias, and Yiannis Psaras, *Gossipsub-v1.1 evaluation report*, https://blog.ipfs.io/gossipsubv1.1-eval-report-and-security-audit/, 2020.

[97] Ajay Arunachalam and Ohm Sornil, “Issues of implementing random walk and gossip based resource discovery protocols in p2p manets & suggestions for improvement,” *Procedia Computer Science*, vol. 57, pp. 509–518, 2015, 3rd International Conference on Recent Trends in Computing 2015 (ICRTC-2015), ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2015.07.374. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050915019031.

[98] F.M. Cuenca-Acuna, C. Peery, R.P. Martin, and T.D. Nguyen, “PlanetP: Using gossiping to build content addressable peer-to-peer information sharing communities,” in *12th IEEE International Symposium on High Performance Distributed Computing, 2003. Proceedings*, ISSN: 1082-8907, Jun. 2003, pp. 236–246. DOI: 10.1109/HPDC.2003.1210033.

[99] C. Gkantsidis, M. Mihail, and A. Saberi, “Random walks in peer-to-peer networks,” in *IEEE INFOCOM 2004*, vol. 1, 2004, p. 130. DOI: 10.1109/INFCOM.2004.1354487.

[100] Daniel Stutzbach and Reza Rejaie, “Towards a better understanding of churn in peer-to-peer networks,” *Univ. of Oregon, Tech. Rep*, 2004. [Online]. Available: https://mirage.cs.uoregon.edu/pub/tr04-06.pdf.

[101] Daniel Stutzbach and Reza Rejaie, "Understanding churn in peer-to-peer networks," in *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '06, Rio de Janeriro, Brazil: Association for Computing Machinery, 2006, pp. 189–202, ISBN: 1595935614. DOI: 10.1145/1177080.1177105. [Online]. Available: https://doi.org/10.1145/1177080.1177105.

[102] Xianfu Meng, "A churn-aware durable data storage scheme in hybrid P2P networks," en, *J Supercomput*, vol. 74, no. 1, pp. 183–204, Jan. 2018, ISSN: 1573-0484. DOI: 10.1007/s11227-017-2125-4. [Online]. Available: https://doi.org/10.1007/s11227-017-2125-4 (visited on 06/01/2021).

[103] Katsuya Suto, Hiroki Nishiyama, Nei Kato, Takayuki Nakachi, Tatsuya Fujii, and Atsushi Takahara, "Thup: A p2p network robust to churn and dos attack based on bimodal degree distribution," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 9, pp. 247–256, 2013. DOI: 10.1109/JSAC.2013.SUP.0513022.

[104] Yong Liu, Yang Guo, and Chao Liang, "A survey on peer-to-peer video streaming systems," *Peer-to-peer Networking and Applications*, vol. 1, no. 1, pp. 18–28, 2008. DOI: 10.1007/s12083-007-0006-y.

[105] Lucianna Kiffer, Asad Salman, Dave Levin, Alan Mislove, and Cristina Nita-Rotaru, "Under the hood of the ethereum gossip protocol," *Proceedings of the Financial Cryptography and Data Security (FC'21). St. George's, Grenada*, 2021. [Online]. Available: https://www.ccs.neu.edu/home/amislove/publications/Ethereum-FC.pdf.

[106] Muhammad Anas Imtiaz, David Starobinski, Ari Trachtenberg, and Nabeel Younis, "Churn in the bitcoin network," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2021. DOI: 10.1109/TNSM.2021.3050428.

[107] Marcos Pinheiro, Silvio Sampaio, Francisco Vasques, and Pedro Souto, "A dht-based approach for path selection and message forwarding in ieee 802.11s industrial wireless mesh networks," in *2009 IEEE Conference on Emerging Technologies Factory Automation*, 2009, pp. 1–10. DOI: 10.1109/ETFA.2009.5347111.

[108] Kaimin Wei, Mianxiong Dong, Kaoru Ota, and Ke Xu, "Camf: Context-aware message forwarding in mobile social networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 8, pp. 2178–2187, 2015. DOI: 10.1109/TPDS.2014.2346212.

[109] Armel Esnault, Nicolas Le Sommer, and Frédéric Guidec, "A peer-to-peer overlay system for message delivery in wide intermittently-connected hybrid networks," in *Wired/Wireless Internet Communications*, Abdelhamid Mellouk, Scott Fowler, Saïd Hoceini, and Boubaker Daachi, Eds., Cham: Springer International Publishing, 2014, pp. 200–213, ISBN: 978-3-319-13174-0. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-13174-0_16.

[110] Matt Holdrege and Pyda Srisuresh, *IP Network Address Translator (NAT) Terminology and Considerations*, RFC 2663, Aug. 1999. DOI: 10.17487/RFC2663. [Online]. Available: https://www.rfc-editor.org/info/rfc2663.

[111] Philipp Richter, Mark Allman, Randy Bush, and Vern Paxson, "A primer on ipv4 scarcity," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 2, pp. 21–31, Apr. 2015, ISSN: 0146-4833. DOI: 10.1145/2766330.2766335. [Online]. Available: https://doi.org/10.1145/2766330.2766335.

[112] Randy Bush *et al.*, "The address plus port (a+ p) approach to the ipv4 address shortage," RFC 6346, August, Tech. Rep., 2011. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6346.html.

[113] Arno Wacker, Gregor Schiele, Sebastian Holzapfel, and Torben Weis, "A nat traversal mechanism for peer-to-peer networks," in *2008 Eighth International Conference on Peer-to-Peer Computing*, 2008, pp. 81–83. DOI: 10.1109/P2P.2008.29.

[114] Marc Petit-Huguenin, Gonzalo Salgueiro, Jonathan Rosenberg, Dan Wing, Rohan Mahy, and Philip Matthews, *Session Traversal Utilities for NAT (STUN)*, RFC 8489, Feb. 2020. DOI: 10.17487/RFC8489. [Online]. Available: https://www.rfc-editor.org/info/rfc8489.

[115] Simon Keller, Tobias Hoßfeld, and Sebastian von Mammen, "Edge-case integration into established nat traversal techniques," in *2022 IEEE Ninth International Conference on Communications and Electronics (ICCE)*, 2022, pp. 75–80. DOI: 10.1109/ICCE55644.2022.9852092.

[116] Prashanth Patil, Tirumaleswar Reddy.K, and Dan Wing, *Traversal Using Relays around NAT (TURN) Server Auto Discovery*, RFC 8155, Apr. 2017. DOI: 10.17487/RFC8155. [Online]. Available: https://www.rfc-editor.org/info/rfc8155.

[117] Elie Kfoury and David Khoury, “Securing natted iot devices using ethereum blockchain and distributed turn servers,” in *2018 10th International Conference on Advanced Infocomm Technology (ICAIT)*, 2018, pp. 115–121. DOI: 10.1109/ICAIT.2018.8686623.

[118] Ari Keränen, Christer Holmberg, and Jonathan Rosenberg, *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal*, RFC 8445, Jul. 2018. DOI: 10.17487/RFC8445. [Online]. Available: https://www.rfc-editor.org/info/rfc8445.

[119] Bryan Ford, Dan Kegel, and Pyda Srisuresh, *State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs)*, RFC 5128, Mar. 2008. DOI: 10.17487/RFC5128. [Online]. Available: https://www.rfc-editor.org/info/rfc5128.

[120] Bryan Ford, Pyda Srisuresh, and Dan Kegel, “Peer-to-peer communication across network address translators,” Apr. 2006. DOI: 10.48550/ARXIV.CS/0603074. [Online]. Available: https://www.usenix.org/legacy/event/usenix05/tech/general/full_papers/ford/ford.pdf.

[121] Jochen Dinger and Oliver P. Waldhorst, “Decentralized Bootstrapping of P2P Systems: A Practical View,” en, in *NETWORKING 2009*, Luigi Fratta, Henning Schulzrinne, Yutaka Takahashi, and Otto Spaniol, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2009, pp. 703–715, ISBN: 978-3-642-01399-7. DOI: 10.1007/978-3-642-01399-7_55.

[122] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron, “One ring to rule them all: Service discovery and binding in structured peer-to-peer overlay networks,” in *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop*, ser. EW 10, Saint-Emilion, France: Association for Computing Machinery, 2002, pp. 140–145, ISBN: 9781450378062. DOI: 10.1145/1133373.1133399. [Online]. Available: https://doi.org/10.1145/1133373.1133399.

[123] Michael Conrad and Hans-Joachim Hof, “A Generic, Self-organizing, and Distributed Bootstrap Service for Peer-to-Peer Networks,” en, in *Self-Organizing Systems*, David Hutchison and Randy H. Katz, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2007, pp. 59–72, ISBN: 978-3-540-74917-2. DOI: 10.1007/978-3-540-74917-2_7.

[124] Chris Gauthier Dickey and Christian Grothoff, “Bootstrapping of peer-to-peer networks,” in *2008 International Symposium on Applications and the Internet*, 2008, pp. 205–208. DOI: 10.1109/SAINT.2008.15.

[125] Mirko Knoll, Arno Wacker, Gregor Schiele, and Torben Weis, “Bootstrapping in peer-to-peer systems,” in *2008 14th IEEE International Conference on Parallel and Distributed Systems*, 2008, pp. 271–278. DOI: 10.1109/ICPADS.2008.26.

[126] Simone Cirani and Luca Veltri, “A multicast-based bootstrap mechanism for self-organizing p2p networks,” in *GLOBECOM 2009 - 2009 IEEE Global Telecommunications Conference*, 2009, pp. 1–6. DOI: 10.1109/GLOCOM.2009.5426215.

[127] Pradnya Karbhari, Mostafa Ammar, Amogh Dhamdhere, Himanshu Raj, George F. Riley, and Ellen Zegura, “Bootstrapping in Gnutella: A Measurement Study,” en, in *Passive and Active Network Measurement*, Chadi Barakat and Ian Pratt, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2004, pp. 22–32, ISBN: 978-3-540-24668-8. DOI: 10.1007/978-3-540-24668-8_3.

[128] Lukas König, Stefan Unger, Peter Kieseberg, and Simon Tjoa, “The Risks of the Blockchain A Review on Current Vulnerabilities and Attacks,” English, *Journal of Internet Services and Information Security (JISIS), volume: 10, number: 3*, no. Volume 10, pp. 110–127, 2020, Projekt: Blockchain Security, ISSN: 2182-2077. DOI: 10.22667/JISIS.2020.08.31.110. [Online]. Available: https://isyou.info/jisis/vol10/no3/jisis-2020-vol10-no3-06.pdf.

[129] John Douceur, “The sybil attack,” in *Peer-to-Peer Systems*, ser. Lecture Notes in Computer Science, Jan. 2002, pp. 251–260, ISBN: 978-3-540-44179-3. DOI: 10.1007/3-540-45748-8_24.

[130] Prakash Linga, Indranil Gupta, and Ken Birman, “A churn-resistant peer-to-peer web caching system,” in *Proceedings of the 2003 ACM Workshop on Survivable and Self-Regenerative Systems: In Association with 10th ACM Conference on Computer and Communications Security*, ser. SSRS ’03, Fairfax, VA: Association for Computing Machinery, 2003, pp. 1–10, ISBN: 1581137842. DOI: 10.1145/1036921.1036922. [Online]. Available: https://doi.org/10.1145/1036921.1036922.

[131] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker, "Search and replication in unstructured peer-to-peer networks," in *Proceedings of the 16th International Conference on Supercomputing*, ser. ICS '02, New York, New York, USA: Association for Computing Machinery, 2002, pp. 84–95, ISBN: 1581134835. DOI: 10.1145/514191.514206. [Online]. Available: https://doi.org/10.1145/514191.514206.

[132] "Broadcasting in unstructured peer-to-peer overlay networks," en, *Theoretical Computer Science*, vol. 355, no. 1, pp. 25–36, Apr. 2006, Publisher: Elsevier, ISSN: 0304-3975. DOI: 10.1016/j.tcs.2005.12.013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304397505009138 (visited on 09/08/2020).

[133] Steve Bellovin, "Security aspects of napster and gnutella," in *2001 Usenix Annual Technical Conference*, 2001. [Online]. Available: https://www.usenix.org/legacy/publications/library/proceedings/usenix01/invitedtalks/bellovin.pdf.

[134] John Risson and Tim Moors, "Survey of research towards robust peer-to-peer networks: Search methods," *Comput. Netw.*, vol. 50, no. 17, pp. 3485–3521, Dec. 2006, ISSN: 1389-1286. DOI: 10.1016/j.comnet.2006.02.001. [Online]. Available: https://doi.org/10.1016/j.comnet.2006.02.001.

[135] Lacine KABRE and Telesphore Tiendrebeogo, "Comparative study of can, pastry, kademlia and chord dhts," *International Journal of Peer to Peer Networks*, vol. 12, pp. 1–22, Aug. 2021. DOI: 10.5121/ijp2p.2021.12301.

[136] Fabrizio Falchi, Claudio Gennaro, and Pavel Zezula, "A content–addressable network for similarity search in metric spaces," Aug. 2005, pp. 98–110, ISBN: 978-3-540-71660-0. DOI: 10.1007/978-3-540-71661-7_9.

[137] Yunqi Zhang and Shaileshh Venkatakrishnan, "Kadabra: Adapting kademlia for the decentralized web," in Dec. 2023, pp. 327–345, ISBN: 978-3-031-47750-8. DOI: 10.1007/978-3-031-47751-5_19.

[138] Simone Cirani, Natalya Fedotova, and Luca Veltri, "A resilient architecture for dht-based distributed collaborative environments," in *Proceedings of the 2008 RISE/EFTS Joint International Workshop on Software Engineering for Resilient Systems*, ser. SERENE '08, Newcastle upon Tyne, United Kingdom: Association for Computing Machinery, 2008, pp. 1–8, ISBN: 9781605582757. DOI: 10.1145/1479772.1479774. [Online]. Available: https://doi.org/10.1145/1479772.1479774.

[139] *Libp2p*, en-us. [Online]. Available: https://docs.libp2p.io/concepts/introduction/overview/ (visited on 09/12/2023).

[140] *Libp2p & Ethereum (the Merge) | libp2p Blog & News*. [Online]. Available: https://blog.libp2p.io/libp2p-and-ethereum/#the-merge (visited on 09/12/2023).

[141] Jean-Philippe Aumasson, Denis Kolegov, and Evangelia Stathopoulou, "Security review of ethereum beacon clients," *arXiv preprint arXiv:2109.11677*, 2021.

[142] *Specs/pubsub/gossipsub at master · libp2p/specs*, en. [Online]. Available: https://github.com/libp2p/specs/tree/master/pubsub/gossipsub (visited on 09/12/2023).

[143] Barbara Guidi, Andrea Michienzi, and Laura Ricci, "A libp2p implementation of the bitcoin block exchange protocol," in *Proceedings of the 2nd International Workshop on Distributed Infrastructure for Common Good*, ser. DICG'21, Virtual Event, Canada: Association for Computing Machinery, 2021, pp. 1–4, ISBN: 9781450391696. DOI: 10.1145/3493426.3493822. [Online]. Available: https://doi.org/10.1145/3493426.3493822.

[144] David Lee Chaum, *Computer Systems established, maintained and trusted by mutually suspicious groups*. Electronics Research Laboratory, University of California, 1979. [Online]. Available: https://chaum.com/wp-content/uploads/2022/02/techrep.pdf.

[145] Alan T. Sherman, Farid Javani, Haibin Zhang, and Enis Golaszewski, "On the Origins and Variations of Blockchain Technologies," *IEEE Security Privacy*, vol. 17, no. 1, pp. 72–77, Jan. 2019, Conference Name: IEEE Security Privacy, ISSN: 1558-4046. DOI: 10.1109/MSEC.2019.2893730.

[146] Christian Stoll, Lena Klaaßen, and Ulrich Gallersdörfer, "The Carbon Footprint of Bitcoin," en, *Joule*, vol. 3, no. 7, pp. 1647–1661, Jul. 2019, ISSN: 2542-4351. DOI: 10.1016/j.joule.2019.05.012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2542435119302557 (visited on 05/03/2021).

[147] ethereum.org, *Ethereum.org*. [Online]. Available: https://ethereum.org/ (visited on 12/10/2023).

[148] Alessia Antelmi, Giuseppe D'Ambrosio, Andrea Petta, Luigi Serra, and Carmine Spagnuolo, "A volunteer computing architecture for computational workflows on decentralized web," *IEEE Access*, vol. 10, pp. 98 993–99 010, 2022. DOI: 10.1109/ACCESS.2022.3207167.

[149] ethereum.org team. “The great renaming: What happened to eth2?” (Jan. 2022), [Online]. Available: https://blog.ethereum.org/en/2022/01/24/the-great-eth2-renaming (visited on 01/11/2024).

[150] Aniket Paul, “Assessing the environmental sustainability of polygons consensus mechanism and transaction processing, comparing its energy consumption and carbon footprint with other layer 2 and layer 1 blockchain solutions, and exploring potential avenues for further optimization,” *International Journal for Research in Applied Science and Engineering Technology*, vol. 11, no. 6, pp. 1497–1507, Jun. 30, 2023, ISSN: 23219653. DOI: 10.22214/ijraset.2023.53900. [Online]. Available: https://www.ijraset.com/best-journal/assessing-the-environmental-sustainability-of-polygons-consensus-mechanism-and-transaction-processing (visited on 01/10/2024).

[151] Zhiqiang Ouyang, Jie Shao, and Yifeng Zeng, “Pow and pos and related applications,” in *2021 International Conference on Electronic Information Engineering and Computer Science (EIECS)*, 2021, pp. 59–62. DOI: 10.1109/EIECS53707.2021.9588080.

[152] Changqiang Zhang, Cangshuai Wu, and Xinyi Wang, “Overview of blockchain consensus mechanism,” in *Proceedings of the 2020 2nd International Conference on Big Data Engineering*, ser. BDE 2020, Shanghai, China: Association for Computing Machinery, 2020, pp. 7–12, ISBN: 9781450377225. DOI: 10.1145/3404512.3404522. [Online]. Available: https://doi.org/10.1145/3404512.3404522.

[153] Cloudflare Inc., *What is http/3?* https://www.cloudflare.com/learning/performance/what-is-http3/, Accessed: (11-02-2024), 2024.

[154] comScore; VuMA; Bitkom Research, *Anzahl der smartphone-nutzer* in deutschland in den jahren 2009 bis 2021 (in millionen)*, https://de.statista.com/statistik/daten/studie/198959/umfrage/anzahl-der-smartphonenutzer-in-deutschland-seit-2010/, Accessed: 2023-01-11, Nov. 2021.

[155] Tevora Threat Research Group, “2021 hyperledger fabric penetration test for linux foundation,” Tevora Threat Research Group, Tech. Rep., Mar. 2021. [Online]. Available: https://wiki.hyperledger.org/download/attachments/13861997/2021%20HyperLedger%20Fabric%20Penetration%20Test%20v1.1.pdf?version=1&modificationDate=1621520080000&api=v2.

[156] Salih Ismail, Hani Ragab Hassen, Mike Just, and Hind Zantout, “A review of amplification-based distributed denial of service attacks and their mitigation,” *Computers & Security*, vol. 109, p. 102 380, 2021, ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2021.102380. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404821002042.

[157] “The sixth ieee workshop on language-theoretic security (langsec 2020),” in *2020 IEEE Security and Privacy Workshops (SPW)*, 2020, pp. xxviii–xxix. DOI: 10.1109/SPW50608.2020.00016. [Online]. Available: https://ieeexplore.ieee.org/document/9283858.

[158] Ross Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, 2001, ISBN: 0-471-38922-6.