VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
STUDY PROGRAM: INFORMATICS

Data storage using blockchain
Duomenų saugojimas blockchain

Master's thesis

Author: Simonas Čėsna
VU E-mail.: simonas.cesna@mif.stud.vu.lt

Supervisor: j. assist. dr. Igor Katin
Reviewer: assist. dr. Adomas Birštunas

Vilnius
2025

TABLE OF CONTENTS

SUMMARY

In this final master's thesis, the main task was to research the blockchain file-sharing technology and, in turn, to address the more common challenges that occur while using it, such as access control, data availability, and security. The project focuses on gathering and proposing solutions for different issues that arise with blockchain file systems, whilst putting all of them into a new single working blockchain system. It delves into concepts such as permission management, parallel downloading, mining, and file splitting, with the final goal being to test their overall usability and applicability within a blockchain system and study the issues or improvements that can be made further on.

Keywords: blockchain, file-sharing, access control

SANTRAUKA

Šiame magistro baigiamajame darbe yra tiriama blokų grandinių technologija, skirta rinkmenų apsikeitimui, bei iškeliami pagrindiniai iššūkiai, kurie kyla ja naudojantis. Tarp šių iššūkių yra tokie kaip: prieigos valdymas, duomenų prieinamumas, sauga. Projektas susitelkia ties dažniausiomis problemomis naudojantis šia technologija, bei sprendimų paieška ir jų pritaikymu į vieną bendrą rinkmenų apsikeitimų sistemą. Šiame projekte apžvelgiamos idėjos ir sprendimai, tokie kaip teisių valdymas, lygiagretus rinkmenų siuntimas, kasimo algoritmas, bei rinkmenų padalijimas. Galutinis darbo tikslas susitelkia ties šių sprendimų pritaikymu ir bandymais bendroje blokų grandinių sistemoje tam, jog būtų galima suprasti iškeltų sprendimų pritaikomumą, kylančius trūkumus, bei galimus patobulinimus.

Raktiniai žodžiai: blokų grandinės, rinkmenų apsikeitimas, prieigos valdymas

1. INTRODUCTION

Blockchain is a complicated, yet widely used technology in the modern-day internet space. It is a secure system for recording and sharing information where each user holds a copy of the records. The records can only be updated if all parties involved in a transaction agree to update them. [BP19] One of the main widely used concepts that is powered by this technology is cryptocurrency. However, there is a wider variety of concepts that can be used in this way of data exchange, such as file transfer. Blockchain is suitable because one of the main blockchain mechanisms is trusted information exchange and reliable storage that is shared between all parts of the system. The research objective of this project is to propose and analyze a design of a blockchain data storage system, which would allow secure file exchange between clients with the ability to change certain file permissions of the entire network.

2. PROBLEMS

The blockchain technology provides the ability to quickly and reliably have a shared database (ledger) that can be accessed from every node and has the newest transaction data stored in separate blocks. This technology can be used for various applications of data storage, including serving entire file systems. However, in comparison to traditional file-sharing applications, blockchain file system technologies can quickly encounter multiple problems that are more pronounced in distributed networks. Even the newest projects that use blockchain for file transfer often face various problems, which some of the more common are:

1. Continuously increasing file size on a single network.
2. Old, expired, or redundant files.
3. Lower download speeds and/or file reachability.
4. Malicious activity on the network, such as spam attacks.
5. Lack of permission management for network moderation.

The first such issue is file size. Having to store each file of different types ever added into the system can become quite costly in terms of storage space and network speeds [KYZ22]. In some cases (such as decentralized data storage system “Sia”), it can lead to a concerning size expansion of the blockchain network [Aus20]. One of the ways to solve this problem is to store only a certain percentage of files on each node that would be responsible for file storage. This would cut the storage requirements for each node; however, it can negatively impact redundancy and file availability (in such cases as connecting to a node that does not have such a file saved, or the nodes containing the file suddenly going offline). One way to try and reduce this negative

impact would be to separate the encrypted file into smaller bits stored on different nodes, with additional shared distributed hash table that keeps track of the required split bits [Nai22]. Another way to cut down the storage space would be to add file type limits, which would only allow document and photo exchange of certain types and size limits. This would limit the usage of the application, but it would also make it more tailored towards a specific use, such as a private or semi-private document (or screenshot) sharing network.

An additional issue can arise with some files that can become old and unusable. That's why another way of file size and traffic control is to add a possible expiration date setting for files in order to clean up the ones that are old and not likely to be accessed anymore.

Furthermore, a lack in download speed can become a noticeable barrier in some blockchain systems, due to slower update rates, as well as lack of reachability due to certain concerns, such as some nodes not hosting the file needed, or even basic internet latency, that can be caused by either congested networks or long physical distances apart from different nodes.

Last but not least, some network users may need to have their access limited due to reasons such as network abuse (e.g., DDOS attacks, file spam) or breach of preset terms of the blockchain system's service. This problem can be addressed by limiting access using file permissions, because it can give the power to the network's owner to prevent some sorts of abuse of the terms of service that can be detrimental to a file-sharing system. However, it is important to ensure that the requester is allowed to receive a service or to perform a system-specific action. [Elr19]

Using all the mentioned solutions, the issue of file permission control still exists. Sometimes, especially in private networks, there is a need to change certain attributes (such as file types, allowed users, size limits, date limits, etc.). One way to solve this problem is to use a shared global file that defines file attributes and permissions, and is controlled by the network's owner. The main purpose of the file is to adapt the attributes quickly throughout each node of the network. Several ways to implement this solution would be to either share a signed attribute file, that defines certain values, such as permissions, or to store the attributes directly in the ledger involving file data, that is shared between clients.

3. THE GOAL

The main goal of this project is to propose multiple solutions that can be adapted to a blockchain file-sharing system, which in turn would solve the more common problems with the blockchain adaptation for file systems: continuously increasing usage of space and lack of access control. These solutions would then be integrated into a full system design of a secure file-sharing system that leverages blockchain technology and ensures robust security and transparency. One of the main focuses will be to develop an algorithm that would update the system's global network file permissions, providing dynamic control over user access rights and file limits.

In order to test the viability of the design, a prototype application will be created in order to evaluate the proposed solutions using practical tests consisting of mainly performance evaluation through speed and reliability analysis in simulated networks.

4. TASKS

1. Analyze currently relevant blockchain studies and applications.
2. Analyze and compare projects that use blockchain technology for file-sharing.
3. Create an algorithm to reliably exchange file permissions throughout nodes.
4. Propose a solution for saving storage space for each node.
5. Create a load-balancing algorithm to increase node file transfer speeds.
6. Design a blockchain file exchange system based on the created algorithms.
7. Create a test suite to and analyze separate parts of the system involving the created algorithms.

To achieve the set project goal, multiple tasks were defined. In the beginning, the first task will be to gain knowledge that is relevant to the topic. A thorough analysis of current blockchain studies and applications will be conducted in order to gain a solid background understanding of the concept and overall field. After learning about possible implementations and pitfalls involving blockchain, a comparison of available blockchain technology projects will be done in order to identify the most promising solutions. Then, file attribute exchange and load-balancing algorithms will be created or selected from currently available implementations. Afterwards, using the algorithms involved, an architectural design of a blockchain file-sharing system will be created, describing the network's functionality in detail.

After the creation of the design, a prototype blockchain file-sharing application will be created in order to test the design's real-life application scenarios. A suitable test suite will be designed for the application, in order to test it out in a simulated larger-scale network, consisting

of multiple nodes. During the testing phases, the application's speed and stability will be measured and analyzed to ensure optimal performance and reliability. This approach will be used to test the viability and usability of the algorithms involving the system's main design.

5. HYPOTHESES

1. Data access and storage on blockchain systems can be reliably controlled by a privileged user in a predictable schedule.
2. File transfer load-balancing solutions, such as parallel downloading from multiple sources, can help achieve a speed increase on a blockchain file system.
3. Data storage can be partially split along different nodes without a considerable loss to availability or file lookup performance

6. EXPECTED RESULTS

In order to prove the hypotheses, the following results need to be reached in the research work:

1. A mining algorithm whose working time is determined by preset difficulty to control the amount of time a new block is created in a blockchain network of different sizes.
2. An algorithm for reliably updating the system's file permissions.
3. A solution for saving file space for each different node.
4. A load balancing algorithm to download data from multiple nodes simultaneously.
5. Analysis of the proposed solutions on a simulated blockchain network.

To be able to analyze the proposed results, a system is expected to be created that would consist of multiple interconnected modules incorporating functionality as defined in the results. These modules would then be analyzed separately and in a fully working blockchain system in order to check their viability, uses, and overall performance in a blockchain network.

7. METHODOLOGY

7.1. The blockchain

At the current time, blockchain is an evolving technology, that is constantly being improved upon in some shape or form with each application that utilizes it. The main idea of the blockchain is to provide a shared database (also known as a ledger) in a decentralized and immutable manner. It allows reliable data management with integrity and removes any chance of centralized failure. [ZZ16] Blockchain provides a trusted way of sharing data between different participants of the network (also known as nodes). The data relevant to the system using the blockchain is kept on the ledger and the changes that are accepted are the only ones that are trusted by the majority of the network. This sort of trust is laid out by the concept of smart contracts. These are a set of preset rules that are programmed to run automatically under certain conditions (ex. incoming transactions requiring a new block) and involves key signing and hashing concepts that are later used to validate and confirm changes.

Each change to the blockchain is grouped into sets, otherwise known as “blocks” (hence where the term “blockchain” comes from). These blocks stay in a sequence and are accepted only if validated by the preset consensus algorithm, which often involves one or several different network nodes in a process that is known as mining.

The blockchain consists of a network that is a chain of multiple computers hosting a service that relies on the technology. The network can be private – a limited set of nodes connected using authentication methods such as login tokens, or the entire network being set in an intranet, which is only accessible to selected devices. It can also be completely public, and the service might not need any sort of authentication at all, besides a public-private key pair to confirm changes.

There are many different uses of blockchain for various needs of people, but the most common of them are:

- **Cryptocurrencies.** Most cryptocurrencies (ex., Bitcoin) use blockchain to perform transactions without the forced control of third parties such as banks or governments.
- **Data storage.** Blockchain provides a stored database (ledger), which can be used for secure and immutable data storage of various needs.
- **File storage.** It allows for decentralized, (possibly) anonymous, and redundant implementations of file storage.
- **Non-fungible tokens (NFT).** They are used as a unique certificate of authenticity or as proof of ownership of different virtual items that they have a reference to.

- **Other means of secure data transfer.** Many different options, ranging from Internet of Things (IoT) appliances to medical and automotive vehicle applications, can pose a requirement for a more reachable and privacy-focused solution for data transfer in comparison to the more commonly available options.

While there are many uses for blockchain technology, file-sharing is the one that's necessary for the project. Considering that there are already widely used implementations (such as IPFS), the possibilities that blockchain offers makes it viable to be used in order to share files in a similar fashion as other data is shared, just with a few additional considerations (ex. file size).

7.2. Node discovery

For every decentralized system concept, a common point to define at the beginning is the way nodes will communicate with each other. While network and connection protocols may differ, the way of discovering nodes to connect to in a decentralized system is the first operation the application makes. In order to be able to interface with the blockchain network, the client needs to be able to connect to at least one of the nodes that make up the network. However, the same rule goes for all the other nodes in the network – they all need a way to be able to discover each other in order to exchange information to keep updated with the latest changes. So, to find the node(s) from the network to connect to, some consideration must be made.

In many blockchain implementations (ex., Bitcoin), there are predefined lists of node addresses that are relied upon to be constantly operating in order to provide initial data to the clients, who connect to them. When a node connects to another one, one of the parts of data they share is the addresses of the clients that were previously connected/discovered. This way, after initial connection, the client does not necessarily need to connect to the initial predefined nodes, but rather, perform a lookup on their cached node list in order to find the ones more suitable based on network latency, location, and availability. In file-sharing applications, the availability of files is also considered, because some networks don't save the same files on every single node in order to save space and allow for bigger theoretical file size limits.

While creating a file-sharing application using blockchain, the requested file lookup algorithm must be considered. Theoretically, one client could perform a naïve search that would look in every node from the cached node list until finding the first one that contains the requested data is found, however this sort of search would be quite expensive in terms of bandwidth and time, especially on big networks where the search time would grow exponentially due to the

considerable number of nodes involved. Also, the resulting node may not be as suitable in terms of connection speed.

To effectively find out the best suitable nodes that contain the requested data, a system known as a distributed hash table (DHT) may be used to considerably increase lookup speed in comparison to a naïve search. This method first appeared in peer-to-peer systems, and is still being widely used in applications such as BitTorrent. It is defined as a distributed data structure that can be used to store and retrieve key pairs. "DHT stores resources on different nodes through a predetermined agreement, and uses unique keys to identify specific resources". [CZS22]

By definition, the DHT system is a derivative of hash tables. A hash is generated by a predefined hash function (ex. SHA256) every time a value needs to be placed inside the distributed table and the pair is added. This way it's faster to find the data needed instead of looking through all options, although more memory is used due to every entry needing its own hash. This sort of table is shared and synchronized throughout all the clients, or in blockchain's case – it's nodes. What is put in key-value pairs is different, depending on use basis, but more common cases involve the assignation of a unique ID to a node, which is then used as a value. As for the key, in file-sharing applications it may be picked as the hash of the file data (or metadata), or a unique identifier (ex. CID in IPFS applications) of the file. Using this pair, whenever a client is requesting a file, a node that contains the data can be found quickly just by taking the value (the corresponding node) of the distributed hash table with the given file hash as the key. The value can also be a list of nodes, if there is more than one node involved in saving the file.

7.3. Mining

Every change on the blockchain applications needs to be verified and confirmed in order to keep the network trusted and immutable to unauthorized tampering. This is done by using a concept known as mining. Some of the network's nodes can opt in to calculate cryptographic challenges in order to validate a set of changes (a.k.a. transactions) done to the ledger, the latter of which are known as "blocks". During the process of mining, every miner node performs various cryptographic calculations (such as signature verification using a public key) in order to verify the validity of the unconfirmed changes made. These changes are later bundled by the mining software (usually prioritizing the ones that yield the highest reward if one is present, or by oldest timestamp if not). After bundling the changes into a block, it has to be confirmed using a consensus algorithm in order for the block to be accepted by the rest of the blockchain. One of such algorithms is known as "Proof of work".

The “Proof of Work” algorithm is a widely used consensus algorithm for mining blocks. The core idea is to solve a complex mathematical puzzle into a simple hash solution that can quickly be validated. The puzzles may vary, but they usually include the hash of the block in order for other nodes to be able to quickly confirm and verify the data contents alongside the puzzle’s result. Such challenges are intentionally time consuming and difficult to solve in order to minimize situations where multiple miners confirm a block simultaneously (such an event is known as “temporary fork”). A puzzle can differ but a commonly used method in Bitcoin implementation would be to add a number to the block, which must be set in such a way that the hash of the entire block is smaller than a known target. [Vuk16] In algorithmic terms: find a number X where X added to the block’s data D would be smaller than a number Y ($H(D \cup X) < Y$). Another example could involve a variable such as a difficulty level and simply try to find a number with which once appended a hash value of the block starts with multiple (x difficulty levels) repeating numbers (for example a hash value starting with 000 if the difficulty level is 3). Because such tasks require hash algorithms, the results of hash functions are varied and not much predictable (ex. 2^256 possible results using SHA256). This is why the initial result number is solved usually just by using random generation until it suits the puzzle’s preset task. For example, following the first puzzle algorithm example, the mining node would generate X until it would find such an X that ($H(D \cup X) < Y$) would be a true result. After X was successfully found, the result (sometimes known as “nonce” value) is appended to the data of the block (which likely consists of: block data that contains the set of transactions, hash of previous block to keep the chain linked similarly as in linked lists, timestamp of the block creation, and finally – the nonce value). This data is then broadcasted to other nodes and if it’s confirmed to be valid and first to be mined – the block is added to the chain and its changes are accepted. “The system also defines a fixed block frequency which determines the average amount of time the winning node will take to find such a nonce” [LDC+17]. If this time (also sometimes known as “block time”) passes, and no nonce number is found, the miner updates its database with the new block’s data and restarts the mining process using not previously included (and newly received) transactions.

In some cases, like cryptocurrencies, mining can also yield reward money in case of a successfully solved challenge (also known as “proof of work”). This is done to promote volunteered mining in decentralized systems and therefore, transactions with higher transaction fees may be taken as a priority by the miner in order to possibly yield a higher income. This sort of transaction sorting results in uneven acceptance in changes (ex. those that pay more in fees get their transaction passed faster than those that do not.). However, it also ensures that owners of mining nodes get compensated for their computing power. The randomness of the consensus algorithm also makes such process fairer, in comparison to having ex. many computers or quicker

connection in the network because every mining node has a chance to be the one that is selected. This chance is still improved with more computing power due to more numbers and their hashes generated in a limited time span.

In theory, validating transactions that are put into blocks is an optional procedure for miners, however, it has become de-facto standard due to high possibility of rejection of mined blocks by the network (if at least a single invalid transaction is present in the block), due to the fact that all the other nodes are required to verify the changes themselves before accepting the newly mined block. This causes more instances of wasted work, time, energy, and possibly money, if mining is done with a monetary reward in mind.

During mining, sometimes two or more miners can find a block puzzle at the same or very similar time for the same block. This is an event known as a “temporary fork”, and it needs another consensus algorithm for the nodes to be able to pick only one block for the time. This is done either by comparing how many changes are taken into both blocks (a.k.a. “block height”) and the bigger one is kept. On the event that both blocks are of same sizes, different preset consensus algorithm can be used, for example, select the block which has the lowest result (nonce) value.

7.4. File transfer

Considering that file transfer on the blockchain systems can be done with looking up a single node that contains the file, a peer-to-peer style approach could be assigned to transferring files in order to increase download speeds and save bandwidth for the file hosting nodes. Given that there is more than one active node containing the same file, and there is a quick way to find out all of them (ex. DHT algorithm), a load-balancing algorithm, similar to the one done in BitTorrent protocols could be done. In this algorithm, the file size is split into small pieces of a pre-defined size. The size can also be varied (ex. smaller pieces due to slower network). When a piece size is decided, it can be requested off a node instead of a full file. The following pieces can be simultaneously requested from different nodes. The pieces of the file are downloaded randomly and the client then rearranges them into the correct order. [VJP+22] These steps are then repeated, until the entire file pieces are downloaded. This sort of approach could help speed up the transmission, especially from nodes that have more limited bandwidth speeds, in comparison to the clients.

This approach could further be improved by another load balancing algorithm, such as “Weighted response time” load balancing method, to find the faster-communicating nodes and reduce the number of connections, while simultaneously increasing transfer speeds.

7.5. Security

A common claim among the blockchain technology applications is that it is usually safe due to features such as decentralization and public-private key use for trust confirmation in smart contracts. This does provide a considerable advantage over common web technology that relies on traditional methods (such as a simple web server) to keep and transfer data due to the extra features involved that can improve upon the user's anonymity and increase the redundancy of the data used. Also, if correctly implemented, the immutability feature that the blockchain network provides gives a substantially more robust resistance to the more common methods of hacking. However, security most commonly relies on the implementation of the service/application that relies on the technology, and different approaches can either fortify or hinder the conceptual security that the blockchain provides. This can prove to be a point of question for most high-profile networked applications (such as cryptocurrencies), and therefore, the more common ways (and then some, that apply specifically to blockchain) of breaching security still need to be considered when creating a new application. The more common ones to be considered being:

- **Code vulnerabilities.** Writing code always poses risk for vulnerabilities (ex. remote code execution exploits using unsafe practices in combination with a low-level compiled language such as C). To prevent this, safe code practices should be maintained. Also, code reviews should be considered while developing the software.
- **Network consensus mechanism.** Since blockchain network relies on predefined rules for confirming changes (ex. multiple nodes confirming a block by mining process), such process needs to be thoroughly planned out and analyzed. Weaker cryptographic functions or flaws in implementation can lead to exploits that would allow unauthorized changes to the network's ledger.
- **Encryption.** In blockchain as a whole, encryption is critical to securing communication between devices. Data at rest and in transit should be secured using cryptographic algorithms **[**CDP20**]**. In blockchain file-sharing systems, not having encryption for storage could allow malicious nodes to extract data that they are not supposed to access by design. An example of that would be a node storing and extracting sensitive files submitted by other nodes.
- **Phishing attacks.** Users may fall victim to various sorts of social engineering attacks and give out their way of verifying their identity. Attackers use different deceptive communication tactics, such as pretending to be a government agency in order to gain false trust or scare the user into submission to finally give up their crucial parts of identifiable blockchain access data (such as the private key). This is very common amongst cryptocurrencies due to financial gain from

stolen wallets as well as considerable anonymity, allowing for better chances of successfully cashing out the gained illegitimate money under the radar.

- **Majority attack (51% attack).** Malicious parties could try to overtake the authority in the blockchain network by creating or hijacking enough nodes, to make the overall majority of the network. This could allow different network manipulations, such as double spending. One of the ways to prevent this would be to initiate a new type of node – Authority node that would be responsible of other node's authentication and validity. This solution would prevent any malicious actions in a majority attack. [IAA21].

Considering there are still multiple ways of subverting the security of applications that rely on blockchain technology, some precautions are important to be taken. One of them being reviews of code. A popular belief is that open-source projects are less susceptible to such problems due to them being able to be looked at (and possibly fixed) by many different interested parties. However, such claim can prove to also be negative, due to attackers also being able to freely analyze the code and discover vulnerabilities for malicious purposes.

Another way of combatting issues is use of already tested methods of reaching consensus for smart contracts. Using something that was tested by time and has already withstood many different hacking attempts throughout the years (such as Bitcoin), may be considerably more reliable than trying to come up with something new.

One issue that remains among most access-based services online is phishing. There are many instances where the weakest link in the usability chain is the user itself. To combat this, some extra features are usually added (such as 2 factor authentication online in order to prove that the user's credentials are not stolen), however these features may not be as feasible with blockchain due to its latency and overall sensitivity on real time key generation. A way to approach this may be to educate the user itself in safe practices of storing their access keys and laying down clear tips on communication with the service representatives.

Final consideration in terms of security is file security. In blockchain platforms, since files can be accessed by multiple clients, some consideration for file privacy and ownership may be needed. One way to perform this would be to only upload files that are encrypted with the client's public key, so only the same client itself can decrypt it with their private key. In this case, the files may still be accessible, yet only usable by their owner.

7.6. Interplanetary File System

While the blockchain constantly evolves, the same rule applies to file-sharing technologies that utilize it. Currently, there are many different blockchain file-sharing implementations available, but only a few can solve any one of the problems that are present. Yet most of them share a very similar base functionality for transferring and sharing files, which is an approach known as IPFS.

IPFS stands for “Interplanetary File System”. It’s a file-sharing protocol that relies on peer-to-peer connection between nodes and relies on hashing and version change tracking which all get broadcasted over the network and securely kept on each node. This system was released back in 2015 by Protocol Labs and is still being actively used and developed to this day. IPFS implements base key concepts, such as Blockchain and Distributed Hash Table (DHT).

The base IPFS functionality is done by providing a flexible method for transferring files of various sizes (no theoretical size limitation, yet it exists in most implementations due to storage space prices and constraints). The technology does not rely on a single method of network transfer protocol, and differs with every implementation. “When a file is sent to the IPFS for storage, a unique hash is generated for it.” [AOA22] This generated hash, by which the file is indexed, is used as a reference to the file, whenever it needs to be accessed. Data is replicated and shared throughout each node of the network, making the files easy to be accessed. It is also set in a decentralized manner, allowing for better reachability and redundance (ex. If the storage system fails in one of the nodes, the client can find the same file using the same hash on another node, without any notice or downtime). However, the downside of data replication is a bottleneck on data lookup, which happens during the process of a file request using a hash. After client sends the request, the search for the nearest node that contains the data begins, and can take some time, depending on the number of nodes that contain the file in the whole chain.

IPFS is a popular choice as a basis for blockchain file-sharing for many reasons such as:

- **Redundancy –** the data has copies throughout different network nodes, making it less likely to be lost.
- **Decentralization –** the network can work without a central server, instead connecting to the nodes that the network consists of.
- **Version control** – all the changes to the network and its data are tracked and saved.
- **Reliability** - having multiple nodes on the network reduces the risk of access loss to the service due to offline nodes, internet outages, or attacks (such as DDOS).

The main use cases of IPFS are as follows: uploading files, downloading files, permission control, and deleting files.

7.6.1. Uploading files

The client begins by requesting a file upload from one of the network's nodes by performing a request using the protocols provided by the application that implements IPFS technology. This request can involve authentication (such as API keys or cookies) if the service provides a login feature for reasons such as permission control. The request also usually contains the file data (or it is contained in a separate request, depending on the communication protocol implementation), which is uploaded to the connected node. After the upload data stream ends, the node generates a new cryptographic hash known as CID – Content Identifier. It is used as a unique reference to the file in order to be able to request and find it on the blockchain. At some instances, there can be multiple CIDs, if the application uses certain approaches to save storage space, such as file splitting, in order to create multiple smaller files to be shared throughout nodes. After the content identifier is created, the node creates a record of the file in the shared ledger that can then be discovered by other nodes that connect to this one. After another node connects to the one holding the file, they perform synchronization, in which the files that are not present on each node are synchronized. In many cases this process depends on implementation and does not apply to every file – ledgers can be synchronized, yet not all files are shared due to storage and bandwidth limits.

Amongst public IPFS networks it is common that files are not stored on every node to save space. One of the ways to decide what nodes receive what files is to have a preset setting which contains the weight (or percentage) of nodes that will receive the file. Another popular implementation is to have a changing weight for each file depending on its popularity, therefore whenever a file request is made, the popularity value increases, and the file is more likely to be synchronized amongst more nodes, making it more quickly accessible and redundant. However, this implementation can be flawed in terms of private or less popular files, due to them appearing in less nodes and therefore resulting in slower lookup speeds or availability.

7.6.2. Downloading files

To download a file, the client performs a request, containing the file's CID to one of the IPFS nodes. The request may also contain authentication data, if the implementation requires it. The node then looks up the file by its CID hash in the possessed ledger. If such a hash is not found in the ledger, a negative response may be given. If the hash exists in the ledger, but no file is found, the client then picks another node from its known nodes list in order to look there instead. The client can also utilize DHT (Distributed Hash Table) solutions to quickly pinpoint the nodes that would likely contain the file. Once a node that contains the file is reached, a download request is granted with a response that contains the file data. If the file was split into multiple files with

different CIDs, multiple download requests and searches for existence throughout the nodes may be needed.

7.6.3. Permission control

In most file-sharing services that run on a network, especially those that are open to the public internet rather than using only a private intranet, there is a need for controlling permissions. This requirement is even more prevalent for decentralized technologies such as IPFS due to their more anonymous nature by design. Permission control stems from multiple reasons, the most common of them being:

- **Storage limits** – storage space is usually limited and costs money to buy and maintain. Some services tend to limit storage space at free or cheaper options and offer expanded storage for a price (that often comes in a subscription form in order to keep a constant revenue stream).
- **User control** – control of certain permissions or limitations for users. For example: providing access to certain files, or restricting access to network due to various reasons (such as abuse of the service's defined terms of service)
- **Content control** – some material can be unsupported by the service or unwanted by its owners. Also, some content can be illegal (according to the law where the service is hosted or consumed from). Permission control allows preventing the upload (or access) of such content.
- **Bandwidth limits** – networked services get their connectivity to the internet from an internet service provider (ISP) that usually sets a fixed speed (bandwidth) limit. If many users decide to use the network at the same time the bandwidth might get used up resulting in slow speeds or loss of service.

Because of these reasons, permission control is often seen in most public file-sharing services that rely on blockchain technology. To implement control, such services usually require the user to create an account that gets linked to the service's blockchain network using a uniquely generated key, which can then be used to access and perform the requests needed to the nodes of the network. However, this sort of identification leaves a trace on the network (ex. X key did Y change, and key X was generated for user Z, therefore user Z is responsible.). This can lead to major issues with anonymity, due to the service keeping some sort of user data that can later be parsed and linked to the changes in the blockchain.

While by default, there is no implementation in base IPFS that would allow for different permissions (ex. users can reach each other's files), an option can be considered such as pinning public keys of super users or banned users to data kept in ledger, that would allow for permission

overriding (ex. deleting other people's files) or adding a rule for nodes to ignore certain public keys of service abusers in order to disallow further changes done by them to the blockchain.

7.6.4. Deleting files

In blockchain implementations data is usually considered as immutable. However, since files may be considered as additional data that is only referenced by the ledger but not in the ledger itself, deletion is possible. This is how the process works in IPFS due to the ledger holding the file's ID (CID) instead of the actual file due to size and efficiency constraints. Due to this implementation, file deletion is possible. The only thing required for performing such an operation is a signed change by the file owner or a super user (if the network allows for such a feature). In this case, a file would be marked for deletion. In IPFS this process is known as "unpinning" and instead of instantly deleting the file, it marks it for the garbage collection process which in simpler terms is just a file deletion job that runs on defined time periods.

However, one downside of such system is that nodes are trusted implicitly to honor the deletion change. When a deletion change is added to the ledger, each node is programmed to mark the file for deletion instantly or later. But in reality, nothing considerably stops a node from making changes to its own used blockchain code or executable itself with the aim of ignoring the deletion process, or to just perform backups and take the file with themselves. This is one of the reasons why file encryption or splitting may prove to be also promising in terms of safety. Since the file is encrypted, or the misbehaving node only has a part of it, there is a good chance that the data will be unusable and therefore of no worth to consider such an exploit.

7.7. Current implementations

At the time of writing this research, there are multiple services that implement blockchain file-sharing and are used publicly online. Each implementation varies in multiple ways, from inner base workings, to permissions, access publicity, and monetization. A few of more widely accessible services are as following:

- **web3.storage –** a public service implementing the IPFS protocol as a base. It uses the global IPFS network for storage and allows accessing files through it. User access is controlled through authenticating unique access tokens that are received when registering to the service. The service provides free and paid options that differ in file storage space limits. Each uploaded file is accessible publicly using a CID index; however, the unique file CID is 256 bits long, reducing the risks of discovery by brute-

force attacks. However, since the files are stored on the network without any layer of extra encryption, they are theoretically able to be accessed freely by third parties.

- **SAFE Network ("Secure Access For Everyone") –** an open-source blockchain file-sharing service with a base similar to IPFS that additionally focuses on anonymity and file safety. "The main goal of SAFE is to provide a network which everyone can join and use to store, view, and publish data without leaving a trace of their activity on the machine." [DT22] Before transmitting a file to the network, the file is encrypted with the owner's unique key, giving only the owning user access to the data. This service is paid, but it provides other features such as the ability to host simple websites and access them using the provided browsing utilities.
- **Filecoin –** a public cryptocurrency that is also used to pay for file storage using the same blockchain network. This service provides file hosting capabilities similar to IPFS to the coin traders, allowing them to store files on the network, which reside on mining computers by paying a fee. In this case, a mining node is used not only for block algorithms, but also as a rented file storage, which is rewarded by the same cryptocurrency. The service does not offer an extra encryption layer; therefore, this responsibility falls on the user to protect their files.

Each blockchain file system has its unique differences, yet there are a lot of similarities. One of them being that most variants implement crucial parts or the entire protocol of IPFS. Second one being that file encryption is not always the focus of file storages, which leaves the user with the file protection responsibility. This allows for the possibility of the mining node that stores the file to be theoretically able to parse the data of the files, if they are not properly protected. On a system that focuses on security, an idea could be taken from SAFE's implementation to encrypt files for each user individually, guaranteeing protection against unwarranted access from malicious third parties.

Another motive in common throughout most file storage implementations is payments for storing files. This is mainly because file storage has a cost that consists of physical media and running fees. In this case, the services involved either ask for payments based on the maximum size of files to be able to upload, or on a per-file basis. However, since the written services are hosted on public blockchain networks, the nodes that host the file may be public volunteering computers. Therefore, to incentivize hosting files, services such as Filecoin pay a fee to the mining nodes, which also do the file hosting.

The final common part is authentication: each service provides some sort of registration or unique token that is linked with a purchased service, in order to keep track of the access for uploading files to the network. This sort of token data, in some implementations, is kept on the

ledger in order to synchronize the entire network with information required to recognize users and their access level.

7.8. Proposition

There are various existing implementations of blockchain and its file-sharing technology derivatives. Some of them share noticeable similarities (such as file-sharing infrastructure being based on IPFS). But to solve the defined problems, concrete ideas need to be taken from multiple different solutions, in order to create a single cohesive system.

To begin with, a basic blockchain concept needs some changes and additions in order to support file-sharing. IPFS is a solid technology that many existing file-sharing services are already based on, and crucial parts of it, such as node communication, requests, and file indexing using unique hashes can be applied to set a solid groundwork design of a blockchain file-sharing system.

Going further, to be able to control permissions of the network, ideas such as user authentication by public key or unique user token can be taken from more popular blockchain file-sharing services such as web3.storage. This should grant an ability to provide decentralized user control in terms of receiving and deleting owned files. Also, it should give power to the network's administrator – the way to globally disable (ban) users from the network and delete unwanted files.

One common occurrence along blockchain file storage implementations (such as Filecoin) is that only mining nodes are responsible for file storage. This is a considerable solution in order to not rely on random clients for file hosting, as well as to only have nodes that are supposed to have better longevity, increasing their individual reliability. As a result, this approach should also increase the file lookup process, because only miners will be hosting files, therefore – less nodes will be needed to be searched through, or synchronized in a distributed hash table.

Noting the safety concerns of nodes that store files, especially the ability to parse and read the stored data, the idea to encrypt files can be leveraged, as it is used in SAFE Network**.** Before any upload process is done, files can be encrypted by each client's own public key respectively, limiting access to the file's content to its owner only, using their private key.

For faster file transfers, a load balancing algorithm similar to the one in BitTorrent protocol can be utilized, in order to download files in smaller chunks from different nodes that contain them. In this case, less bandwidth should be used for the hosting nodes, allowing to send out more file parts simultaneously, as well as higher download speeds reached by the client, due to downloading multiple parts at the same time, and not being limited by a single node's internet speeds.

To evaluate the specified solutions and their compatibility within a blockchain system, each solution will be developed as a module within a comprehensive blockchain file system. The main purpose of this is to demonstrate that these solutions can effectively address common issues related to access control and transfer speed in blockchain networks.

8. REALIZATION

8.1. Solutions

In the beginning, the first step is to propose solutions to the problems defined in the project. Taking each problem into consideration, and deciding on an approach for each occurrence separately.

First of all, in order to tackle the problem of increasing file sizes, the main solution was decided to split nodes into clients and worker (mining) nodes, the latter which will not only perform constant block mining, but also the storage for the files involved in the file system. Also, to further decrease the file size, an approach was created that would only select a percentage of files based on overall predicted node file system amounts as well as current node's identifier value, which would get used as a random seed in order to use a random number generation algorithm in the range of a given file storage percentage which in turn would allow to find which files that are registered in the ledger can be downloaded from a certain node.

In regard to the files becoming old and unused, the solution is to add a privileged deletion ability to the network, which in turn would allow either the file's owner or a privileged supervisor, for example a network administrator or privileged automated cleaner application which would force delete files exceeding a pre-set timeframe off the network's ledger.

To improve upon the issue of the possibility of lower file download speeds into account, it was decided to use the approach of downloading from multiple nodes simultaneously, as done in different file transfer protocols such as BitTorrent. This would not only improve the download performance by providing the ability to go past a single file hosting (upload) node's speed, but also reduce the network usage for upload nodes, allowing for more nodes to download files at once and reducing the overall network congestion.

Finally, taking the lack of permissions and malicious activities on the network, it was decided to add additional ability to identify and ban different nodes in the network. This would allow privileged users (such as administrators, or privileged watchdog bots based on different heuristic rule sets) to insert the banned node IDs into the global ledger, which in turn would then

cause other nodes to cancel and ignore all communications with any banned node that tries to parse data or perform blockchain transactions.

To test all of the proposed solutions, a blockchain file system application was designed, which would implement each algorithm and allow for testing the applicability of these in blockchain file networks of different sizes and further applications.

8.2. System design

8.2.1. Main principles

In order to create the system, at first, the main structure of the program was planned. This step involved deciding on the software components that will be the building blocks of the system, as well as planning out the involved modules and their connections. Each functionality that is needed by the system (ex. Networking, Blockchain, Serialization, etc.) is split into a different module and done separately and only then connected throughout during the latter steps to avoid causing problems during parts of the development that may not allow for quick debugging due to the premise of unfinished features. This is important not only to hold on to base coding principles but to also provide smoother developing experience by creating each module separately and testing their own functionality and bugs, so less issues are experienced when doing the finalization steps which will interconnect the finished modules.

During the planning and development phase, a lot of focus was also spent on deciding where to use concurrency features provided by modern programming languages such as asynchronous tasks and threading. Since there is a need for client/server architecture to provide networking features of the Blockchain system, the client and server were written in a single thread using asynchronous programming in order to provide simultaneous and thread-safe way of testing the communication between nodes (client for performing requests and server for receiving connection requests), allowing for simultaneous node download in the same thread, as well as leaving more space in terms of multi-threading performance for more performance-heavy tasks (in this case – Mining).

8.2.2. Architectural design

The main design of the system was to support the base functionality of a Blockchain platform that shares block data between nodes. To create such a system, at first, a networking system was decided. The networking would consist of a non-continuous request system similar to HTTP, where a client and server would be involved. A client would usually send requests, while the server would respond with data that is requested and validated in accordance to the client’s

needs and permissions. For such communication, each of the client and server modules was split into separate parts, yet written with a lot of shared object types, such as Packets that were designed as a tool that would help serializing, encrypting request and response data. The networking component, consisting of client and server, would utilize its own key storage, which would contain unique RSA2048 keys that were individual per node. These keys would then be used to encrypt the traffic being sent to the server as well for key exchange (public keys) in order to avoid snooping of the data traffic in cases such as Man in the Middle (MITM) attacks.

The networking system would load the visited node key list in order to find out the nodes that may be available to connect to and connects to them. The very first key is pre-set to a master-server which would serve as the first point to connect to in order to get information about other possible nodes in the blockchain. This is done similarly to the Bitcoin platform, where a list of master-servers is pre-set for the initial node discovery process. After connecting to the specific node, the current lists would be exchanged between nodes, increasing the known node lists and removing the need of a master server, if it was an initial launch.

The application would have a constant data storage which would store the cached blockchain object data, as well as other modules such as keys and visited nodes involved. Such data would first be serialized using XML language in order to be able to quickly exchange between structured class data inside the application and serialized data stored on the drive.

A signing module would also be provided. Since the blockchain systems requires nodes to have their transactions signed with a unique public key, the private/public RSA2048 key pair is used for signing outgoing transactions/mined blocks and validating the incoming data.

A mining module is a vital part of the blockchain system. Since the blocks will be confirmed and added to the chain using a Proof-of-Work algorithm, a mining system is needed. It will split into workers and continuously (using brute force methods) look for a fitting algorithm solution that would be acceptable for the preset rules of the blockchain. These workers will be scalable into any provided number of threads, which would work towards the solution independently and increase the probability of the current node reaching the mined block first.

All these separate components would, in turn, result in a full blockchain system, which would, in turn, allow for full usage of the modules involving the proposed solutions. The overall component design can be visualized (see Figure 1).

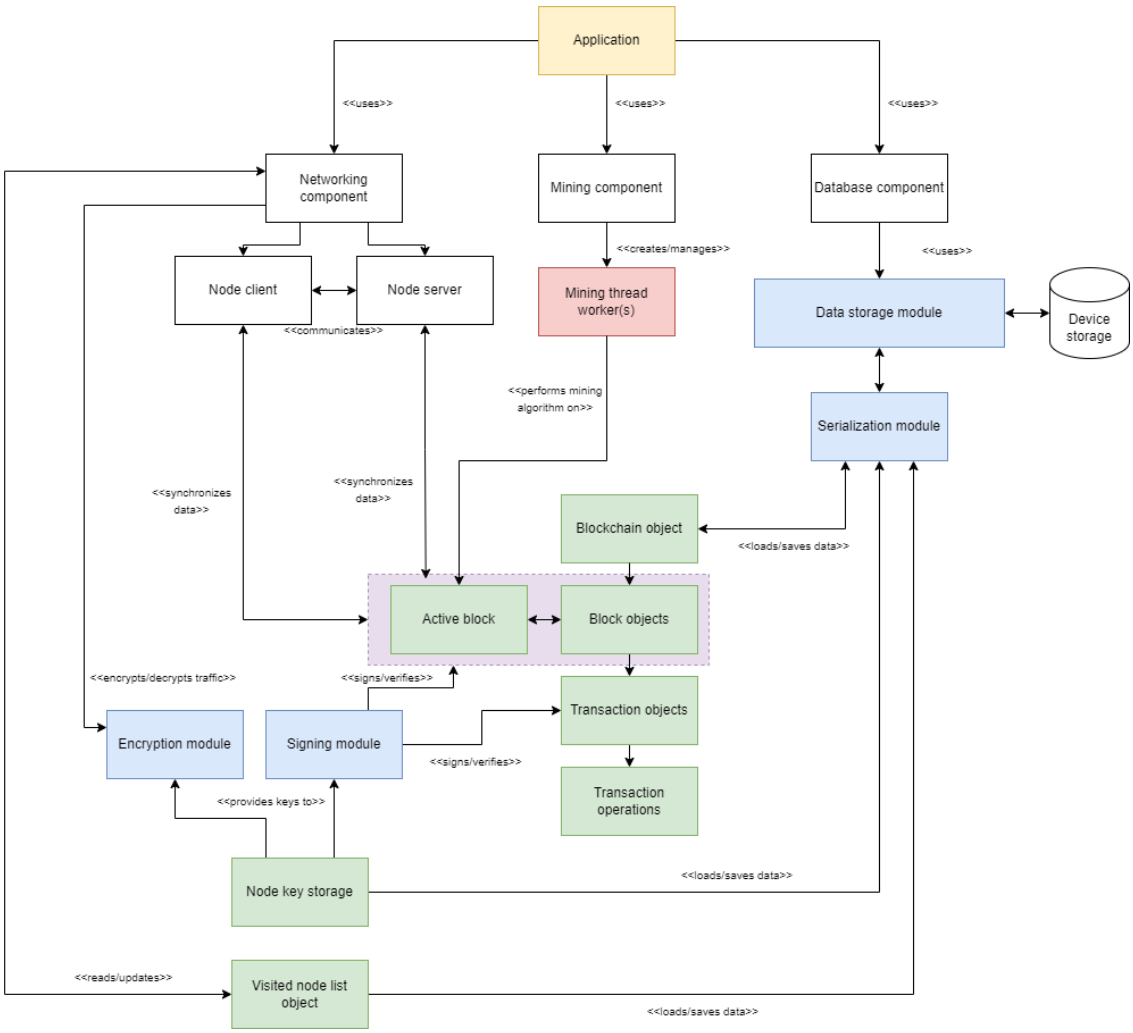

Figure 1: Software component design diagram

8.3. Networking system

8.3.1. Basics

The networking system begins with the selection of the base communication method between nodes. For this, a TCP networking protocol was chosen in order to provide a reliable communication without the unnecessary risks of connectivity/packet loss that are usually more involved with UDP protocol. After the main protocol was selected, the main connectivity interaction was decided. The system was designed to work on a single request per TCP connection basis, similar to HTTP API requests. However, in difference to HTTP, it was decided to use a different way of formatting sent data and encryption methods.

8.3.2. Packets

The system uses a shared way of formatting data in the form of packets. A packet class is created that allows writing data to a byte array buffer of variable length. For ease of access, multiple ways of writing data are created, starting from basic ones such as writing pure bytes and appending byte arrays, to writing integers, floats and even variable length strings, that are also UTF-8 encoded prior. Similar functions are created that work in reverse, allowing the reading and sequent deserialization of data (numbers, strings, etc.). This allows for quick way of writing and reading data that is then shared across the connection between client and server. The packets also contain a static header and additional header data which consists of a packet identifier and its length. Static header allows the server to quickly identify packets that may malformed or sent by a client of an incompatible program, making them quicker to dismiss. The packet identifier tells the receiving end on how to process the incoming packet.

8.3.3. Communication logic

The system consists of a TCP Client and Server, each node running both objects simultaneously using asynchronous execution. The server constantly listens for requests from the incoming clients, while the client is used to connect to other nodes (aka. their server listeners). Once a connection is made, the client begins the first step. The client sends the first packet with an according packet identifier (key information packet), which contains the client's RSA2048 public key. The server then verifies the packet and public key structure, and if the checks pass, the server then responds with a packet of the same Id containing its own public key. This time, however, the packet is encrypted using RSA2048 with the client's public key, so only the client could read the data with its private key. The client then itself reads and verifies the returned packet, takes the server's public key, and uses it to encrypt the following packets. After this handshake step, the client is then ready to perform a request. The requests are made by the application depending on the need, whether it would be asking for the node list, a list of blockchain blocks, or the currently active transaction counts which are yet to be mined. The client creates a specific packet, which contains the data that is needed by the specific request. After the request is sent, the server validates and executes the following request. Afterwards, depending on the specifics/type of the request, the client may or may not wait for the server to return a response. Once the request processing ends, the client and the server close their connection. The server then returns into its waiting loop, allowing the next node to connect and perform their own requests. The same client may also perform subsequent connections to the server node to perform their needed requests if a need arises.

8.3.4. Request processing

Before the application decides to send a request to another node, a request object is selected. This object contains the appropriate packet identifier, as well as initial data that may need to be sent as a request parameter (ex., the block identifier to request). This is then passed to a preparation stage (method), where the parameters or the initial packet may be freely modified before being sent to the server. The request processor class features two methods – one for handling the according request in the server and another – for handling the response in the client. This makes it more optimal to add new request functionality quickly and allows each different class to contain the functionality separately, increasing the cleanliness in the code while also keeping it short. These methods are inherited from a base request processor class, which allows to select a different class object depending to process request based on the packet identifier.

When the client sends the prepared request packet to the server, the server then handles the request in an according request processor class request handling method. The request handling method validates and performs the requested functionality, and then, if needed, creates a response packet which would finally be sent back to the client. Following this step, the client, if a response is needed, would then wait for the server's response and process it accordingly in the request processor's client-side response handling method.

8.3.5. Node lists

To be able to connect and discover nodes, a first node is needed to gather initial data from. During the initial start of the application, there are no nodes known, therefore there is nowhere to connect to the blockchain network. To solve this problem, a first node is added by default to the node list. This node is expected to constantly work in order to provide other nodes with initial information, and is not required after the first run for the network. When the client connects to this node, it takes its node list which consists of other node data previously connected to/from (which includes IP addresses, node type and unique node identifiers). After this process, the node list is serialized and saved onto the physical drive and further node lists can be sent or downloaded from nodes other than the first one. Also, the node list can be populated by previously unknown nodes that connect to the current node as well. This way the node list constantly increases and the network can work in a decentralized manner.

8.4. Mining system

8.4.1. Algorithm

A blockchain system is usually reliant on some sort of mining system in order to verify and confirm block data while finally adding it to the chain. The developed application also uses mining to perform such a task. The main mining algorithm is based on proof-of-work algorithms and works by constantly calculating hashes of verified data of the current block. The entire block's (including its transaction) data is taken, its data is verified checking the RSA2048 signatures of block transactions with their respected public owner keys to prove validity (and the invalid transactions are removed if any), and then finally in a mining loop, a random 64-bit long integer nonce value is generated and added to the end of the block. This data is then hashed using a SHA256 hashing algorithm. The sum of the hash value is then calculated. The sum is then compared with the current sum limit setting, which in turn decides whether the mining attempt is successful. If the resulting sum is below the sum limit, then it is considered a success, and such block then becomes successfully mined. After reaching success in mining, the block is then transmitted to other nodes by notifying them of a new block using a notification request. If a mining success is not reached within a given time limit (by configuration), a new block is generated with the current pending transactions and the mining algorithm loop is repeated. However, if a new block is mined by another node first, then the new block is first accepted into the chain, before a new active processing block is generated for the mining task with the remaining pending transactions that may not have ended up in the newly mined block.

8.4.2. Threading

The mining system also features threading. This is needed in order to allow higher speeds and therefore higher probability of a block to be mined based on the current performance of the computing device. The system features a thread manager, which sets the currently active block to be mined, as well as its transaction list, that would constantly be increasing under use. The manager features a mutex lock on the shared block data in order to prevent race condition errors, and this data is only accessible through according get/set methods that would ensure correct locking and passing of data without issue. During the execution of these methods, a deep copy of the data is done in order to avoid writing to a class reference that may still be inside the shared block object, causing another way of a race condition. The thread manager also has a volatile result state value, which is meant to signal when a block is successfully mined and is available for taking.

The mining system consists of a thread manager and worker threads that are created by it. The number of thread workers created can be any, since each thread performs its calculations

independently. The workers each take a copy of the processing block from the thread manager using the designed get/set methods to keep the functionality thread-safe. Then they each on their own check the thread manager's result state to see whether any other worker has already completed the mining task. If such a task is completed, no action is taken, and the thread constantly checks for new task (a processing block to mine) with 1000ms delay times in order to not overcrowd the CPU core power. When a new processing block is set in the thread manager, the result state is cleared and the thread begins its calculations anew without any delays. During active calculation loop, the worker also constantly parses the block data from the thread manager every few seconds, in order to get the newly updated data (in case new transactions are available to mine).

After the calculations, if the worker thread has successfully mined its own block, it sets the resulting block in the thread manager with its own mined copy that consists of the suitable nonce value, and the manager sets the according result state. Finally, this result state is read by the application, and then the resulting block is transferred to other nodes.

8.4.3. Performance

To check the viability of the implementation of the mining algorithm, as well as to check the performance settings of the currently pre-set task difficulty (the hash sum limit value), multiple series of tests were conducted by running multiple mining threads (between 1 and 8 as provided by the multi-core processor of the machine these tests were running on). These tests consisted of creating a bogus block with 10 separate transactions. The test would begin with creating such a block and then starting a pre-defined amount of mining processes. The mining process difficulty (the hash sum limit) would also be set into different values to see which value is the more applicable for a small network. The application would then wait until one of the threads was successful in mining the block, and provides the time difference (between the start and end of the mining process creation) to show the time it took in mining a single block. Also, since the mining process involves generating random values, and in true random fashion, there is a very small chance of getting the hash in a few (or even one) attempts, multiple test runs were done, and the resulting times were averaged out into a single total second value for each, and a data graph was created based on the average time values. In the end, the main purpose of the test is to see how quickly the algorithm can mine a single block and to consider making adjustments in accordance further on. The first test was run on an 8-core machine, going up from a single core to 8 for each different hash sum target and core count. Such a run was treated as a single cycle, and multiple (14) cycles were ran. The test results are presented in Figure 2.

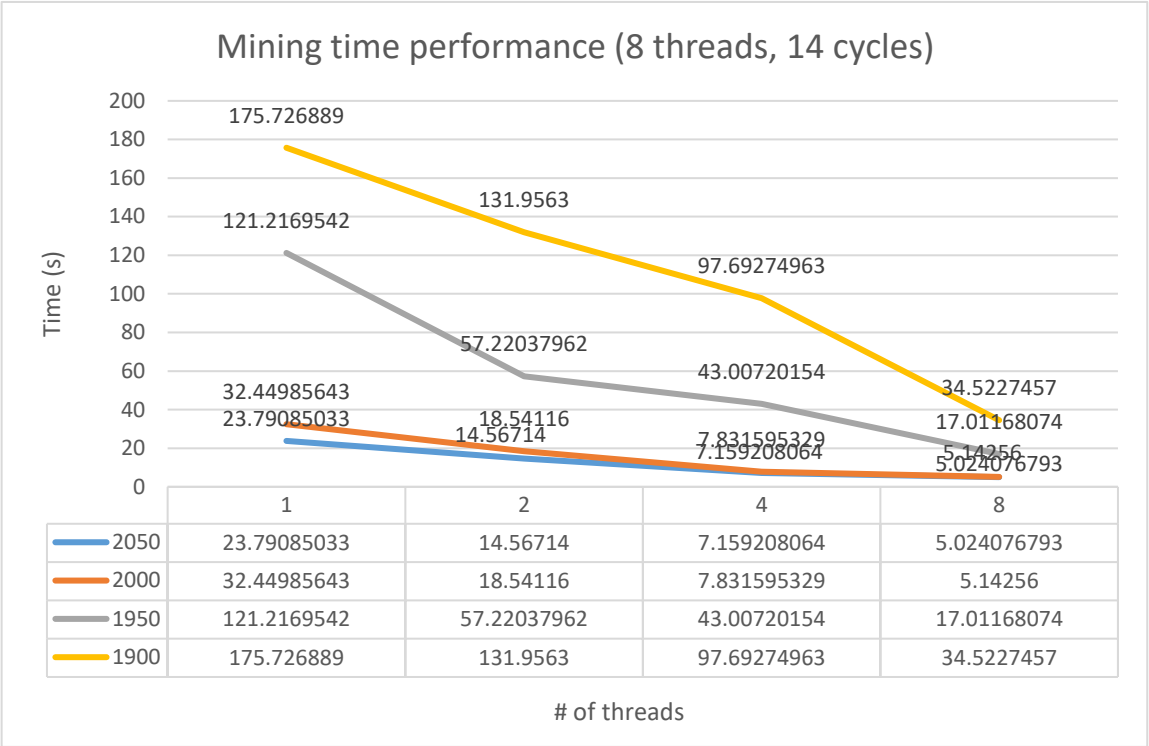

Figure 2: Mining algorithm performance per core count and task difficulty

Judging by the data in the test result, multiple cores working on a single mining task can indeed increase speeds considerably. Comparing the speeds between 1 and 2 threads, the time decreased by up to a 50%. This trend continues with the mining time decreasing on 4 threads. Going further into 8 threads, the mining time drop-offs were less significant on quicker mining difficulties (2050 and 2000), although in more difficult sums to achieve, the speed increases are more apparent again, reaching up to almost 65% increase in speeds at 1900 difficulty.

Moving along, the mining speed increases seem to be similar yet slightly random per thread count, ranging from (10-65%) increases per doubling of a thread count. The 8-thread result is especially an outlier, considering some quicker mining tasks (2050-2000) speed gains are considerably lower. This can be a result of certain random chances given during testing, although can also mean uneven thread scaling due to the testing environment's OS scheduler picking same cores which is a tendency with higher core counts. Overall, the speeds improve considerably by increasing thread count, and this tendency can be extrapolated for multiple different nodes along the network. Meaning that the more mining nodes there are, the faster the mining task will be accomplished. Judging by the current data, the simplest (highest) task difficulty provided (2050) can be mined in about 5 seconds under the load of 8 threads (or simulated 1-core nodes) which while suitable for testing purposes, is not applicable to a bigger network, due to blocks being mined way more often than any transaction occurs, resulting in a ledger bloat and unused blocks needing to be processed while recreating databases. This is why, a more difficult task should be picked for

a bigger consumer network. The difficulty increases the smaller the hash sum target becomes smaller, due to the likelihood of hitting such a sum decreasing almost exponentially (as judging by the times increasing up to magnitudes of up to 5 times when looking at 1 thread speeds for different task difficulties (2000-1900) in the chart). The difficulty of 1900 seems to be difficult enough to reach around a minute of a block turnaround on a very small network (if averaging the values between each speed of different amounts of threads of successfully mining a single block). However, for big networks, such a value should be picked way lower, so the average predicted number of nodes would take a longer (ex 10 minute or different, depending on how many transactions are expected per given timeframe) time of solving the mining task.

The discrepancies between different time drop-offs on different thread amounts can be accounted to the fact that the time to get the hash is still random, and can result in disproportionately quick or slow times depending on the luck of the current test run. In order to increase the accuracy of the data to show better averages of possible mining time decreases by increasing core (node) counts would be to run the same testing process hundreds (or even more) times (considerably on different hardware set ups as well). This would further decrease the likelihood of random tendency to skew results and show the more accurate average times expected in relation to core counts. To perform a test that can be done more times, a 3-core server was taken and the same mining block testing application was ran on all 3 cores for a time span of about 3 days to reach over 100 cycles of mining block tests with different difficulties and thread counts.

Judging by the data presented in Figure 3, the tendency seems to be similar. There are significant drop-offs in time, only increasing in magnitude by the task difficulty and core counts. It can be said, that the time difference when adding an extra thread (3) in comparison to two threads is noticeably lower than what a doubling in threads (ex. 1-2-4-etc) can achieve as shown in the previous graph (Figure 2). Yet after running this for a longer period of time, the effect on the performance data caused by the randomness in the algorithm is significantly reduced, and each of the thread count increases show signs of a stable drop-off in time required to mine, instead of fluctuations, that can be seen in the lower cycle count test above.

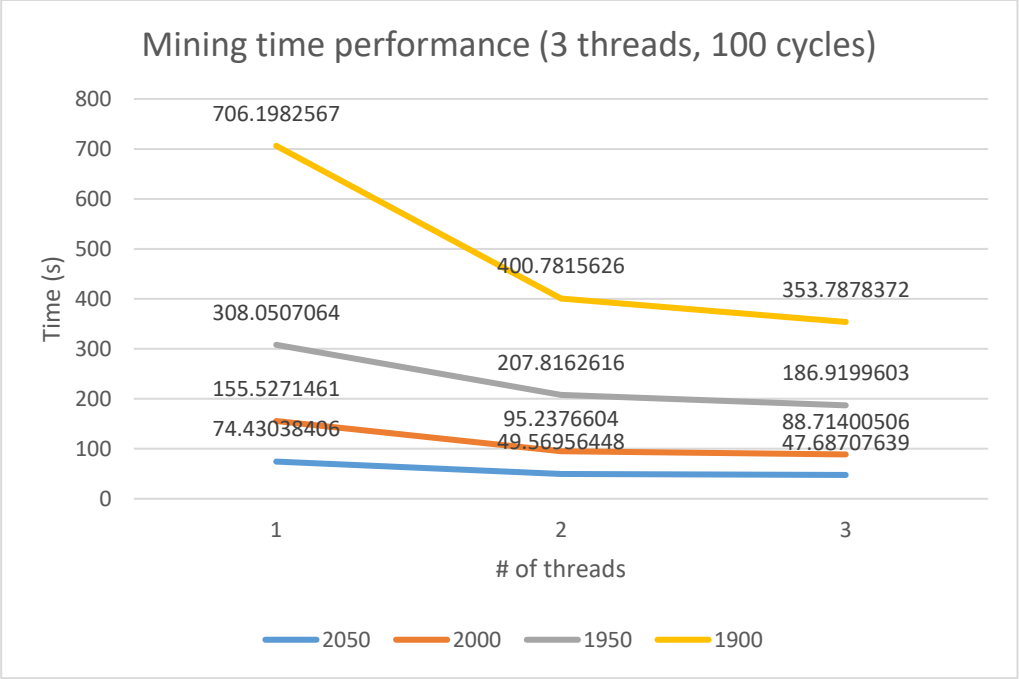

Figure 3: Mining algorithm performance per core count and task difficulty (higher amount of tests, less cores)

In the end, judging by the data, the speed gains were also up to around 45% when jumping from a single thread to two threads, and lower gains (around up to 13%), when switching to three threads. This can be influenced by slower and virtualized cores provided by the server the long test was run on in comparison to a dedicated machine, as well as in the performance of the processor the algorithm was calculated on. The difference in a single core performance is very noticeable when comparing the speeds of the highest (1900) difficulty mining sum value, which in turn shows that the previous machine which ran the algorithm could be up to 3 times more faster per core in comparison to the server. This metric is a good indicator of the possible differences in machine performance and in turn – mining speeds, which will overall make the ability to calculate the average mining speed difficult in advance.

The mining time can also change with time when more nodes appear in the network, or as the overall performance of the technology used increases with the future advancements in CPU and GPU markets. This is why for a wider application of the system, a functionality for varying the task difficulty depending on the network size and average block mining speeds should be considered. A similar approach is done in commonly used blockchain applications (ex. Bitcoin) to maintain average mining speeds.

8.4.4. Communication

When the miner thread successfully mines a block, it is marked as successful and is transferred to other nodes. Because the transactions are constantly being updated in a separate loop from other nodes, new transactions may have appeared that have not yet made it into the newly mined block. If this happens, the leftover transactions (or an empty transaction list if there are not any) are taken, and put into a newly generated active block, which then will be again mined for the time limit provided by the configuration.

When transmitting the block, each node that receives the block, validates the block's SHA-256 hash as well as its nonce value's combination to match the pre-defined rules to not exceed a given sum. If the hash is correct, the transactions are then each also validated by their respected hash signature values. If any error is found in this step, the entire block is disregarded, and the node awaits new data, or mines its own. If the block is successfully validated, it is added into the transaction list, the transactions are processed by the transaction operation system, the existing transactions in the node are refreshed, removing the ones from the mined block from the active list, and the mining process is restarted to fit the according changes. Finally, the node transmits the valid block further to its known nodes to make the changes reach the wider part of the network.

Since blockchain system is decentralized, some changes take their time to expand throughout the network. This means that in bigger systems, or those that have less hard proof-of-work tasks, can have a rare possibility of more than a single node mining a new block. This happens when neither of the nodes are yet aware of a new block appearing, and usually, both blocks have the same link to a previous block, but different transactions or contents. In this case, certain rules need to be made in order to handle such an issue, otherwise known as a block collusion.

When a block collusion occurs, a node receives two different blocks that reference the same previous block and may contain some of the same transactions as the other one. In this case some of the common rules are applied in order for each node to automatically accept one of the blocks, and refuse another. These can be: the number of transactions included in the block, the time the block was mined, the overall nonce sum amount, etc. In the case of current implementation, the transaction amount is selected to be the comparer for the first priority. In the very unlikely case, the transaction amount would occur to be the equal, the blocks would then be compared by their timestamps and the one selected would be the one that is mined earlier according to their time data. The logic behind this selection, is that more transactions mean that more operations and changes to the filesystem can be performed from one block and earlier in the blockchain than needing to wait for the next one to be mined. Also, in terms of the timestamp comparison, the block that is

mined earlier is selected due to the possibility of a such a block having higher likeliness of being spread further along the network, than the one that was mined later.

After a block is selected during collusion resolving process, the other block is discarded; the transactions that were contained in the discarded block are returned to the active block mining queue to be mined in the next block. The node then only proceeds to broadcast the mined block that was selected during the collusion resolving process. The mining process inside the node is updated in accordance of the changes of a new block being created as well as the missing transactions that were added. Afterwards, the system returns to a normal process of mining the current block, as well as waiting for new transactions, or blocks that will be mined from the other nodes first.

8.5. Storage system

The storage system's main purpose is handling the file system interface with the device as well as providing a useful tool of quickly serializing/deserializing and storing/loading data. This system is used for multiple parts of the program, such as saving the current blockchain state in order to cache it, loading database data, loading node lists, and storing the files that are received over from other nodes.

The system's base is the file loader platform, which uses the involving operating system's library provided tools for reading and writing data. This allows for quick file access and provides the functionality of reading and writing byte arrays into specified files just by calling a single correspondent method. Following this functionality, the next step is serialization. Objects such as blockchain, database, node list are all defined by separate classes in the program. This would normally require each field of the class to be written manually into a file using some sort of defined formatting and then loaded back the same way. However, this problem is solved using XML serialization. This allows for the ability to turn a class object data into XML text data and vice-versa. According to the inner workings, the text is still generated by looping through different fields, but instead of requiring manual work of specifying each field, the field lists are achieved by using techniques that allow the application to know specific additional parts about its source code design, such as reflection. Reflection allows for gaining information about the program's classes during runtime without having to re-define them manually in code, however it also comes with the downside, that some compiler optimizations may not work due to reflection data being required to be saved into the resulting executable file.

The last remaining part of the file storage pipeline system is the base64 encoder. This was chosen in order to keep data more obfuscated and therefore, less likely to be edited manually. At

first, the XML serialized string is taken and turned into a byte array using a UTF-8 encoding to preserve the possible multilingual characters. Then, the following bytes are encoded by the base64 encoder which turns the array data back into a base64 encoded text. This text is then saved into an according file.

The file loading process is done in reverse. For files that are stored from the blockchain file system, the file gets read and a byte array is provided back to the calling module without further processing. However, in other cases that involve serialization, the file data is taken and it's full base64 string data is read. Then, that data is turned back into a byte array using a base64 decoder tool. The byte array is then turned back into XML text using a UTF-8 decoder. Finally, the resulting XML text is then finally deserialized into the provided class type, providing all the data the class needs to function. However, if an error is detected in this pipeline (ex. corrupted data, inability to decode data, bad XML structure or unsuitable type), the storage manager returns no (null) object and the calling modules need to handle the fallback option on their own (ex. create a fresh initial node list if one did not exist or was corrupted).

8.6. Database system

The database system consists of a module that keeps the resolved data of the blockchain. This system is updated by the node system that constantly adds newly mined blocks of the blockchain. Each blocks' transaction is then processed accordingly and cached on the local drive using the storage system for quicker access later, without the need to re-process the entire blockchain.

The main purpose of the database is to provide the resolved ledger/file-system data that is accumulated by the blockchain's constant modifications. This data consists of currently available files on the blockchain, as well as administrator users and banned users (users – identified by public keys of their specific nodes). These modifications are done using unique transactions that are present in lists in every block.

Each transaction consists of an operation that is performed during its execution. The operations are currently as follows: add file, remove file, add administrator user, ban user. Each node that is not banned, can perform transactions, but privileged operations (such as banning a user or adding a new admin user) can be only done by nodes that are present in the administrator list. The transaction operations, similarly to network's request handling, features an operation processor, which, depending on each operation type in a block, validates and performs the steps that are described inside the according transaction operation type. For these operations to work, metadata is needed that would describe the parameters required by the operation (for example,

adding a file to the blockchain would require its file name and hash). These parameters are different by operation type and are serialized/deserialized using XML and UTF-8 encoding into byte array due to each parameter being defined by a different class type and therefore this causing it to be optimal to load and save operation data. During creation of the transaction, the operation type is defined and parameters required for executing the operation are defined. Then the transaction is signed by the node and is added to the current active processing block, as well as transmitted to other nodes. After the transaction is validated, confirmed by other nodes and is finally mined into a block, each node processes it alongside other transactions that have ended up in the block. The encoded operation parameters are deserialized from its transaction's attached metadata. After parsing these parameters, the operation processor runs an according operation by its type as specified in the transaction object with the deserialized operation parameters passed. Each operation performs changes to the owned ledger database by either adding new entries to the banned/administrator user lists or adding/removing entries from the file list. After processing every transaction that is saved in the block, the database is saved as a local cached copy for more optimal access and to save time of having to re-download and re-process the blockchain data once again.

Lastly, during the first run of the node, the node first downloads the block list data from the master node or other nodes that it later visits. It then downloads every block that is missing from the current local blockchain, and processes each block accordingly, in earliest-to-latest order by its timestamp.

8.7. Blockchain file system

The file system connects most functionality from other systems in the program. First of all, the system parses the data provided by the resolved database for the currently available list of files that could be downloaded from other nodes. Then this list is checked against currently available files in the local storage by their identifiers and file hashes and file sizes. If any of the local files match their supposed identifiers from the database but mismatch their hashes or their sizes, or in other way, if the according files are not found on the ledger (ex. they were deleted by a recent transaction), they are automatically deleted from the storage and marked as unavailable (a.k.a. not found on the local storage). In case of mismatched existing files, the system is then further prompted to re-download the deleted mismatched files. Same is done for files that just do not exist on the local storage. The available files are kept and are available for transmission to other nodes, as for those that are unavailable, they are constantly being queried from other nodes to find the one that has the file available. When such node is found, the file is downloaded and a local copy is saved, providing more access and redundancy in the overall network.

8.7.1. Storage splitting

For a mining node, which is responsible for file storage, it has the responsibility of keeping and serving a certain percentage of files, as defined in the global settings.

At first, the percentage is a static number that is decided initially by the network's creator. This is then used by each miner node as a limit value in random number generation algorithm do decide whether the file should be downloaded or not. The algorithm is as follows: the node's unique identifier is taken, converted from into a numeric expression. Then for each file that is present in the ledger, the file's unique identifier is also taken and converted to a number. For each file the sum of node's identifier and file's identifier numeric values are passed as the initial seed into a pseudo-random number generator (as provided by the programming language's library, although any seed-based pseudo-random algorithm can be used as long as it is the same throughout the network). The random generator is then ran to generate a value between 1 and 100000. The resulting value is then divided by 1000 to put it into a hundred's precision with some leeway for smaller decimal precisions in case they are needed. The resulting number is then compared with the percentage number, as set in the global settings. If the resulting number is smaller than the percentage limit, it means that the miner node must download and host the file, if it is higher – the node can freely ignore it. This algorithm is quick and, on modern machines, does not cause any considerable delay that would negatively the file lookup process. The same algorithm also allows client nodes to quickly figure out which miners contain the wanted by simply parsing the miner node list and passing their according identifier and target file identifier numeric sums as an initial seed into a random number generator of the exact same kind (since that would result in an exactly the same yielded value based on the seed given).

This allows for saving storage space without having to save all the files present in the network, as well as quick containing node lookup without having to query every node differently while looking for the wanted file.

This algorithm, however, can be further improved by dynamically changing the percentage based on the changes on the mining node count in the network, whilst also (if percentage decreases) deleting whatever file does not match the decreased percentage set. However, this solution also adds availability risk since some miners may not be always present and therefore the percentage may result to be not as an ideal for storage as an initially preset one, although this can also be accounted for in different ways, for example, by taking a week's (or other timespan's) average amount of network nodes into the percentage calculation algorithm.

8.7.2. File download

For mining nodes, the available file list is constantly queried throughout random nodes when a local file is missing from the current file list as specified in the ledger and further resolved by the storage splitter algorithm. These queries are either performed in a loop (during a missing file) or after receiving a newly mined block to receive the rest of newly available files.

During file download, multiple mining nodes are resolved from the node storage splitter algorithm and then queried at the same time. Since the current client node has initial data about the file, such as its identifier, name, hash and size, it can decide how many chunks to split the file into. At first, the node creates the local file filled with zero bytes up to the size of the file. This is done to pre-allocate the space of the file inside the file system. Then the node creates a buffer, consisting of multiple chunks of a similar (default: 1MB) size. Then multiple nodes up to a set limit (5) are queried for a file chunk: a defined TCP packet operation that includes the file identifier, chunk size and starting offset/index. If the queried node is reachable, it checks its local storage for the available file. If such a file is available and has the correct hash, size and identifier as by defined in the ledger, it's requested chunk is then sent to the requesting node using a continuous Write and Read cycles that are specified in the TCP socket platform. This sending system works by first creating a packet by reading a chunk up to maximum chunk size (1MB) from the file at a given index and put into the communication packet which is then encrypted and sent by the TCP continuously until the whole buffer is sent. This is done by seeking by the number of bytes actually sent (as returned by the send method), before sending another data buffer that goes next. This is continued up until the full chunk size is sent. The reading is done the same way in a loop, just by reading multiple parts of data that are available to read per a single reading method call finally consisting into a single chunk buffer of data. The buffer is then decrypted according to packet private key encryption methods and the data is written upon the local file at its regarding offset inside the file system. Its index is then marked as downloaded in the system memory. This whole process is done in asynchronous parallel operation, in order to communicate with multiple nodes while downloading multiple different chunks at once. This helps with download speeds, because multiple nodes can be communicating data at once, making use of more of the client's internet speed. When all chunk indexes are marked as downloaded, the file is finally considered fully downloaded and a new unavailable (not yet downloaded) file is selected to be downloaded next.

To check potential file download speed increases, a testing suite that creates up to 8 file serving nodes and a node that downloads the specified test files was created. Each node was placed on a separate machine, one of them being a local machine server and others being hosted on two separate remote virtual private servers, running multiple virtual machine instance nodes using

Docker virtualization technology. The test was conducted using three files of different sizes, and with multiple settings, picking between a single and up to 8 different nodes to measure speed gains provided by the simultaneous download process. The result was measured by taking the time difference between the start and end of the file download process (ex. between picking a file to download and saving the final chunk) and then dividing the file's overall size in megabit expression from the total time (in total seconds with milliseconds specified after decimal point). This results in a megabit/second expression, allowing to display the possible size/time gains in one of more common ways as used by popular file transfer software or browsers.

The first file was a ~1.84 megabyte .gif animated image. For this file, since the maximum file chunk size is marked as a 1 megabyte, which means it would require at least two chunks for downloading. For this instance, at best, two different nodes would be used to download the file simultaneously. After performing the test with different node settings, the results were as presented in Figure 4.

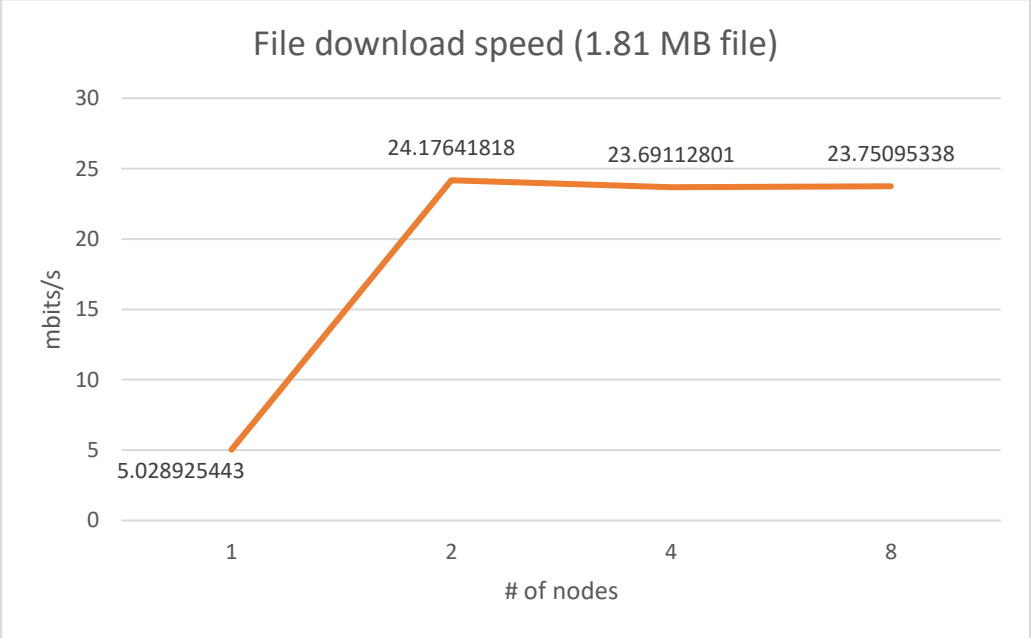

Figure 4: File (1.81MB) download relation to node count

Judging by the data, it shows that there was a significant speed gain when switching between a single and two nodes. This shows that when file is made of two or more chunks, a speed benefit can be achieved be downloading from multiple separate nodes at once. This also shows, that no extra speed would be gained from more nodes than possible file chunks, due to other nodes not being assigned any work/connected to, since there is no other chunk to give them work from.

Further looking into the data, it is possible to see that the initial speed was around 5 Mbits/s and increased by almost 5 times when using two separate nodes. This scaling is not linear and could be the result of different connection speeds as well as processing power of separate nodes,

especially considering that each chunk still needs to perform its handshake part with the according node and the encryption/decryption process. Also, since the speed measuring part takes file saving and chunk marking part into account, this could mean that the initial first chunk download is remarkably slower simply because of the preparation that comes before the connection is made.

For the next test, some bigger files (8.54 MB and 50MB) were taken. This was done to ensure the use of the full extent of the testing environment (8 nodes) and to properly check the scaling that the system provides with its capabilities. The results are presented in Figure 5.

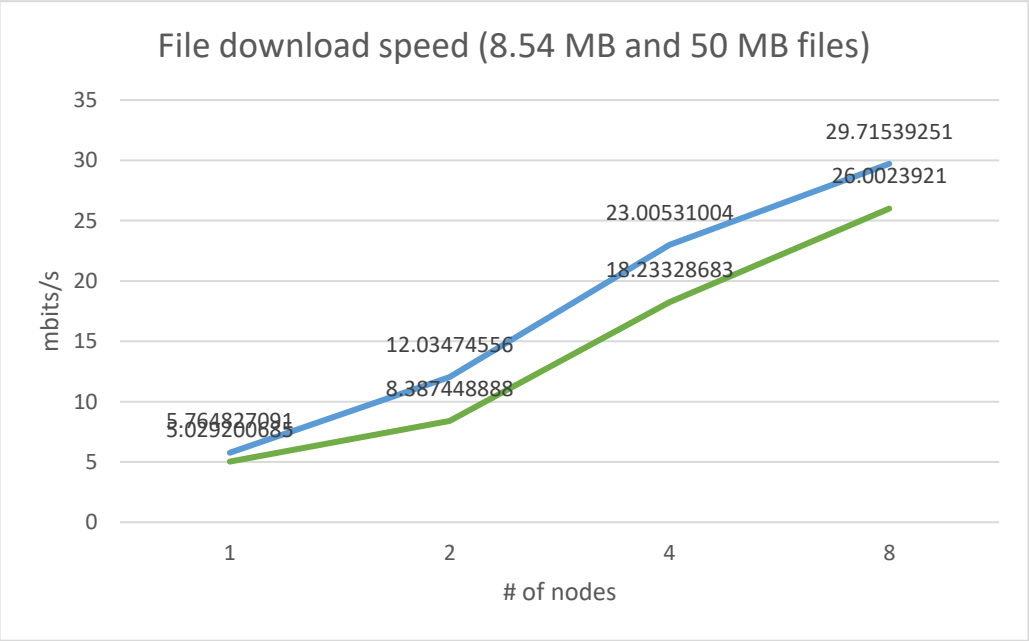

Figure 5: Files (8.54 MB and 50MB) download speed relation to node count

According to the results, both files show a similar tendency of speed increases. However, in comparison to the previous smaller file, the two-node speed increase is noticeably smaller. This may be due to the fact that each file needs more chunks to be processed and in turn – more than a single connection to each node. Since the chunk size is 1MB, the 8.54mb file is split into 9 chunks and the 50MB one – into exactly 50 chunks. This means that using two nodes, more than a single connection will be made to each node. The frequency of those connections, although, is different due to separate connection speeds between each node, resulting in the node that is closer (local node) being used slightly more frequently than the remote node. This also gives weight to discrepancy that is visible in the chart, where the jumps are not as big with additional remote nodes than a single local one.

Going further, the 4-node part shows up to a double increase in speed (in comparison to two nodes) due to connection speed being the main limiting factor in comparison to the main difference between a single local and remote node. This effectively shows, that with a stable connection to separate nodes, a stable proportional growth (with proportional increase to node

counts) can be achieved under good conditions. This is further proved by 8 node parts effectively showing a smaller, yet still noticeable (up to 30% speed increase in comparison to doubling of the node counts). This means that there is a download rate increase going further, but it has its own drop-offs due to different factors.

First and foremost, the entire download speed while manageable is not entirely ideal, higher speeds up to the theoretical limits of the download speed provided by the client's ISP provider can be achieved. The main reasoning behind this can be attributed to additional processing power required by the packet and operation system as well as public-private key encryption and decryption. This could be improved by creating a separate packet system which would just serve file requests and perform no additional processing or encryption. In other way, this can also be achieved by serving the files under more popular file transfer techniques, such as FTP.

The additional drop-off when adding more nodes to connect to can be attributed to the fact that the client program runs in a single thread. Each connection is done in an asynchronous fashion and performs other actions to other nodes while mainly waiting for IO operations (such as connection, packet transfer and file writing). Adding decryption/encryption methods that run in a synchronous fashion on top this results in a less-than-perfect scenario which keeps certain actions for different nodes being delayed while longer synchronous actions are in progress. This can be improved by further splitting asynchronous actions into tasks that can be run in parallel (ex. by using managed thread pools), or just splitting the connection processes entirely into separate threads (although this would also negatively impact mining process, if the node itself wants to perform mining as well, since the process runs in a threaded manner already).

The last part could be also limited by the theoretical connection speed limit. However, this test was done on each node being provided between 1Gbps and 10Gbps speeds which are more than enough for handling such file transfers. This means that the connection speed bottleneck was not achieved, and the speed can be increased considerably further (in a perfect scenario: up to the connection limits provided).

8.7.3. Storage control and encryption

The encryption is also used during the creation of the file. To add a file, a method is called which takes a new file, encrypts it using the node's RSA2048 public key by splitting it into 256-byte (245 bytes for file data, 14 bytes for RSA padding) blocks, and then finally saves it in the same folder that is used for local file storage. The public key is used for privacy and data safety: the node will be the only one that is able to decrypt the file themselves using their own private key. Then, a transaction is created with the operation that is meant for file creation. This operation is then given parameters of the file's main data: it's identifier (currently same as the file's hash),

its size, its file name, and finally the node's public key is added to be recognized as the owner. This transaction is then signed and broadcasted to other nodes to be received. Afterwards, after the transaction is added onto the following block and successfully confirmed/mined, the file gets actively queried by other nodes in order to be sent over the network. This way the file spreads around the blockchain and becomes widely available to be downloaded on demand without sticking to a single point. Also, since the file was encrypted beforehand, it eliminates the extra need of encryption during the file transfer process, removing the bottleneck of having to encrypt the sent file blocks on the fly, on top of the sending/receiving process.

The nodes also have the possibility to remove (delete) their files from the blockchain. To remove a file, its identifier is taken by the node that is owning it. Then a transaction operation (delete file) is created with the parameters being the identifying hash of the file. This transaction is then again signed and broadcasted to other nodes to then be added into a following block and finally mined. After the block is mined, and its operations are executed, all the nodes that have the file currently available, remove it from their databases, and if it is available – deletes it from the local storage.

Furthermore, privileged users (such as administrators) can also perform file deletions by adding a transaction with a privileged node identifier (as from the administrator list that can be assigned using privileged transactions in the ledger). This allows for deleting files in cases such as violating the network's policies or simply being duplicate or old. This can be extrapolated to work with a watchdog program which would have its own privileged node client and look for old or duplicate files in order to delete and constantly free up some space upon the network. While the methods for accomplishing such a task can be various, one way could be implementing a timed check that once every few hours checks for duplicating file hashes appearing in the ledger and then deleting the duplicate files using the according transactions.

8.8. Future improvements

- **Improve file download process.** Instead of using separate encrypted communication packets for every single file request, file requests could be done in bigger chunks using non-encrypted continuous transmission, maybe even using separate protocol such as FTP, given that those files would be encrypted before in the file storage. This would decrease the amount needed for file packet processing, encryption and decryption, and overall improving possible speeds that would help with reaching better file download performance with a lower amount of processing nodes.

- **Use variable file chunk sizes.** Currently, the system uses a hardcoded file chunk limit of a 1MB as set by the configuration file. In a better implementation, chunks could be split by a dynamic file size depending on factors such as connection speed and quality for different nodes. This would allow using more nodes for smaller files as well as giving bigger chunks to the nodes that provide better speeds. Such an idea would overall improve the download speed.
- **Create a usable UI.** Currently the application is done more as a proof of concept, and even though most of the base file system functionality is implemented, the current system is a console screen (see appendix 1) and it is not straightforward and would be confusing for any user that would try to use this in a real scenario. This is why a better way of controlling the application should be implemented, which in this case would be best done as creating a user interface which would allow for quick file system and permission control. To be able to use such an interface on many devices with a responsive and easy-to-use design language, it could be done using a frontend web application, which would then use API communication with the main application to provide visible control and access to separate functionality, that involves the blockchain file system.
- **Use a different local storage encryption method.** Encrypt storage data with something else other than base64. Currently, for simplicity's sake, the local database and node list data is stored with XML format that is then encoded on top with a base64 encoding. This is not a fool-proof solution for those that would want to modify the data, because base64 is an encoding that can be freely reversed. Another encryption method needs to be used in order to provide better security against possible cached file manipulation. Some of the suggestions may be just using the stored RSA2048 public/private key as one way of encryption, or instead, using some other key-based encryption algorithm (ex. Twofish).
- **Variate mining task difficulty.** One of the purposes of a mining task is to reduce and control the number of blocks being generated by the network. Giving a static task difficulty according to the currently planned network size and performance limits may seem enough in a short term, but in a longer term – a variable solution is needed (ex. difficulty controlled by an administrator or the entire network, depending on recent mining speeds). This is required due to quickly increasing computing performance as well as the possibility of the network size expansion, resulting in more nodes being more likely to find the hash and so overall – in decreasing mining difficulty/mining time of the network.

CONCLUSIONS

After analyzing the implementation of proposed system solutions, a few conclusions can be made:

1. A random number generator algorithm based on unique identifier seeds and percentage value ranges can be successfully used to quickly find out nodes containing certain files.
2. Data access control can be achieved on a blockchain network by storing additional data containing privileged operations (such as file deletion or user banning) inside the network.
3. Parallel downloading from multiple sources at once allows for speed increases and decrease in host node congestion. However, additional processing (such as packet encryption) during download should be avoided, considering it can result in a significant download speed degradation.
4. Variable mining task difficulty should be used in bigger networks, to keep average mining speeds in consideration to overall network growth and future technological improvements.

Overall, the proposed solutions are applicable to a blockchain network, and they successfully improve upon the issues raised in the project.

REFERENCES

[AOA22] Azbeg, Ouchetto, O., & Jai Andaloussi, S. (2022). BlockMedCare: A healthcare system based on IoT, Blockchain and IPFS for data management security. *Egyptian Informatics Journal*, *23*(2), 329–343. Page 331. https://doi.org/10.1016/j.eij.2022.02.004

[Aus20] Austria, P. (2020). Analysis of Blockchain-Based Storage Systems. *UNLV Theses, Dissertations, Professional Papers, and Capstones*. 3984. http://dx.doi.org/10.34917/22085740

[BP19] Bashir, I. & Prusty, N. (2019). *Advanced blockchain development: build highly secure, decentralized applications and conduct secure transactions*. Packt Publishing.

[CDP20] Choo, Dehghantanha, A., & Parizi, R. M. (2020). *Blockchain Cybersecurity, Trust and Privacy* (1st ed. 2020., Vol. 79). Springer Nature. Page 166. https://doi.org/10.1007/978-3-030-38181-3

[CZS22] Chen, Zhang, X., & Sun, Z. (2022). Scalable Blockchain Storage Model Based on DHT and IPFS. *KSII Transactions on Internet and Information Systems*, *16*(7), 2286–2304. https://doi.org/10.3837/tiis.2022.07.009

[DT22] Daniel, & Tschorsch, F. (2022). IPFS and Friends: A Qualitative Comparison of Next Generation Peer-to-Peer Data Networks. *IEEE Communications Surveys and Tutorials*, *24*(1), 31–52. Page 40, https://doi.org/10.1109/COMST.2022.3143147

[Elr19] Elrom. (2019). *The blockchain developer: a practical guide for designing, implementing, publishing, testing, and securing distributed blockchain-based projects*. Apress.

[IAA21] Idrees, Agarwal, P., & Alam, M. A. (2021). *Blockchain for Healthcare Systems: Challenges, Privacy, and Securing of Data*. Taylor & Francis Group.

[KS19] Kravchenko, P. & Skriabin, B. (2019). *Blockchain and Decentralized System*s. Distributed Lab.

[KYZ22] Kang, P., Yang, W., & Zheng, J. (2022). Blockchain Private File Storage-Sharing Method Based on IPFS. *Sensors (Basel, Switzerland)*, *22*(14), 5100. https://doi.org/10.3390/s22145100

[LDC+17] David Lee, Robert H. Deng / David LEE Kuo Chuen, Robert H. Deng. (2017). *Handbook of Blockchain, Digital Finance, and Inclusion, Volume 2: ChinaTech, Mobile Security, and Distributed Ledger* (1st ed.). Elsevier Science.

[Nai22] Nair, R. (2022). Blockchain-Based Decentralized Cloud Solutions for Data Transfer. *Computational Intelligence & Neuroscience*, 1–13. https://doi.org/10.1155/2022/8209854

[VJP+22] Venčkauskas, Algimantas, Jusas, Vacius, Paulikas, Kęstutis, & Toldinas, Jevgenijus. (2017). Methodology to investigate BitTorrent sync protocol. *Computer Science and Information Systems*, *14*(1), 197–218. https://doi.org/10.2298/CSIS160212032V

[Vuk16] Vukolić (2016). The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication. *Open Problems in Network Security*, 112–125. Page: 113. https://doi.org/10.1007/978-3-319-39028-4_9

[ZZ16] Zhu, & Zhou, Z. Z. (2016). Analysis and outlook of applications of blockchain technology to equity crowdfunding in China. *Financial Innovation (Heidelberg)*, *2*(29), 1–11. https://doi.org/10.1186/s40854-016-0044-7

Appendix 1. Blockchain file system program log example