UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Saeid Mousavifar

An Improvement for The Decentralized Privacy System Using Random Linear Network Coding

Master's Thesis (30 ECTS)

Supervisor: Rune Hylsberg Jacobsen, Associate Professor
Supervisor: Satish Narayana Srirama, Professor
Supervisor: Ali Marandi, Postdoc

Tartu 2020

An Improvement for The Decentralized Privacy System Using Random Linear Network Coding

Abstract:
With the constant rise of applications, there is a huge amounts of generated sensitive and private data for each person. Hence, services need to store the generated data in a cloud or distributed hash table. Two of the main issues with external storage is privacy and security of the stored data. The privacy of such as data is preserved by implementing a permission Blockchain on top of the distributed hash table that grants the access for a user's data to allowed services. However, the security of the data is only achieved by symmetric cryptography which is not a strong security mechanism. In this work, we apply a network coding scheme to this setup to achieve the goal of maintaining the security of the stored data. Our analysis show that by implementing random linear network coding in this setup, we achieve the security of stored data, as well as improving resiliency and retrieval time of the stored data with the expense of storage overhead and storage time. Our simulation results show that the expected retrieval time of the data is increased significantly while the expected storage time is increased with respect to the traditional setup. it also show that there is a trade-off between expected retrieval time and expected storage time in the system. These results confirm that our framework achieves the desired goal of making a faster, more resilient and secure setup for storing sensitive data with the requirement of slightly more storage size.

Keywords:
Peer-To-Peer(P2P) Network, Distributed Hash Tables (DHT), Random Linear Network Coding(RLNC), Permissioned Blockchain

CERCS: CP170, Computer Science

Detsentraliseeritud privaatsussüsteemi täiustus, kasutades juhuslikku lineaarset võrgu kodeerimist

Lühikokkuvõte:

Mobiiliteenuste kasutamise pideva kasvuga on iga kasutaja jaoks loodud tohutul hulgal tundlikke ja privaatseid andmeid. Seega peavad teenused salvestama loodud andmed pilve või hajutatud räsitabelisse. Kaks välise salvestuse peamist probleemi on salvestatud andmete privaatsus ja turvalisus. Selliste andmete privaatsus säilitatakse, rakendades hajutatud räsitabeli kohale lubade plokiahela, mis annab juurdepääsu kasutaja andmetele lubatud teenustele. Andmete turvalisus saavutatakse siiski ainult sümmeetrilise krüptograafia abil, mis ei ole tugev turvamehhanism. Selles töös rakendame sellele süsteemile võrgu kodeerimise skeemi, eesmärgiga säilitada salvestatud andmete turvalisus. Meie analüüs näitab, et juhusliku lineaarse võrgu kodeerimise rakendamisel selles süsteemis saavutame salvestatud andmete turvalisuse, samuti parandame salvestatud andmete vastupidavust ja väljalaadimisaega, üldkulude ja säilitusaja arvelt. Meie simulatsiooni tulemused näitavad, et andmete eeldatav otsinguaeg on märkimisväärselt pikenenud, samal ajal kui eeldatavat andmete säilitamisaega pikendatakse võrreldes traditsioonilise süsteemiga. See näitab ka, et eeldatava väljastusaja ja eeldatava süsteemi säilitamisaja vahel on kompromiss süsteemis. Need tulemused kinnitavad, et meie raamistik saavutab soovitud eesmärgi - tundlike andmete salvestamiseks kiirem, vastupidavam ja turvalisem süsteem - pisut suurema salvestusmahu nõudega.

Võtmesõnad:

Peer-to-Peer (P2P) süsteemid, Hajutatud räsitabel (DHT), Juhusliku lineaarse võrgu kodeerimine (RLNC), plokiahel

CERCS: CP170, Arvutiteadus

Acknowledgement

I would like to express my special gratitude to main supervisors Mr. Rune Hylsberg Jacobsen and Mr. Ali Marandi in Aarhus university who made my mobility possible during my thesis. Furthermore I would like to thank Mr. Satish Srirama who undertook to be my home supervisor at Tartu University and helped me greatly during my studies.

I would also like to express my deepest gratitude to Hadi Sehat who helped me beyond measures in this thesis and also during the darkest moments of my life.

Also, Greats thanks to Toomas Ginter who helped with the Estonian translation of this thesis.

Contents

1 Introduction

Due to recent improvements of technology people have increasingly used devices and applications to engage in different social activities and commerce. The increasing use of technological applications with large number of users worldwide has resulted into generation of huge amount of data which can not be stored locally [Pes13]. Motivated by this explosion of data, new forms of data collection has appeared by different service provides. In these methods, the data of each user is stored in a third party cloud or distributed hash tables [HLZ10]. The privacy and security of the data stored in the external storage has been a major research topic in recent years. A very difficult challenge is meeting the expectations of users regarding their data privacy, as the user's privacy expectation varies depending on the data type collected and context of use [Goe14]. One of the storage systems that is used extensively in recent decade are peer-to-peer (P2P) storage systems [Pes13]. Using peer-to-peer storage systems remove the need of a trusted centralized storage, however, As the users do not have authority over the storage systems, they are deprived of full control over the privacy their own data [CL11]. In this chapter, we first take a look at peer-to-peer storage systems and then discuss the possible solutions to enhance the privacy and the security of such storage systems.

1.1 Peer-to-peer storage systems

Peer-to-peer distributed storage systems are an alternative way to achieve a storage solution with high reliability and lower cost compared to the traditional server-side or cloud storage models. The advantages of using a peer-to-peer network for data archival is that, by nature, it can distribute the information into different locations and has a high potential to be scalable [Pes13]. for the sake of better understanding the differences between Peer-to-Peer and client-server model of storage, a high level architecture of both systems are illustrated in the Fig 1. One of the bold differences between these two systems are the degree of decentralization.

Decentralization means that the architecture does not rely on a central point of authority who has full access to the data and can control the access permissions, i.e., decentralization means the decision making is performed independently by all the participating nodes, instead of relying on a single node to make the decision [Pes13]. Decentralization has been been used in many applications recently, Blockchain [Swa15] is a perfect representation of a decentralized network, where the decision making and governance is made by all the participating nodes in the notwork.

Each architecture has its own advantage and disadvantages. For example in the centralized architecture the system faces a single point of failure, meaning that the network would be unavailable in case it runs into a problem or if someone takes it down [HLZ10]. another disadvantage for this architecture is that, by nature it poses more security threat compared to the decentralized networks due to the fact the system has a

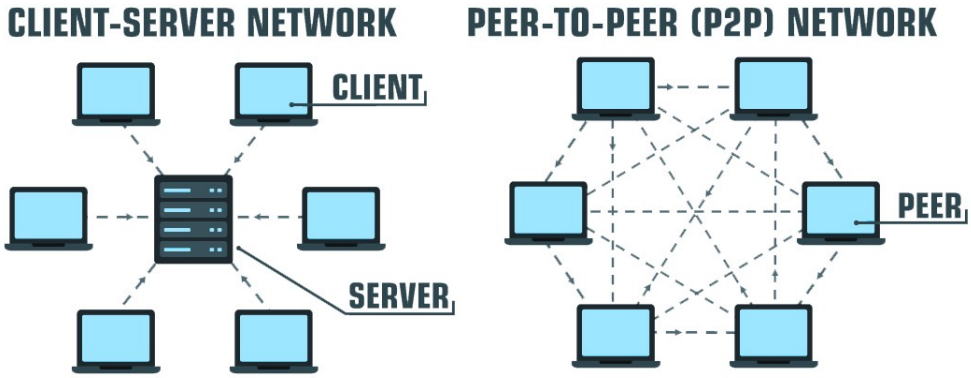


Figure 1. The high level architecture of a traditional client-server vs Peer-to-Peer Network [LLC99]

single or a few points of authority which are subjected to intrusion [LT11]. There are various use cases in which centralized storage systems are preferred, e.g., Implementing patient record information system, where having a central and unified authority over the data is preferred [LT11].

Unlike the centralized architecture, the participating peers or users in the decentralized architecture do not rely on a single point of authority. Instead, they collectively and collaboratively form a service themselves with each peer doing a fraction of the work to ultimately solidify a unified service [Pes13]. The decentralized networks have many interesting advantages that can makes these systems ideal for many use cases. These advantages come from the nature of decentralized architecture. One of the most important advantage is the elimination of central authority which leads to more security in the system, since the sensitive data for example is distributed among the participating peers, it is less unlikely that an adversary can intrude the system or perform other type of malicious activities i.e. Denial of Service (DoS) attacks [WDMS17].

As mentioned, the usage of peer-to-peer storage to store user's data has many advantages over centralized storage systems. However, the privacy of the stored data is still a major concern as the user may not be part of the peer-to-peer storage and due to the nature of such systems and lack of centralized authority, the user can not control its own data in a traditional way.

1.2 Handling privacy in peer-to-peer storage systems

To address the privacy issue, many solution have been proposed both in legislation such as GDPR [Alb16] and academic research. A promising scheme is proposed in [ZN$^+$15]. The authors of [ZN$^+$15] introduced a permissioned Blockchain on top of distributes storage system where the data of users is stored. This permissioned Blockchain allows

the users to control the access of service providers over their data, i.e., the user gives access or revoke access to the stored data at any given time. Hence, in this system the users ensure that their privacy is met and the system is more transparent compared to the existing central systems where users do not know how their data is handled.

Blockchain, as the name suggests is a chain of blocks. In a Blockchain a block represents a node where there is a cryptographic link between particular nodes. Each block contains a hash of previous block and a transaction data. This link makes the block immutable [Swa15]. Due to this structure of Blockchain, it is implemented in a P2P network where all blocks adhere to a certain protocol in order to perform inter-block communication and in case of this work, decide whether or not to grant permission to an identity. Although Blockchain is mainly used in cryptocurrency systems [W+14], there has been implementations of Blockchain to use for identity and access management, such as [MJ18]. This type of Blockchain is used in this work to ensure the privacy of data stored in the P2P storage system. Chapter 3 has more details on integration of Blockchain and peer-to-peer storage systems and how this setup ensures the privacy of the data stored in peer-to-peer systems.

1.3 Handling security in peer-to-peer storage systems

Using peer-to-peer systems in general add a great amount of security to the stored data as mentioned in section 1.1. However, there are still security issues that need to be handled in the certain setups such as the one proposed by [ZN+15]. One of the major issues in such systems is that the system stores raw data of each user in the P2P network, which is susceptible to certain attacks such as byzantine attacks [CT06].

Although by current advances in the computational power allows the nodes to use cryptographic methods to increase the security of data they store in the P2P storage system [MLFR01], the nature of peer to peer systems adds the possibility of security breach in terms of malicious nodes [Pes13]. This issue can become a stronger problem if the cryptographic method used for encryption of data is not unconditionally secure [LT11].

1.4 Motivation

In this work, we work on the basis of a privacy enhancing P2P storage system proposed in [ZN+15]. The privacy of this system is preserved by implementing a Blockchain on top of the P2P storage system. However, the security of data is preserved by a simple symmetric cryptographic method. This method adds a weak amount of security to the data stored in the P2P storage system, but is vulnerable to certain attacks such as man-in-the-middle or Byzantine attacks [KL14].

In this work, we propose the usage of network coding in order to ensure the security of the stored data in the DHT. We propose using random linear network coding, i.e., RLNC [HMK+06a] to code the data before storing in the DHT. As network coding is

a weak security (is inherently secure) by itself [BN+05], it is trivial that implementing RLNC provides increased security for the stored data. Furthermore, in the setup we use in this work, we have the advantage that the data is transferred to DHT from the system which is outside the DHT. Hence, we can use network coding prior to sending data to DHT and store coded packets in DHT nodes. Using RLNC gives us two major features:

- In the case of presence of one or multiple malicious nodes in the DHT Nodes, the data is still secure. In order to retrieve the original data, the malicious nodes must have access to k coded packets. In other words, the malicious nodes must hold at least k coded packets of a single data. The probability of this event is negligible.
- The nodes can perform re-coding to relocate data [HPFM11], instead of traditional broadcasting data to neighboring nodes. In the re-coding procedure, innovative coded packets are generated that are linearly independent to other coded packets already stored in the DHT. Therefore, the re-coding property further increases the reliability of storage, because increasing the number of linearly independent coded packets increases the number of nodes that can leave the network without eliminating the possibility of retrieving the original data.

1.5 Contribution

In this thesis, we propose an RLNC scheme on top of the system introduced in [ZN+15]. It is trivial that RLNC increases the security of the stored data. We then analyze the performance of the system in terms of expected retrieval time, resiliency and storage. The results show that our system improves in terms of expected retrieval time and resiliency with the expense of larger required storage.

Our contribution is to add a layer of network coding using RLNC on top of a Blockchain-DHT system to improve the following metrics:

- The average Retrieval speed of data from Kademlia.
- Security of the stored data in case of presence of malicious nodes or eavesdroppers.
- Balanced data load on each peer in Kademlia.
- Improved resiliency.

1.6 Outline

The outline of this thesis is as follows. In chapter 2, we first point out some related work on peer-to-peer storage systems, Blockchain and network coding solutions for privacy and security of big data applications. Then we propose our system model and

theoretical framework of our proposed method in chapter 3. In chapter 4, we analyze the performance of our system. In chapter 5 we validate our system using simulation technique and finally chapter 6 includes conclusion of the thesis.

2 Related works

In this chapter, we mention some related work on the peer-to-peer storage systems and using permissioned Blockchain and network coding in peer-to-peer storage systems. The goal of this chapter is to familiarize with the concepts we used in this thesis.

2.1 Peer-to-Peer distributed storage systems

We first look at some major characteristics of peer-to-peer storage systems, then in the next subsection, we focus on Kademlia [MM02], which is the focused peer-to-peer storage in this thesis.

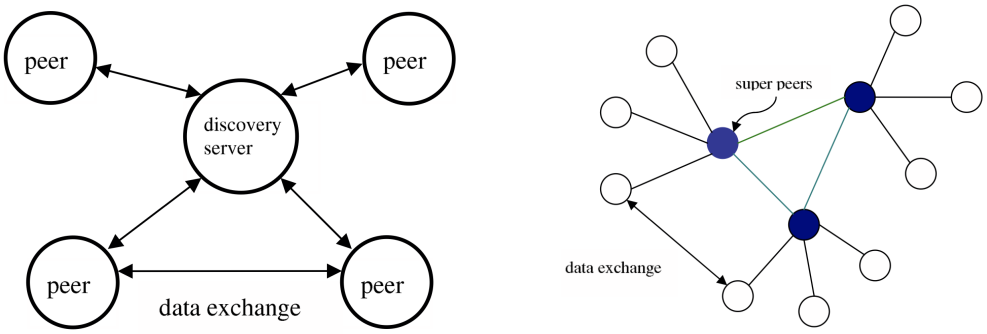
2.1.1 P2P Architecture

Since decentralization is a major characteristic of P2P networks, based on the degree of decentralization we can classify these networks into two categories [KMM+02]:

1. **Purely Decentralized:** Pure P2P systems are networks without any centralized control. In such systems all the nodes are equivalent in functionality, In other words, in these networks, nodes are both server and client at the same time. Gnutella [RFI02], Freenet [CSWH01], Chord [SMK+01a], CAN [RFH+01] and Kademlia are example of such systems. These systems are inherently scalable due to the fact that such system try to avoid central entities or servers. These kinds of systems are also inherently fault-tolerant, since there is not any central point of failure and depending on architecture, changes in the network such as loss of a peer or a number of peers can easily be recovered. They also have a great degree of autonomousity over their data or resources. On the other hand such systems usually suffer from slow information discovery and lack of guarantee about quality of service. Also due to the lack of a global view at the system, it is not easy to predict the system's behavior [KA17].

2. **Hybrid Architecture:** In hybrid P2P systems, there is a central entity that contains directories of information about the registered users in the network, these data are in the form of meta-data. There are two type of hybrid systems - centralized indexing and decentralized indexing [YGM03]. In centralized indexing as can be seen in Fig. 2a, a central server keeps an index of the data that are currently being offered by the active peers. To make a query, each peer should contact the central server to send the query. Such systems with the central servers are simple and operate quickly and efficiently for information discovery. On the other Because of central servers operating in the network, they are single points of failure that make them vulnerable to malicious attacks. These systems are not inherently scalable, due to the limitations on the size of the central servers database and the capacity

to respond the queries. The directories also have to be periodically updated. In decentralized indexing, a central server is responsible of registering the users into the network and further facilitating the peer discovery process. An example can be seen in 2b. In these systems a number of the nodes have a more important role than the rest of nodes. These nodes are called ”supernodes” [YGM03]. These nodes are responsible to index the information shared by local peers connected to them and do proxy search on behalf of these peers. The queries are thus sent to SuperNodes, not to other peers. Kazaa[GK03] is an example of such system. In such systems supernodes are collaboratively elected if they maintain the sufficient bandwidth or processing power [NS06].



(a) centralized indexing (b) decentralized indexing

Figure 2. [PBV05]

2.1.2 P2P Network Discovery Mechanism:

Distributed P2P systems often require a discovery mechanism through which the nodes can locate specific data in the system. P2P systems have evolved through three generation of discovery mechanism as follows [KMM+02]:

Centralized indexes and repositories: This mechanism is mainly used in hybrid systems. In this model, the peers connect to a centralized directory server, which store all the information regarding the location of data. Upon request from a peer, the central server will match the request against all the peers in it’s database and chooses the best peer that matches the request (e.g., the node that has the requested data). The best peer based on the application can be the one that is cheapest, fastest, closest or most available. After the central server responds to the query, the data exchange will happen directly between the two peers. Napster uses this method [AT02]. A central server keeps:

- The index with meta data (file name, time of creation and etc.) of all files available in the network,

- A table containing the information of the users (IP addresses, connection speeds etc.)
- A table indexing the files each user holds in the network.

At the beginning, the new client contacts the central server and reports the list of files it contains, Later, when the server receives a query from another peer, it looks for matches in its index, and returns a list of users holding the matching file. The user then initiates a direct connection with the peer holding the requested file, and downloads it [Pes13].

Flooding broadcast of queries: This model is mainly seen in purely decentralized P2P models. In these networks, since no peer has the overall top view of the network, for resource discovery, each requests from peers are flooded (broadcasted) to the directly connected peers, the receiving peers would also flood the request to their connected peers until the request is answered or a maximum number of flooding steps has been reached. Different broadcast policies have been implemented to improve the search in the P2P networks [YGM02], [KKE03], [TR03]. The early architecture of Gnutella [Rip01] uses the flooding broadcast to locate the files in the network. There are four types of messages in the Gnutella protocol as follows [Rip01].

- **Ping:** A request for a specified peer to announce itself.
- **Pong:** Reply to a Ping message which contains the IP and port of the responding peer and information about files shared.
- **Query:** A search request. It contains the search string and the minimum speed requirements of the potential matched peers.
- **Query hits:** Reply to a Query message. It contains the IP, port and the speed of the matched peer.

After joining the Gnutella network, a node sends out a Ping message to any node in its neighbourhood. The nodes in return send back a Pong message to identify themselves, and also propagate the ping message to their neighbors. to prevent loops Gnutella uses time-to-live(TTL) flooding to control the spread of these messages. At each hop the value of the field time-to-live(TTL) is decreased by one, and the message is dropped when the counter is reached to zero. Once a hit is found, the two peers form a direct channel to transfer the data. These systems do not scale well and the accurate discovery of peers is not guaranteed when using flooding. Also using TTL creates virtual regions that limits users to it meaning that, it creates a limit that users cannot send request beyond that reach. If on the other hand the TTL is removed, the network would be flooded with unwanted requests [Rip01].

Routing Model: The routing model gives more structure to the placement of information by using a distributed hash table. These models have a clear mapping between the resource identifier and location of the resources in the network, in the form of a distributed routing table [Pes13]. The look-up operation is realized by organizing the peers in a structured overlay network, and routing a message through the overlay to the responsible peers. Several system such as Freenet , Chord , Content addressable networks (CAN) , Pastry [RD01] and Kademlia use this model.

2.2 Kademlia

Kademlia is a decentralized P2P Distributed Hash Table based on XOR distance metric. Kademlia has been used as the storage solution of many systems such as [ZN$^+$15]. Kademlia offers a flexible routing table and minimizes the number of configuration messages sent among the nodes for locating contents [MM02]. Due to this characteristic and effective routing time of Kademlia, it is one of the best choices to use for storing application data [Pes13]. Therefore, we use Kademlia in our system as the P2P storage system. In this section, we point out major properties of Kademlia.

2.2.1 Kademlia system model

Kademlia is a system of interconnected nodes which are forming a P2P storage network. The participant nodes in Kademlia each have a unique ID, which is a 160-bits long random numbers [MM02]. Kademlia leverages this assumption to use a simple algorithm for locating the data by treating each node as a leave in a binary tree, in which the position of each node is determined by the shortest unique prefix of its ID as shown in Fig. 3. This illustration helps us to understand the functionality of Kademlia in a better way.

Any value that needs to be stored in the Kademlia is assigned by a 160-bit long key. The important aspect here is that the keys generated for each value and the node IDs must belong to the same key space, which is the space of 160-bit long values in this case. The <Key,Value> pair is then stored in the node whit the closest ID to key. Closeness in Kademlia is described in the next section.

2.2.2 Notion of closeness

The particularity of Kademlia is the use of a novel bit-wise Exclusive Or (XOR) metric in order to calculate the distance between points (the node IDs) in the key space [MM02]. Given two 160-bit identifiers, i.e., NodeID or key, x and y, Kademlia defines the distance between them as follow:

$$d(x, y) = x \oplus y$$

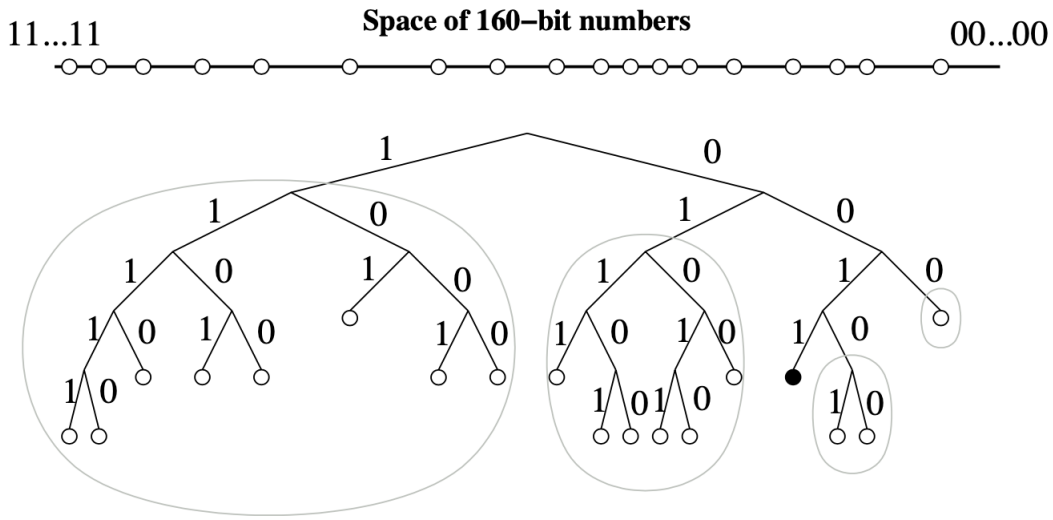


Figure 3. The ID of the Nodes in Kademlia, The black dot shows the location of the node 0011 and the grey ovals show the subtrees corresponding to different k-buckets [MM02]

XOR metric was chosen because it presents some common properties with the geometric distance formula, in particular [MM02]:

- The distance between a node and itself is zero.
- It is symmetric, i.e., $d(x, y) = d(y, x)$
- It offers the triangle property: i.e., for points x, y and z, we have:

$$d(x, z) \leq d(x, y) + d(y, z)$$

Using this notion, when we want to store a <key,value> pair in Kademlia, this pair needs to be stored in the node p that ID_p has minimum distance with key, i.e., $\min_p d(ID_p, key)$.

2.2.3 Routing in Kademlia and $\mathcal{K}$_buckets

Every node has a routing table as can be seen in Fig. 4. This table consists of 160 rows. Each rows consists of $\mathcal{K}$ entries, where each entry is information of a node in Kademlia, stored as a triplet <INodeID, IP Address, UDP Port>. This information contains the necessary information in order to find a node. This list is called $\mathcal{K}$-buckets in Kademlia literature [MM02]. The value of $\mathcal{K}$ is a system-wide replication parameter which defines

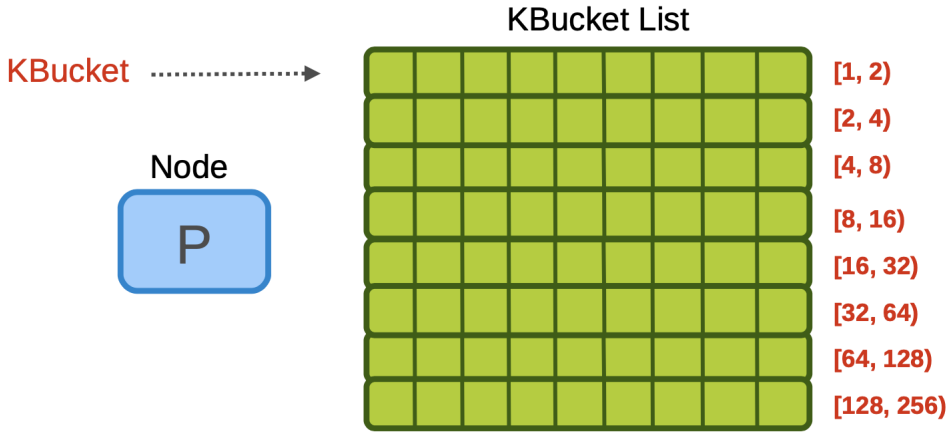


Figure 4. Kademlia Routing Table Structure [PH]

the upper bound to which the lists can grow, i.e., maximum number of nodes that is stored in each node.

$\mathcal{K}$-buckets are filled out with respect to the distances of nodes from each other, as seen in Fig. 4, the bucket for node P has 8 buckets each responsible for storing the information of other nodes with respect to the distance, for example if the distance node P and node Q is 8 based on the distance metric XOR, information about node Q is stored in the third row of the $\mathcal{K}$-bucket of node P.

When a node P receives a message from another node Q, it updates the specific $\mathcal{K}$-bucket for the sender's node Q. The update procedure is done as follows:

- If Q is not already in the appropriate $\mathcal{K}$-bucket and the bucket has fewer than $\mathcal{K}$ entries, then the recipient simply inserts the new sender at the tail of the list.
- If the appropriate k-bucket is full, P pings the $\mathcal{K}$-bucket's least-recently seen node R to decide what to do. If node R fails to respond, it is removed from the $\mathcal{K}$-bucket and Q is inserted at the tail of the $\mathcal{K}$-bucket. Otherwise, if R responds, it is moved to the tail of the list, and the information of Q is ignored [MM02].

Kademlia exploits the fact that the longer a node has been up, the higher the probability to remain up in the future [SMK+01b]. By keeping the oldest live nodes in the routing table, we maximize the probability that the nodes contained in each $\mathcal{K}$-bucket to be up in the network.

2.2.4 Messages

Every message exchanged between the nodes in Kademlia typically include a tuple of $<$ NodeID, IP Adress, UDP Port$>$. This ways other nodes which receive the message can

become aware of the existence of the other nodes. There are four types of messages in Kademlia.

- **PING:** Used to check if a node is alive.
- **STORE:** Used to request for storage of a <key, value> pair.
- **FIND NODE:** Used mainly for routing. it takes a 160-bit ID as an argument. the recipients of this message return the $\mathcal{K}$ closet nodes they know of, which have the closet NodeID to the requested ID.
- **FIND VALUE:** This message functions similar to the FIND NODE message, except that if the receiver of the request has the requested key in its storage, it returns the associated value.

2.2.5 Joining the network

When a new node P is joined in the network, It chooses a random Node-ID from the key space, then it contacts a Node Q, and updates the appropriate $\mathcal{K}$-Bucket with the information of node Q. Then Node P starts a Find Node procedure for its own Node-ID. This procedure results in finding the nodes that are closest to the P. By the end of this procedure, node P receives the information on the closest nodes to itself, i.e., its neighbors. then P contacts its neighbors and upon receiving an answer from these nodes, adds them to its $\mathcal{K}$-Bucket [MM02].

2.2.6 Relocation

In order to prevent loss of the data in the network in the case of peers leaving the network. After a node stores a data, it starts a timer which is defined in the system. After the timer for that particular data expires, the node broadcasts the stored <Key, Value> to the $\mathcal{K}$ closest nodes in the network, i.e., its neighbors. This relocation ensures that likelihood of a data being missed in the network is negligible [MM02].

2.2.7 Storing a data

In order to store a file into the Kademlia network, the user does the following on the value v it wants to store.

- The user uses a hash function $\mathcal{H}$ to hash the value of the value v into a key k to form <k, v> .
- The user finds the node P whose ID is closest to the k using Find Node message.
- The data is then stored in the node P using Store message.

To Retrieve a data, the user must know the assigned key value k, then it executed the following procedure.we first have to find the nodes which are closest to the key of the data being searched.

- The user finds the node P whose ID is closest to the k using Find Node message.
- The data is then retrieved in the node P using Find Value message.

we can see that in both cases, the most important part is to find the closest node to the mentioned key. In the following section, we illustrate how this procedure is executed.

2.2.8 Routing

The procedure of finding a node that is closest to a given key is called lookup. The search initiator picks α nodes from the respective $\mathcal{K}$-bucket. Then the initiator starts to send parallel, asynchronous FIND NODE messages to the α nodes it has chosen. The value of α, which defines the degree of simultaneous messages that are sent in the network is usually defined three in practical setups. [Pes13]. Figure 5 illustrates and example in which node P is finding Q.

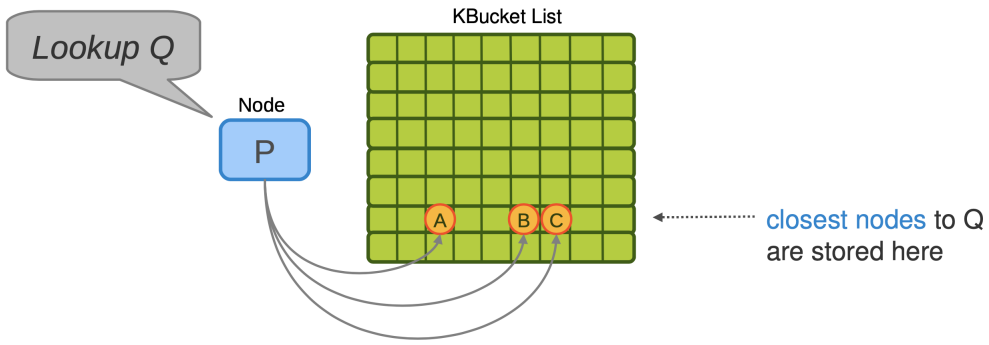


Figure 5. The lookup procedure in a node P [PH]

When the recipient nodes A, B and C receive the request, each of them look in their own $\mathcal{K}$-buckets and will similarly return the $\mathcal{K}$ closest nodes they know to the desired key. Upon receiving the results from the nodes A, B and C, node P selects α nodes from the received information as can be seen in fig. 6. In this case, M, N and O are the chosen nodes. P will then send Find Node to M, N and O. This procedure is iterated until either Q is found or no new nodes closer to Q are found in an iteration of the algorithm. In the latter case, the lookup fails as Q can not be found in the network [MM02].

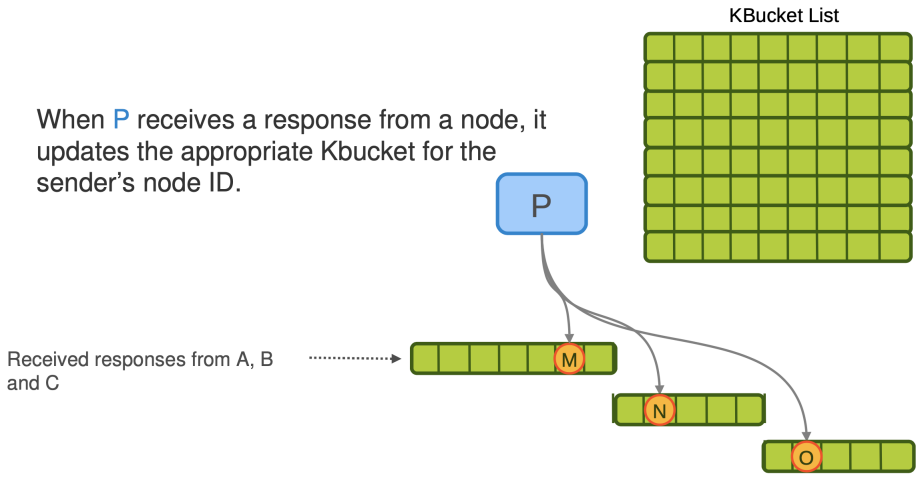


Figure 6. The iterative step of the lookup procedure in Kademlia [PH]

2.3 Permissioned Blockchains

As mentioned in section 1.2, Blockchain is a decentralized network mainly used in cryptocurrencies [W+14] A Blockchain can be perceived as a chain of blocks that are linked with a cryptographic protocol. As this blocks are distributed between the nodes in the network, there is no requirement for a centralized authority [Swa15]. An example of a Blockchain structure can be seen in Fig.7.

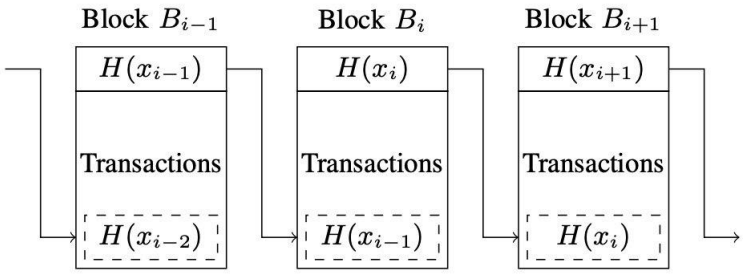


Figure 7. An example of a Blockchain Structure[MJ18]

In a Blockchain network, the participating nodes agree on a set of rules known as smart contracts. A smart contract is a set of functions that are executed on the entities in the distributed ledger, i.e., Blockchain. The functions on smart contract is being agreed on by the nodes in the Block chain and is converted to a computer code, which is then stored and replicated in the Blockchain [MJ18]. In our work, this smart contract contains rules for authentication and authorization . This means that when an external identity tries to execute a certain functionality, the validation process queries the Blockchain

for permission status and grants authority for the identity if the majority of the ledger validate the authority of the user as positive [MJ18]. Fig. 8 illustrates this validation protocol in a simple setup.

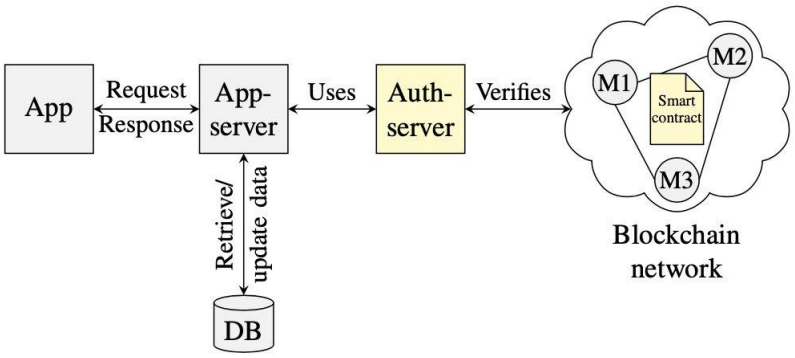


Figure 8. Architechture of an identity and authorization management system using Blockchain and smart contract[MJ18]

There are three types of Blockchain based on how they control the access given to the nodes in the ledger.

1. Public Blockchains are open for anyone to join and all transactions are visible to anyone. Public Blockchains are secured by cryptographic validation between the nodes [FLP85]. This transparency makes the Blockchain highly secure with the expense of increased redundancy and cost of transactions and decreased speed of processing [MJ18].

2. Another type of Blockchains are permissioned Blockchains. In this type of Blockchains, a control layer runs on top of the Blockchain that controls the actions that are executed by the participants of the ledger. This means that extrenal identities can not join this Blockchain unless they are approved [WG18]. Permissioned Blockchains are more cost effective and faster in processing speed. However, permissioned Blockchains are not completely distributed and therefore are less secure with respect to open Blockchains.

3. Private Blockchains are not open to public at all, which means that only a set of pre defined nodes are able to write and perform transactions in the Blockchain. This type of Blockchain is more efficient and has a higher degree of privacy. However, due to partially distributed nature of this type, it requires trust in the controlling authority [MJ18].

By looking at the characteristics of the three types of Blockchains, it is clear that in order to provide privacy, we need to use permissioned or private Blockchain. However,

private Blockchain still requires trust in the controlling authority, which removes the reasoning of using a P2P storage system. Therefore, the best choice for an identification and authorization Blockchain is permissioned Blockchain. Using Blockchain as a privacy preserving system is relatively new, as far as our knowledge there are very few researches on this matter, most notably [ZN+15] and [MJ18] are the two major researches done in this area. In this work, we are using the framework introduced in [ZN+15] in order to preserve the privacy of data stored in P2P storage system. Chapters 3.2 and 3.3 include more detailed explanation of this setup. However in our system, the application of blockcahin remains intact.

2.4 Network Coding in P2P storage systems

Network coding, mainly Reed-Solomon codes [WB99] has been used in Kademlia in order to improve the reliability of the transmission of data between peers [BKV+18]. The authors of [DGW+10] introduced the theoretical framework of using error-correcting codes in distributed storage systems to improve the reliability of the system in case of presence of error in channels. In practice, error-correcting codes are used to relocate data in distributed storage systems more reliably. Error-correcting codes improve the possibility of relocating data in a distributed storage by ensuring that the data is not lost in the case of bit-level error in the channels between nodes in the distributed storage system.There have been many practical research that show the improved performance and resiliency of distributed storage systems when using error-correcting codes. [HSX+12],[KGJØ17], [RNW+15].

In this work, we focus on Random Linear Network Coding, i.e., RLNC [HMK+06b] as our network coding scheme. RLNC is a method to code the data into K smaller chunks and then generating n coded packets ($k < n$). These coded packets are created by combing all the K chunks using a random linear combination as shown in Fig. 9. In the next subsection, we describe RLNC and how it works. After that we can argue how implementation of network coding benefits our system.

2.4.1 Random Linear Network Coding

In RLNC, a value v is divided into Given k chunks, $\{v_1, ..., v_3\}$. the encoding algorithm generates n linearly independent coded packets $\{c_1, ..., c_6\}$ by creating a linear combination of all the k chunks as follows:

$$n_i = \sum_{j=0} \alpha_{i,j} v_j,$$

where $_{i,j}$ is a uniformly random variable from a Galois Field $GF(q)$. Fig. 9 shows an example of the encoding procedure for $k = 3$ and $n = 6$

The outcome of encoding procedure is a set of coded packets alongside the coefficients. All coefficients are stored in a matrix M where $M_{i,j} = \alpha_{i,j}$. This matrix is called the coefficients matrix and is required to retrieve the original data v from the coded packets.

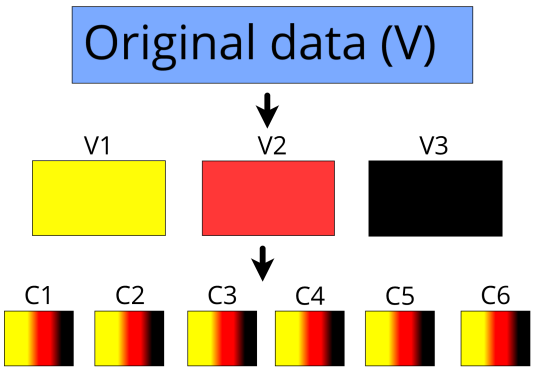


Figure 9. encoding v into coded packets using RLNC with $K = 3$ and $N = 6$

To decode the original data, we need to have the coefficients matrix and at least K coded packets. These coded packets can be put together in a system of equations with order k. The original chunks $\{v_1, ..., v_3\}$ are calculated by solving this system of equations. This procedure is called decoding. Fig. 10 illustrates the procedure to build the original data.

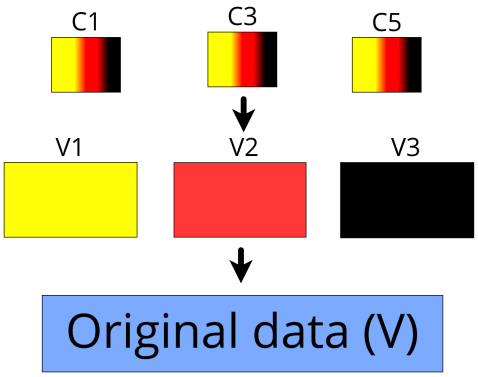


Figure 10. The user can build the original data by getting 3 independent coded packets

It should be noted that the value of n is an indicator of how redundant the data is stored in the system. More coded packets enables user to gather the required number of packets more easily while it also demands more storage capacity in the system [HPFM11].

This value is also a good indicator of how resilient a Peer-to-Peer storage system is. It is trivial that the more we spread the coded packet over different peers in a P2P system, the more system is resilient to losing or leaving nodes in the network due to the fact we only need K coded packets to build the original data.

2.4.2 Gauss Jordan Method:

In order to decode the coded packets in the receiver and generate the original packet, we use Gauss Jordan Method [HPFM11]. In this method, each coded packet is treated as a equation. The system of equations is written in form the following matrix equation, where M is the coefficients matrix , C is a vector consisted of the coded packets and v is a vector consisting the chunks of original value.

$$M_{k*k} \times C_{k*1} = V_{k*1}$$

we then use Gaussian Elimination for matrix M_{k*k} to make up an upper or lower triangle matrix to derive the matrix V_{k*1}.

It is trivial that storing coded packets instead of the original data v adds a layer of security to the system as any adversary will have to gain access to at least k coded packets in order to retrieve the original data [BN$^+$05]. However, as mentioned, the resiliency of the system against nodes leaving the network is increased as the system can tolerate losing some of the coded packets until the number of remaining coded packets are larger than k. In chapter 4 we analyze the impact of using network coding on different aspects of our system.

2.5 Summary

In this chapter we introduced the background on P2P storage systems, especially Kademlia, Blockchain and RLNC. we now move forward to introduce our system model, which integrates these three structures in a whole system to make privacy and security available in Kademlia. In the next section, we describe the system model.

3 System Model

In [ZN$^+$15], the authors introduce Blockchain [Nak19] to protect data privacy in an off DHT. The basic idea behind this system is to allow the users to adjust the privacy of their stored data by granting or provoking access to services. In this system, the permissioned blockcahin is used to control and provide access to the data. Using this system provides a number of benefits to users compared to the traditional data sharing system, the main advantage gained from this system is the fact that there is no need for a trusted third

party authority to control over access permissions. This system is the foundation of our research in this work.

In this chapter, we introduce the backbone system which is the basis of our work. we also explain in detail the protocols used in the system and later in this chapter we introduce our contribution to the system and the changes that we make in order to implement our ideas and contributions.

3.1 DHT and Permissioned Blockchain

Fig. 11, illustrates the high-level model that is proposed in [ZN+15]. As we mentioned previously, the proposed system has addressed the aspect of security by introducing a trustable decentralized element - Blockchain to control the access to the data generated by the users, as opposed to the centralized control entities who had authority over the data, which provides little transparency on how the privacy of the stored data is handled.

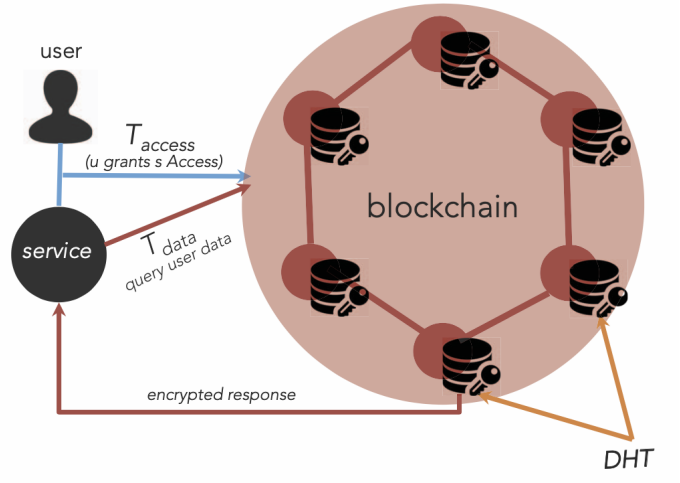


Figure 11. Overview of the decentralized platform. [ZN+15]

As seen in the Fig. 11, the overall system consists of five major components as follows:

1. **User:** The user is defined as someone who owns the data. This user can at anytime define the access for other services in the network.

2. **Service:** Defined as an entity which processes the data for the user upon users request. The service can be a tool or an application that users use to generate data, e.g., a word processing application is a service through which the users generate data.

3. **Data:** Defined in form of a pair of $<$ Key, Value $>$. where Key is a 160-bits identifier of the value being stored in the DHT.

4. **Permissioned Blockchain:** The role of this component is to control and verify accesses to the data in DHT. In the forthcoming, we describe the permissioned Blockchain in detail.

5. **Distributed Hash Table (DHT):** is a distributed storage system based on Kademlia used to store the data generated by different services. In this work, we use Kademlia as the chosen DHT implementation.

The System addresses the following aspects regarding the users' data [ZN+15]:

- **Data Ownership:** The system recognizes the owner of the file, and the owner has the control to grant or revoke access to its data. in this system, users are recognized as the owner of the files while services are the treated as guests with delegated access to these files.

- **Data Transparency and Audibility:** The users have transparency over what data is being collected from them and also how the data is accessed.

- **Fine-grained Access Control:** the users can define and alter access levels to different users at any given time, i.e., in terms of a mobile application, an application might only be allowed to access a certain or only a part of a certain data instead of having access over all the users data.

There are two types of Blockchain transactions in this system namely T_{Access} and T_{Data}:

- T_{Access}: executed when the user wants to grant access or remove an existing access to a service.

- T_{Data}: executed when a service wants to access or write a data.

In the next section we describe the protocol used in this system to enable communication between service, Blockchain and Kademlia.

3.2 The Protocols

The mentioned system uses a protocol with four procedures to achieve a privacy enhanced storage using Blockchain and Kademlia. Here we describe these procedure.

3.2.1 First Procedure:

The first procedure is to create a compound identity to uniquely distinguish the pair of (user, service) in the system. this compound identity is used by the Blockchain to validate the requests for example to access a file. when Blockchain receives a request it should be able to first distinguish which (user, service) does this request belong to.

The compound identity is a pair of public signature keys generated by the user and the system. Algorithm 1 shows a high level psudocode for this procedure. We can further break down this procedure as follows:

- User and service form a secure channel.
- User and service each generate public and private signature keys.
- User also creates an encryption key to encrypt the data.
- User and service exchange their generated public keys to each other. user also shares it's encryption key for the service so that the service is able to encrypt the data it wishes to store or decrypt the data upon access.

Algorithm 1: Generating a compound identity

Input: u, s
Result: $pk_{sig}^{u,s}, pk_{sig}^{s,u}, sk_{enc}^{u,s}$

Procedure COMPOUNDIDENTITY(u, s)
u and s form a secure channel
u executes:
    $(pk_{sig}^{u,s}, sk_{sig}^{u,s}) \leftarrow g_{sig}()$
    $sk_{enc}^{u,s} \leftarrow \mathrm{g}_{enc}()$
    u shares $< sk_{enc}^{u,s}, pk_{Sig}^{u,s} >$ with s
s executes:
    $(pk_{sig}^{s,u}, sk_{sig}^{s,u}) \leftarrow g_{Sig}()$
    s shares $pk_{Sig}^{s,u}$ with s
// Both u and s have $sk_{enc}^{u,s}, pk_{Sig}^{u,s}, pk_{Sig}^{s,u}$
return $pk_{sig}^{u,s}, pk_{sig}^{s,u}, sk_{sig}^{u,s}$
End Precedure

3.2.2 Second Precedure:

This procedure is intended for the Blockchain to check the service permission against the requested data. The procedure can be seen at Algorithm 2. This procedure consists of the following steps:

- The service requests the Blockchain to access a specific users data by providing the compound identity $pk^{(}_{sig}s, u)$ and x_p which is a variable specifying the aspect of access to the data, e.g., in case of the mobile application, this could specify the access to the contacts, photos or videos.
- The Blockchain checks the policy for this compound identity by hashing the provided key and fetching the saved policy associated with that compound key in the Blockchain memory.
- The Blockchain parses the policy from its memory in the form of $pk^{u,s}_{Sig}$, $pk^{s,u}_{Sig}$, $POLICY_{u,s}$. By doing this step, the Blockchain extracts the permissions that are granted to the service, therefore, the Blockchain can validate if the service has been granted the requested permission.
- If the request is made form the user, i.e., $p^k_{sig} = pk^{u,s}_{sig}$, the access is granted by setting the flag $s = 1$. This is because the user always has access to its own data.
- if the request is made from a service, i.e., $(p^k_{sig} = pk^{s,u}_{sig}$, then it should also be checked whether the service does have the required policy to access, the latter is check by $x_p \in POLICY_{u,s}$.
- if the service meets the conditions $p^k_{sig} = pk^{s,u}_{sig}$ and $x_p \in POLICY_{u,s})$, the flag s is set to $s = 1$, which means that the service is granted access to the data.

3.2.3 Third Precedure:

The third procedure can be seen in Algorithm 3. This procedure is used by the user to add a policy or remove an existing policy stored in the Blockchain.

- The user request the invokes the $HANDLEACCESSTX()$ function with the its compound identity key and m which is a variable to specify the policies user wants to add or rmeove.
- The Blockchain checks to see if the provided key is for a user, i.e., $pk^k_{Sig} = pk^{u,s}_{Sig}$, since a service is not allowed to add or modify any policies.

Algorithm 2: Permissions check against the Blockchain

Input: (pk^k_{sig}, x_p)
Result: s

Procedure CHECKPOLICY(pk^k_{sig}, x_p)
$s \leftarrow 0$
a_{policy} = $H(pk^k_{sig})$
if $L[a_{policy}] \neq 0$ **then**
    $pk^{u,s}_{Sig}, pk^{s,u}_{Sig}, POLICY_{u,s} \leftarrow Parse(L[a_{policy}])$
    if $p^k_{sig} = pk^{u,s}_{sig}$ ***or*** $(p^k_{sig} = pk^{s,u}_{sig}$ ***And*** $x_p \in POLICY_{u,s})$ **then**
        $s \leftarrow 1$
    end if
end if
return s
End Precedure

- if the provided key is a user side compound key, then the provided policy is saved into the Blockchain memory. it should be noted that the location index in which the policy is saved, is the hash of the value of m. Using the Hash function in the placement of data in Blockchain ensures consistency in the whole system.

Algorithm 3: Access Control Protocol

Input: (pk^k_{sig}, m)
Result: s

Procedure HANDLEACCESSTX (pk^k_{sig}, m)
$s \leftarrow 0$
$pk^{u,s}_{sig}, pk^{s,u}_{Sig}, POLICY_{u,s} \leftarrow Parse(m)$
if $pk^k_{Sig} = pk^{u,s}_{Sig}$ **then**
    $L[H(pk^k_{sig})] = m$
    $s \leftarrow 1$
end if
return s
End Precedure

3.2.4 Fourth Procedure:

The forth procedure is intended to store or load the data into Kademlia. More specifically, it is part of the T_Data transaction previously introduced. This procedure can be seen in Algorithm 4 and can be further divided into the following steps.

- The service requests to store or load a data by invoking the $HANDLEDTATX()$ function with the its compound key and m as discussed in procedure 3.
- The system parses the input m into c, $x_p$and rw. rw and c may differ as the service wants to write or read data as follows:
  - If reading data : c is the data and $rw = 1$
  - If writing data: c is the key of the data and $rw = 0$
- The system checks if the request made is legitimate, i.e., the service has access to the data by using the procedure 2.
- If the service has access, it can use Kademlia to read or write the data.
- in case of writing the data, the hash of the data is stored in the Blockchain to be used for future access control.

3.3 Proposed Model

In this section, We propose an improved protocol based on the system that was introduced in [ZN$^+$15] and described in section 3.2 and 3.3. We first introduce our system model which is an extension to the model of Fig. 11. Then we update the procedures of the protocol used in this system and point out the differences with the protocol described in section 3.2.

3.3.1 System Model

Fig. 12 illustrates the proposed system. In this system, there are a number of users and services. The services store the data in Kademlia, while the user controls the access of any service to the stored data that belongs to him. This system is an extension to [ZN$^+$15] previously introduced. The contribution in our system is the following.

Our proposed system has the following differences from the setup introduced in [ZN$^+$15]:

- The service divides the data into k chunks $\{v_1, ..., v_k\}$. Using this chunks, the service generates n coded packets $\{c_1, ..., c_n\}$, where $n > k$. Each coded packet

Algorithm 4: Storing or Loading Data

Input: (pk^k_{sig}, m)
Result: 0 or $ds[h_c]$ or h_c

Procedure HANDLEDATATX (pk^k_{sig}, m)
$c, x_p, rw = Parse(m)$
if $CheckPolicy(pk^k_{sig}, x_p) = True$ **then**
    $pk^{u,s}_{sig}, pk^{s,u}_{sig}, POLICY_{u,s} \leftarrow$Parse$(L[H(pk^{u,s}_{sig})])$
    $a_{x_p} = H(pk^{u,s}_{sig}||x_p)$
        if $rw = 0$ **then**
            $h(c) = H(c)$
            $L[a_{x_p}] \leftarrow L[a_{x_p}] \cup h(c)$
            (DHT) $ds[hc] \leftarrow c$
            return h_c
        else if $c \in L[a_{x_p}]$ **then**
            (DHT) **return** $ds[h_c]$
        end if
end if
return 0
End Precedure

is created by using k randomly generated coefficients from a certain Galois Field $GF(q)$, i.e.,

$$c_i = \sum_{j=1}^{k} \alpha_{i,j} v_j.$$

- Upon granting access to read, the service sends all coded packets to Kademlia. Each coded packet is stored with its unique key, i.e., $< key_1, c_1 >, ..., < key_n, c_n >$.
- The coefficients are stored in the service who originally generated encoded packets in order to decode the data in retrieval phase. Hence, in total, a set of n keys and $n \cdot k$ coefficients are stored for retrieval of the data. These coefficients are stored locally and is not sent to the Blockchain or Kademlia.
- If a service wishes to retrieve the data, it asks for the set of keys and the coefficients associated with the coded packets of the data it wishes to retrieve from the service that has stored the data.
- After the service acquires the set of keys, it asks the Blockchain for permission, upon granting the permission to access Kademlia, the service starts to retrieve the

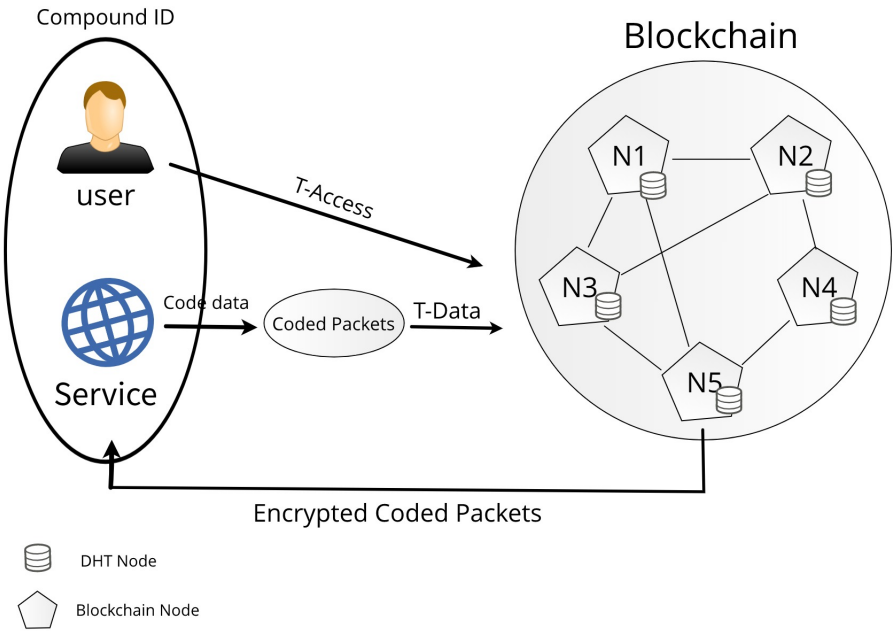


Figure 12. The proposed system model

coded packets from Kademlia. The process ends where k linearly independent coded packets are retrieved form Kademlia. The service then starts to decode the received coded packets by acquiring the coefficients from the service that has the coefficients.

- In Kademlia, a node that wishes to broadcast its data, Which is a common tool on most Kademlia systems to ensure reliability, asks the Blockchain for a permission to re-code. In this case, we have to outcomes:

    1. The Blockchain grants permission, i.e., the node is trustable. In this case, the node retrieves r coded packets from other nodes, where r is a system-wide variable. The node chooses the closest r keys in order to increase the speed of the function. The node then generates new coded packets using the recoding feature of RLNC [HPFM11] and sends each coded packet to a neighboring node. In this case, the coefficients and key value associated for each recoded packet are sent to the respective system and user in order to ensure access to the newly generated coded packets.
    2. The Blockchain does not grant permission, i.e., the node is not trustable. In this case, the node simply broadcasts the coded packet that it has stored to neighboring nodes. This procedure is identical to the traditional DHT

relocating procedure.

3.3.2 Protocol Procedures:

In this subsection, we are going to define the encoding and decoding procedure. The encoding procedure is performed in the service before sending data to Kademlia. In this protocol, the service acts as the encoder in the traditional encoder and decoder setup of network coding. First, the encoder divides the data into k chucks, i.e., $\{v_1, v_2, ..., v_k\}$. In order to make each coded packet, for example c_i, the encoder generates k random numbers $\{\alpha_{i,1}, \alpha_{i,2}, ..., \alpha_{i,k}\}$ from $GF(q)$. Then generates each coded packet c_i by calculating $c_i = \sum_{j=1}^{k} \alpha_{i,j} \times v_j$. The coefficients that are generated in this step are stored in matrix M. The matrix M is stored in the service in order to be used for decoding the data in the future. The output of this protocol is n coded packets $\{c_1, c_2, ...cc_n\}$. After this protocol is carried out, the service calls algorithm 4 for each coded packet to store them in Kademlia. It should be noted that coded packets generated are linearly independent as the service checks and ensures the linear Independence of these packets before sending them to Kademlia.

Algorithm 5: Encoding

Input: Value v, int q, int N, int K
Result: Matrix M and c_1, c_2, ..., c_n

```
Create Matrix M_{n×k}
Divide v into K chunks
foreach i=1 to n do
    c_i = 0
    foreach j=1 to k do
        Generate Random Number α_j ∈ GF(q)
        Store α_j in M_{i,j}
        c_i += α_j v_j
        end for
    end for
```

In order to retrieve the data, the service needs to know the keys associated with the coded packets generated for the data in the encoding procedure and given access to Kademlia by the user. Upon granting access and receiving the respective keys, the service starts to retrieve all of the n coded packets from Kademlia. In this setup, the service acts as the decoder in a traditional network coding system. Since only k coded packets are needed to perform the decoding procedure and retrieve the original, the process of

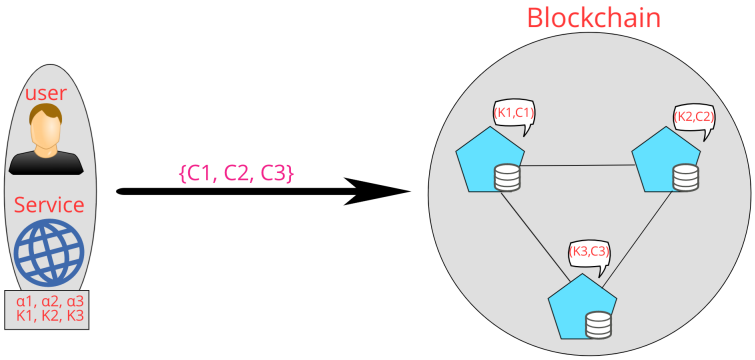


Figure 13. An illustration of the encoding procedure

retrieving the coded packets halts when k linearly independent coded packets are received by the service. This action results in lower traffic in the system and reduction in the overhead of the unnecessary transmissions in the channels compared to the case if all the coded packets were needed. After receiving k coded packets, the service performs the Gauss-Jordan elimination to decode the data and retrieve the original values $\{v_1, ..., v_k\}$. By knowing these values, the service can generate the original data v.

Algorithm 6: Decoding

Input: Matrix $M_{n\times k}$ and $Key_1, Key_2, ..., Key_n$
Result: Original Data v

1 **while** *true* **do**
2 Execute $HANDLEDATATX(P^k_{Sig}, (Key_i, x_p, r))$ in Parallel
3 **if** k *coded packets* $c'_1, c'_2, ..., c'_k$ *are received* **then**
4 Break;

5 Generate Matrix $D = \begin{pmatrix} \mathrm{M}_{\pi_1} & \mathrm{c}'_1 \\ \mathrm{M}_{\pi_2} & \mathrm{c}'_2 \\ \vdots & \vdots \\ \mathrm{M}_{\pi_k} & \mathrm{c}'_k \end{pmatrix}$

6 Use Gauss-Jordan elimination on D to generate $\{v_1, v_2, ..., v_k\}$.
7 Output $v = v_1||v_2||...||v_k$

3.4 Summary

In summary, we proposed a method to implement RLNC on top of a Blockchain-Kademlia system introduced in [ZN+15]. We are able to integrate RLNC in an extremely

efficient way without decreasing the efficiency of Kademlia or the functionality of the Blockchain. In fact, as we will address in the next section, our protocol enhances the performance of the system in various ways other than improved security.

4 Mathematical Analysis

As we mentioned earlier, due to the nature of RLNC, it improves the security of the storage of data in DHT. However, alongside improved security, there are more characteristics that are changed by implementation of RLNC. In this chapter we analyze how RLNC changes the expected retrieval time, resiliency and required storage of the DHT.

4.1 Retrieval Time

The first improvement of our work is the declination of data retrieval speed in the system, to prove this claim, we use the following notions to calculate the expected data retrieval speed in the original and the proposed system (for a random Value V):

- T_{Max} is the worst case scenario (upper bound) for the time it takes to retrieve any data in the DHT system. This upper bound is determined by experience.
- In traditional kademlia, data is requested from one of the nodes in DHT, therefore, the retrieval time t_o is a random variable where $T_o \in [0, T_{max}]$
- (T_1, T_2, ... ,T_n) are the retrieval times of the coded packets (c_1, c_2, ... ,c_n) respectively. Without loss of generality we can assume that $T_1 \leq T_2 ... \leq T_n$ and

$$0 \leq T_i \leq T_{Max} \quad \forall i \in \{1, ..., n\}$$

As mentioned, in the traditional DHT system we have:

$$0 \leq T_O \leq T_{Max} \tag{1}$$

Generally, if we do not know the distribution of T_o, which can be determined by experiment, we assume that T_o is a uniformly random variable, the expected value for the retrieval time is:

$$E(T_O) = \frac{T_{Max}}{2} \tag{2}$$

In our proposed system, are requests are sent in parallel and the transmission stops once we have received k coded packets, therefore, it is easy to see that the retrieval time in this system is equal to T_k. Using order statistic, the expected value for T_k is:

$$E(T_k) = T_{Max} \cdot \frac{k}{n+1} \tag{3}$$

The proof of this equation is in Appendix A [DN04]. From Eq. 2 and 3, we can conclude that:

$$E(T_k) \leq E(T_O) \quad \Leftrightarrow \quad \frac{k}{n+1} \leq \frac{1}{2} \tag{4}$$

Thus, with choosing $n \geq 2k-1$, the expected retrieval time in our proposed system in lower than traditional Kademlia.

4.2 Retrieval Time with channel error

The data might get lost during the retrieval due to channel error, if the we define the probability of loss due to channel error as ϵ, we can rewrite the eq. 3 as follows:

$$E(T_{k(1+\epsilon)}) = T_{Max} \times (\frac{k(1+\epsilon)}{n}) \tag{5}$$

This equation simply follows by the fact that if there is a loss probability of ϵ in the network, we need $k(1+\epsilon$ transmissions in order to receive k packets.

4.3 A tighter bound on improved expected retrieval time

The previously found bound $n \geq 2k-1$ is valid but simulation results show that the expected retrieval time is improved when using network coding even when $n < 2k-1$. Hence, in this section, we break down the expected retrieval time to find a better theory. The following notations are added:

- H is the Harmonic function [ABW13].
- m is the size of the P2P network (number of nodes).
- $\mathcal{K}$ is the size of K-buckets in Kademlia.

The expected retrieval time consists of two parts:

1. The time required to find the node containing the data, i.e., lookup time. We denote this time by T_{lookup}. This time is independent to the size of the stored values so it is identical in both traditional and network coded DHT.
2. The time required to download the data. This time depends on the size of the downloaded so it is different in the two cases. We denote this time as $T_{download}$ and $T_{codeddownload}$ for traditional and network coded DHT respectively.

To summarize, we write:

$$\begin{aligned} E(T_o) &= E(T_{lookup}) + E(T_{download}) \\ E(T_c) &= E(T_{lookup}) + E(T_{CodedDownload}) \end{aligned} \tag{6}$$

The authors of [CD13] have proposed an upper bound for the lookup time in kademlia as follows:

$$\max T_{lookup} = \frac{\log m}{H_{\mathcal{K}}} \tag{7}$$

The time of download has a linear relation with the size of packet [Li15]. The size of downloaded value in traditional and coded cases is $s_k + s_v$ and $s_k + \frac{s_v}{k}$ respectively. Hence, if we consider the size of lookup packet to be s_{req}, we have:

$$\begin{cases} \max T_{download} = \frac{(s_k+s_v)\log m}{s_{req}H_{\mathcal{K}}} \\ \\ \max T_{CodedDownload} = \frac{(s_k+\frac{s_v}{k})\log m}{s_{req}H_{\mathcal{K}}} \end{cases}$$

As the delay of each channel is assigned by a uniformly random distribution, the distribution of download and lookup times is a uniformly random variable in [0,T_{max}]. Therefore, using the same analysis as chapter 3.6.1 for expected time of each variable we have:

$$\begin{aligned} E(T_o) &= \frac{\log m}{2 \cdot H_{\mathcal{K}}} + \frac{(s_k + s_v)\log m}{2 \cdot s_{req}H_{\mathcal{K}}} \\ \\ E(T_c) &= \frac{k \log m}{(n+1)H_{\mathcal{K}}} + \frac{k(s_k + \frac{s_v}{k})\log m}{(n+1)s_{req}H_{\mathcal{K}}} \end{aligned} \tag{8}$$

Using these equations, we can find the condition for which we have an improvement by using network coding in the system. By setting $E(T_c) \leq E(T_o)$ in equation 8, we

have:

$$\frac{\log m}{2 \cdot H_{\mathcal{K}}} + \frac{(s_k + s_v)\log m}{2 \cdot s_{req} H_{\mathcal{K}}} \geq \frac{k \log m}{(n+1)H_{\mathcal{K}}} + \frac{(k \cdot s_k + s_v)\log m}{(n+1)s_{req}H_{\mathcal{K}}}$$

$$\rightarrow \frac{1}{2} + \frac{(s_k + s_v)}{2 \cdot s_{req}} \geq \frac{k}{n+1} + \frac{(k \cdot s_k + s_v)}{(n+1)s_{req}} \tag{9}$$

$$\rightarrow \frac{s_{req} + s_k + s_v}{2} \geq \frac{k \cdot s_{req} + k \cdot s_k + s_v}{n+1}$$

$$\rightarrow n + 1 \geq 2(\frac{k \cdot s_{req} + k \cdot s_k + s_v}{s_{req} + s_k + s_v})$$

If we set $n = k$ in the above equation we have:

$$k + 1 \geq 2(\frac{k \cdot s_{req} + k \cdot s_k + s_v}{s_{req} + s_k + s_v})$$

$$\rightarrow (k-1)s_{req} + (k-1)s_k \leq (k-1)s_v \tag{10}$$

$$\rightarrow s_v \geq s_k + s_{req}$$

This result shows that $E(T_c) \leq E(t_o)$ for all values of $n \geq k$. Therefore, when using network coding, we will always have gain in expected retrieval time.

Note that the analysis in this section is valid for large number of m as Eq. 7 is valid when $m \rightarrow \infty$ [CD13]. However, this analysis gives an insight on how network coding improves the performance of the p2p system in terms of average retrieval time.

4.4 Resiliency

The second improvement of our work is the improvement of resiliency, meaning that the nodes in the DHT are less probable to lose a data in case a node suddenly leaves the network, without loss of the generality, we use the following notions:

- α is the number of nodes that join the DHT network.
- β is the number of nodes that leave the DHT network.

- t which is a time interval during which α and β nodes are entered and left the network.
- h is the time period in which a node replicates its stored data with the neighbours.

The probability for a data loss during the interval t for the both scenarios, our system and the original system is determined as follows:

For the original system the probability of a data loss at the time interval t is:

$$P_l(t, \alpha, \beta, h) = p_l \tag{11}$$

For our proposed system, since we are retrieving k coded packets, loss would mean that we lose at least $n - k + 1$ packets, and the occurrence of the loss are independent from each other, we can use multiplication rule to achieve:

$$P_l(t, \alpha, \beta, h) = p_l^{n-k+1} \tag{12}$$

Since the $p_l \leq 1$, it is evident that:

$$p_l^{n-k} \leq p_l \tag{13}$$

Thus we can conclude that probability of a data loss is less in our proposed system.

4.5 Storage

To calculate the storage required for the data (Key, Value) in our proposed system, we use the following notions:

- s_k the space required to keep the key.
- s_v the space required to keep the data.
- s_c the space required to keep the coded packet.
- s_{kc} the space required to keep the key of coded packet.

since the data is chunked into k packets and then coded to n packets, and the size of keys remain the same (160-bits) the overall required storage would be:

$$\begin{aligned} storage &= n \cdot (s_{kc} + s_c) \\ storage &= n \cdot (s_k + \frac{s_v}{k}) \end{aligned} \tag{14}$$

Using this mathematical tools, we know that implementation of RLNC improves the performance of the system in resiliency and expected retrieval time in addition to the security it provides. In the next chapter, we prove the validity of our analysis with simulation results.

4.6 Summary

In this section we analyzed how the implementation of RLNC affects the average retrieval time, resiliency and required storage. Our analysis shows that RLNC improves the expected retrieval time and resilience in any case, with the expense of increased required storage. In the next section, we put our analysis to test by using a simulation technique.

5 Simulations and results

In this chapter, we describe our simulation technique as well as Peersim, the simulation engine that we use to validate our analysis. Then we provide the results and debate on the meaning of our simulation results and how these results approve our analysis on the proposed method.

5.1 Peersim

The key characteristics of of peer-to-peer (P2P) networks are scalability being dynamic thus the analysis and evaluation of a P2P protocol in a realistic environments is very expensive and difficult. As simulation like in other areas is crucial in P2P related research chosing the right simulater is very important. For this research we have chosen to use Peersim which is an extremely scalable simulation environment that supports the dynamic environment of P2P networks.

Peersim [MJ09] is a P2P simulation environment with extremely large scalablilty (Millions of Nodes) in which the nodes join and leave continuously as in the real world P2P networks. The simluator is written in Java and it is structured based on java components to makes it easy to prototype a protocol by combining different Java pluggable building blocks. There are two simulation models proposed for peersim as follows:

- **cycle-based model** is a simplified model in which there is the lack of transport layer simulation and concurrency, in other words, nodes communicate with each other directly with each nodes having the control periodically, in a sequential order. This model makes it possible to achieve extreme scalability and performance, at the cost of loss of some realism while several simple protocols can tolerate this loss without problems, care should be taken when using this model in the experiments.
- **event-based model** is a more complicated and realistic implementation of P2P simulator. This model is very simulator to the cycle-based model except that since there are no cycles in this model, the control objects, which are defined later, should explicitly be defined when to run.

5.1.1 PeerSim simulation life-cycle

PeerSim follows the modular programming logic based on objects which are the building blocks of the simulator. Every block is easily replaceable by another component which implements the same interface or functionality. The general precedure to simulate a protocol in Peersim is as follows:

- choosing the size of network (number of nodes).
- choosing and initializing the desired protocols on the nodes.
- choosing the Control objects that monitor or modifies the desired properties during the simulation (e.g., size of the network or the internal state of the protocols)
- Running the simulation by invoking the Simulator class with a configuration file, which contains the information mentioned above.

The life-cycle of a cycle-based simulation is as follows:

1. The first step is to read the configuration file, given as a command-line parameter. The configuration contains all the information regarding the simulation including simulation parameters and the objects involved in the experiment.
2. In the second step, the simulator sets up the network (constructing the number of nodes defined in the network, and the protocols in them.) It should be noted that each node has the same protocol; which are the instances of a protocol, with one instance initialized in each node. These instances are created by cloning meaning only one instance is constructed using the object's constructor and all the nodes in the network are further cloned from this prototype. Due to the mentioned fact, it is very important to pay attention to the implementation of the cloning method for the protocols.
3. In the third step, initialization (setting the initial state of each protocol) is performed. This phase is carried out by the initializer objects that are scheduled to run at the beginning of each experiment. In the configuration file (explained later in this chapter), the initialization components are recognizable by the "init" prefix at the beginning of the line. It should be noted that initializer objects are simply controls, but configured to run only in the initialization phase.
4. In the forth step, the cycle driven engine calls all the components (protocols and controls) once in each cycle, until a given number of cycles, or in case a component decides to end the simulation.
5. In the fifth step, if a Control object has to collect data, the data is formatted and sent to standard output (which can be easily transferred to a file for further analysis.)

Node	The P2P network is made up of nodes. A node is a container of protocols. The node interface provides access to the protocols it holds, and also to the fixed ID of the node.
CDProtocol	It is a specific protocol, that is designed for the cycle-driven model. Such a protocol simply defines a certain operation to be performed at each cycle.
Linkable	Typically used by protocols class, this interface provides a service to other protocols to access a set of neighbor nodes. The instances of the same linkable protocol class over the nodes make up an overlay in the network.
Control	Classes that implement this interface can be scheduled for execution at certain points during the simulation. These classes typically observe or modify the simulation when it is running.

Table 1. Main Components of Peersim

Each object in PeerSim (e.g., controls and protocols) are assigned a Scheduler object which dictates when they are executed exactly. By default, all objects are executed at each cycle. However, it is possible to configure a protocol or control object to run only in certain cycles (e.g., every other cycle), and it is also possible to control the order of the components within each cycle. The latter case is illustrated in Figure 14.

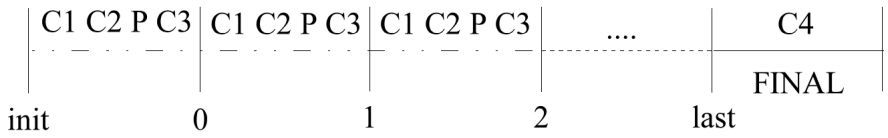


Figure 14. “C” donates a control component, and “P” donates a protocol. The numbers under each vertical line indicate the cycle. In this figure for example, it is possible to execute a final control object to retrieve a final snapshot of the system. [Jes05]

5.1.2 PeerSim Components

As stated earlier, all the object that are created during simulation are instances of PeerSim’s classes that implement one or more interfaces. Table 1 donates some of the most important interfaces.

5.1.3 The configuration file

The configuration file is basically a plain ASCII text file, the lines starting with "#" character are regarded as comments and are ignored. The configuration file is specified in the command line as follows:

```
java peersim.Simulator config-file.txt
```

To have better understating of the configuration files in PeerSim, we would explain our configuration file for our simulation in Appendix B.

5.2 Simulation results

As our system only differs on the aspect of network coding before inserting data in Kademlia, in this section we test the effect of this aspect on the expected retrieval and storage time on Kademlia. we only implemented a Kademlia system and gathered the results on expected retrieval and storage time on the Kademlia. The Blockchain on top of Kademlia has negligible effect on the expected retrieval and storage time so the results of this section can be expanded to our system model. This is because of the fact that the application of the blockchain does not change in our system with respect to the system introduced in [ZN$^+$15]. The protocols and functionalities that involve the Blockchain stay exactly the same. Therefore, if we compare the difference between a Kademlia that uses RLNC to store the data and a Kademlia that does not use RLNC, the results can be extended to our system and the effects that our proposed model has on the system introduced in [ZN$^+$15].

In order to validate our analysis and compare it with traditional Kademlia system in which network coding is not implemented, we carried out simulations using Peersim. In these simulations, we set number of nodes as $m = \{500, 1000\}$. Our topology is a wired network, in which each node is connected to 5 other nodes. In order to better illustrate the topology, figure 15 shows a setup of 10 nodes where each node is connected to 3 other nodes.

In our simulations, we first set up the mentioned topology. Then we set up the following system characteristics:

- Minimum and Maximum Delay in the channels. In this work, we considered that each channel in the network has a delay from a uniform distribution in the range of [50, 100]ms. Which means that the minimum and maximum delay are set as 50ms and 100ms for all our experiments respectively.

- Drop rate. We define the drop rate in as the probability that a packet is lost in a channel. The drop rate in our experiments is set as $drop = \{0.01, 0.05\}$, This drop rate is set in the configuration file as can be seen in Appendix B.

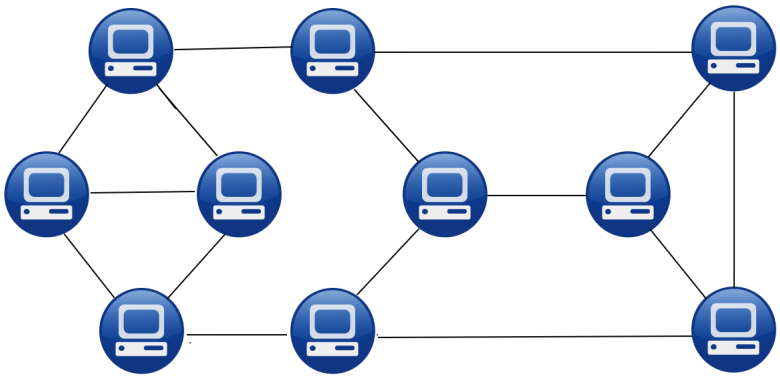


Figure 15. An example of a topology with 10 nodes where each node is connected to 3 other nodes.

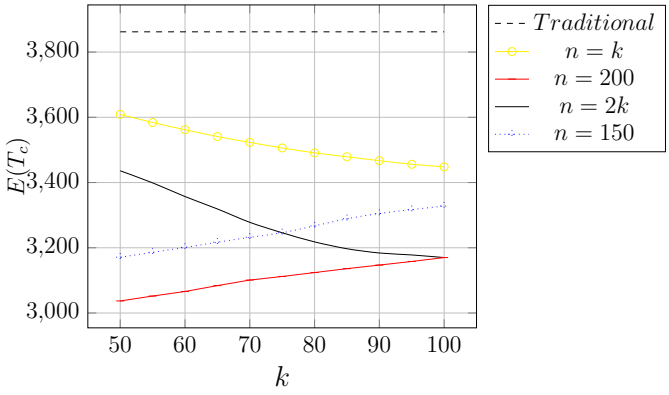
- k and n. In other words, number of chunks and number of coded packets. These variables are set inside the main program. These two variables are the main variables in our experiments as we want to see the effect of these two on the expected retrieval and storage time.

In order to consider the dynamic nature of P2P systems, we have also introduced a turbulence step in our network as can be seen in the configuration file in appendix B. The turbulence step and how it manages the dynamic nature of P2P systems is discussed in appendix B. We make sure that this turbulence step is defined in a way to keep the average number of nodes around a defined value of m. This is ensured by having the probability that a node leaves the network in a turbulence step equal to the probability that a node joins in a turbulence step.

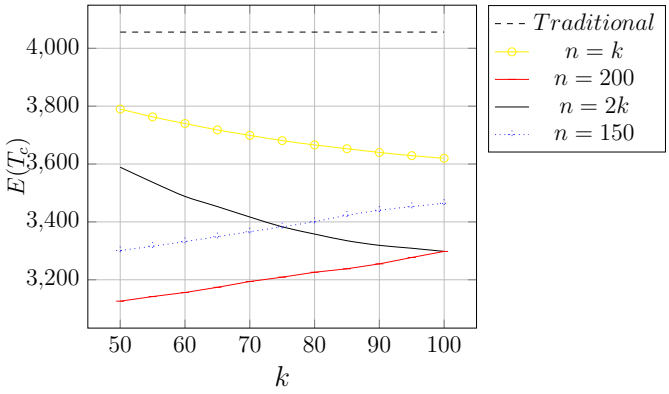
We tested our setup for various set of network characteristics. In order to make sure that the results are statistically bound, we carried out 1000 independent experiments for each set of variables and reported the average result. In the figures we only include the average result without the confidence interval, because the results were generally close and the confidence interval is negligible with respect to the results.

5.2.1 Effect on Retrieval Time

Figures. 18 and 16 show the expected retrieval time for various values of k and n in our system as well as in traditional Kademlia for different values of m and ϵ which denote the average total number of nodes in the network and the drop rate of each channel. As can be seen in this figures, we have significant gain by using network coding with any value of k and n. Also these results confirms the expected outcomes as our mathematical analysis. These results can be categorized as following:



(a) $\epsilon = 0.01$



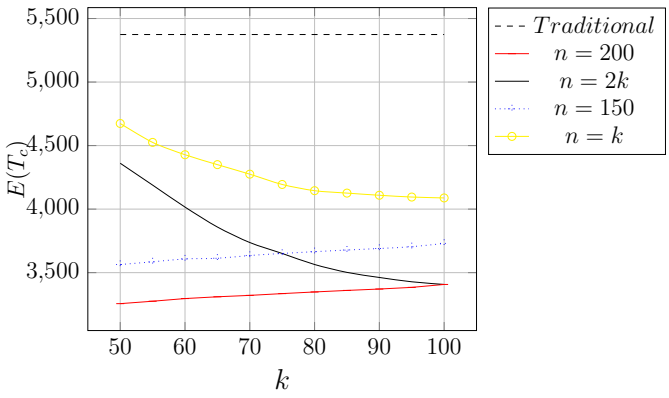
(b) $\epsilon = 0.05$

Figure 16. Average Retrieval Time for $m = 500$ and $\epsilon = \{0.01, 0.05\}$. The size of each data is set at $10KB$.

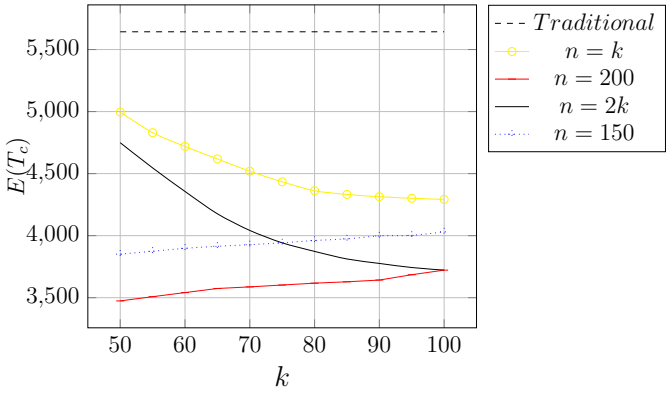
1. By increasing the number of nodes in the P2P network, the expected retrieval time is increased as the expected number of hops required for each retrieval is increased.

2. Increasing the drop rate on channels increase the expected retrieval time which is intuitive as the higher probability of losing a packet requires additional transmissions in the network to gather the required k coded packets. However, the effect of drop rate is higher when network coding is not used. Our results show that when the drop rate is increased from 0.01 to 0.05, the expected retrieval time increases by 6% in traditional Kademlia. While the expected retrieval time in network coding enhanced Kademlia is only increased by 3% on average. This happens because there are multiple coded packets in the network, so if any of them is dropped in the channel, we can substitute it with another packet.

3. For a constant k, by increasing the number of n, the expected retrieval time is



(a) $\epsilon = 0.01$



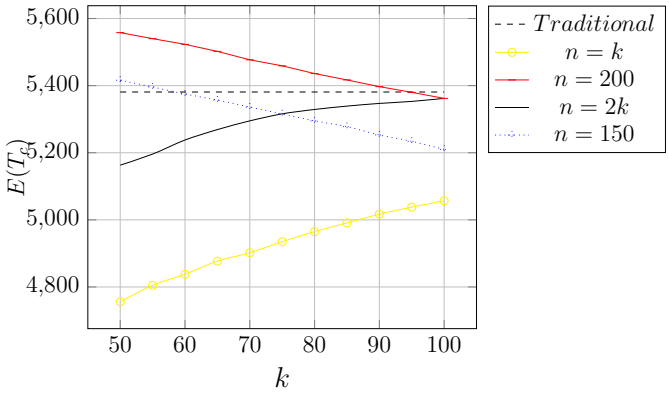
(b) $\epsilon = 0.05$

Figure 17. Average Retrieval Time for $m = 1000$ and $\epsilon = \{0.01, 0.05\}$. The size of each data is set at $10KB$.

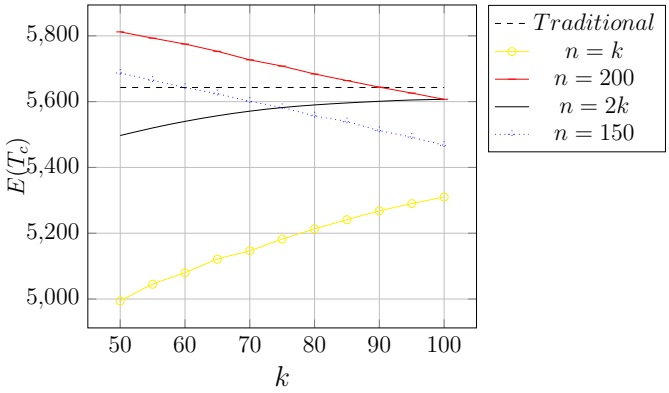
decreased. By fixing the value of k, the number of coded packets that are needed to retrieve and the size of coded packets are constant. Higher number of n means that there are higher number of coded packets in the network so the probability that the coded packets are closer to the receiver node is increased, which means faster retrieval time.

4. For a constant n, increasing k increases the expected retrieval time. By fixing n, the total number of coded packets in the system stay the same, however increasing k means that more packets are needed to be retrieved, therefore, the expected lookup and download time are both increased. This can be clearly seen in Eq. 8.

5. By fixing the relation between k and n, i.e., $\frac{k}{n} = const.$, we observe the expected retrieval time decreases by increasing the value of k and n. This can be seen trivially in Eq. 8. The intuitive reason for this behaviour is that increasing both k and n



(a) $\epsilon = 0.01$



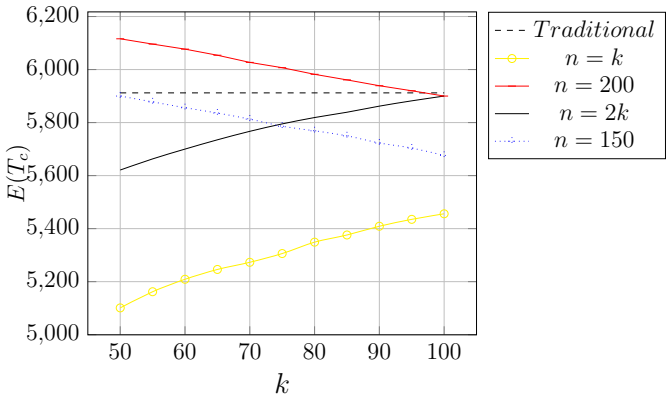
(b) $\epsilon = 0.05$

Figure 18. Average Storage Time for $m = 500$ and $\epsilon = \{0.01, 0.05\}$. The size of each data is set at $10KB$.

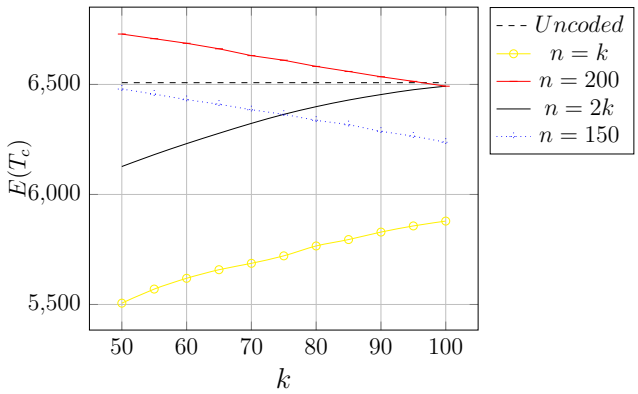
decrease the size of each coded packet and also increases the total number of coded packets in the network, which improves the performance, this improvement is stronger than the drawback of requiring more coded packets to decode the original data, which describes the behaviour that we can observe from our simulation. Theoretically, if we set $\frac{k}{n} = c$ in Eq. 8, we get:

$$E(T_c) \simeq \frac{c \log m}{H_{\mathcal{K}}} + \frac{c(s_k + \frac{s_v}{k}) \log m}{s_{req} H_{\mathcal{K}}}$$

Therefore, increasing k decreases the expected retrieval time by a factor of $\frac{1}{k}$.



(a) $\epsilon = 0.01$



(b) $\epsilon = 0.05$

Figure 19. Average Storage Time for $m = 1000$ and $\epsilon = \{0.01, 0.05\}$. The size of each data is set at $10KB$.

5.2.2 Effect on Storage Time

The plots in Fig. 19 show the results on the expected storage time for various values of k and n in our system as well as in traditional Kademlia for different values of m and ϵ which denote the average total number of nodes in the network and the drop rate of each channel. As can be seen in this figures, Even when network coding is used, the expected storage time is not much increased with respect to traditional Kademlia. In fact, when $n = k$, the expected storage time is significantly lower the traditional Kademlia. comparing the results for various numbers of k, n and m we get the following insights.

1. By increasing the number of nodes in the P2P network, the expected storage time is increased as the expected number of hops required for each store procedure is increased.

2. Increasing drop rate increases the expected storage time significantly. In retrieval

of data, where in case of loss of any packets, we can get another coded packet from another source until we get k coded packets. However, when storing the coded packets, in case of data loss, we need to send the coded packet again until it is stored in the destination.

3. For a constant k, by increasing the number of n, the expected Storage time is increased. By fixing the value of k, the size of coded packets are constant. Higher number of n means that there are higher number of coded packets that need to be stored in the network so the expected storage time is increased.

4. For a constant n, increasing k decreases the expected storage time. By fixing n, the total number of coded packets that need to be stored in the system stay the same, however increasing k means that size of each coded packet is decreased, therefore, the expected storage time is decreased proportional to the size of each coded packet. This can be clearly seen in Eq. 8.

5. By fixing the relation between k and n, i.e., $\frac{k}{n} = const.$, the expected storage time increases by respectively increasing the value of k and n. The reason for this behaviour is that increasing both k and n decrease the size of each coded packet and also increases the total number of coded packets that need to be stored in the network. This behaviour suggests that the higher number of packets that need to be stored has more effect on the expected time than the decreased size of packets, so we have an increase in the expected storage time.

These results show that there is a trade-off between expected storage time and expected retrieval time. This trade-off means that if we increase the total storage time, either by increasing the value of k or n, we decrease the total retrieval time.

5.3 Summary

In this section, we validated our analysis and the performance of our system using a simulation with Peersim environment. Our results show that RLNC significantly increases the expected retrieval time while the expected storage time remains close to the traditional Kademlia in most cases. Also there is a trade-off between storage and retrieval time. This trade-off means that if we increase the storage time by increasing the value of n or decreasing the value of k, the performance of the system in terms of expected retrieval time is decreased.

6 Conclusion and future Work

In this work, we used the system proposed in [ZN$^+$15] and added a network coding feature by implementing RLNC. The original setup was proposed to preserve privacy of

data stored in a P2P storage network. Our analysis and simulation results show that by implementing RLNC in this network, we achieve following improvements in our system:

- The security of the stored data is increased as RLNC provides inherent security on the data., i.e., by storing coded packets, the security is increased as proven by multiple previous researches such as [HPFM11].
- Data recovery time, i.e., the time required for Kademlia to return the data is faster in the proposed solution compared to traditional Kademlia.
- the system is more fault-tolerant to the removal of nodes from the system, which means that The Kademlia requires less number of data relocations, further reducing the traffic on channels between nodes.

Our results further shows that by implementing RLNC there is a trade-off between storage time and retrieval time. Which means by increasing the number of coded packets or the generation size, i.e., the number of chunks that the data is divided, the storage time is increased and the retrieval time is decreased. Therefore, we can manage the system characteristic depending on our expectations of the distributed storage.

As for the future works, we can focus on expanding the theoretical framework to find the expected behaviour of the system on the expected storage time and fault-tolerance of the system in case of nodes leaving the network. This expansion will help us better understand such a system to help us implement the system in a more efficient way and find more trade-offs between system characteristics.

Another feature that we can include in our analysis is the channel capacity and node capacity which can create bottlenecks and delays in the network. This issue is a well known and common topic in P2P networks such as Kademlia. Therefore we can analyze the effect of network coding on systems with this characteristics.

Another interesting feature is the possibility of re-coding the coded packets in the P2P networks. Re-coding is one of the advantages of RLNC which allows generation of new coded packets by using the code packets already in the network without decoding. This feature can be implemented into the P2P networks, where the a set of approved nodes in the network can use this feature to generate more coded packets. This will increase the performance of system by reducing the expected retrieval time and increasing fault-tolerance of the network while keeping expected storage time intact.

References

[ABW13] Sheldon Axler, Paul Bourdon, and Ramey Wade. *Harmonic function theory*, volume 137. Springer Science & Business Media, 2013.

[Alb16] Jan Philipp Albrecht. How the gdpr will change the world. *Eur. Data Prot. L. Rev.*, 2:287, 2016.

[AT02] Stephanos Androutsellis-Theotokis. A survey of peer-to-peer file sharing technologies. 2002.

[BKV+18] SB Balaji, M Nikhil Krishnan, Myna Vajha, Vinayak Ramkumar, Birenjith Sasidharan, and P Vijay Kumar. Erasure coding for distributed storage: An overview. *Science China Information Sciences*, 61(10):100301, 2018.

[BN+05] Kapil Bhattad, Krishna R Narayanan, et al. Weakly secure network coding. *NetCod, Apr*, 104, 2005.

[CD13] Xing Shi Cai and Luc Devroye. A probabilistic analysis of kademlia networks. In *International Symposium on Algorithms and Computation*, pages 711–721. Springer, 2013.

[CL11] Terence Craig and Mary E Ludloff. *Privacy and big data: the players, regulators, and stakeholders*. " O'Reilly Media, Inc.", 2011.

[CSWH01] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Designing privacy enhancing technologies*, pages 46–66. Springer, 2001.

[CT06] Christian Cachin and Stefano Tessaro. Optimal resilience for erasure-coded byzantine distributed storage. In *International Conference on Dependable Systems and Networks (DSN'06)*, pages 115–124. IEEE, 2006.

[DGW+10] Alexandros G Dimakis, P Brighten Godfrey, Yunnan Wu, Martin J Wainwright, and Kannan Ramchandran. Network coding for distributed storage systems. *IEEE transactions on information theory*, 56(9):4539–4551, 2010.

[DN04] Herbert Aron David and Haikady Navada Nagaraja. Order statistics. *Encyclopedia of Statistical Sciences*, 2004.

[FLP85] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[GK03] Nathaniel S Good and Aaron Krekelberg. Usability and privacy: a study of kazaa p2p file-sharing. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 137–144, 2003.

[Goe14] Vindu Goel. Facebook tinkers with users' emotions in news feed experiment, stirring outcry. *The New York Times*, 29, 2014.

[HLZ10] Qinlu He, Zhanhuai Li, and Xiao Zhang. Study on cloud storage system based on distributed storage systems. In *2010 International Conference on Computational and Information Sciences*, pages 1332–1335. IEEE, 2010.

[HMK+06a] Tracey Ho, Muriel Médard, Ralf Koetter, David R Karger, Michelle Effros, Jun Shi, and Ben Leong. A Random Linear Network Coding Approach to Multicast. *IEEE Trans. on Information Theory*, 52(10):4413–4430, 2006.

[HMK+06b] Tracey Ho, Muriel Médard, Ralf Koetter, David R Karger, Michelle Effros, Jun Shi, and Ben Leong. A random linear network coding approach to multicast. *IEEE Transactions on Information Theory*, 52(10):4413–4430, 2006.

[HPFM11] Janus Heide, Morten V Pedersen, Frank HP Fitzek, and Muriel Médard. On code parameters and coding vector representation for practical rlnc. In *2011 IEEE international conference on communications (ICC)*, pages 1–5. IEEE, 2011.

[HSX+12] Cheng Huang, Huseyin Simitci, Yikang Xu, Aaron Ogus, Brad Calder, Parikshit Gopalan, Jin Li, and Sergey Yekhanin. Erasure coding in windows azure storage. In *Presented as part of the 2012 {USENIX} Annual Technical Conference ({USENIX} {ATC} 12)*, pages 15–26, 2012.

[Jes05] Gian Paolo Jesi. Build a new protocol for the peersim 1.0 simulator. Technical report, unibo, 2005.

[KA17] Justin P Koeln and Andrew G Alleyne. Stability of decentralized model predictive control of graph-based power flow systems via passivity. *Automatica*, 82:29–34, 2017.

[KGJØ17] Katina Kralevska, Danilo Gligoroski, Rune E Jensen, and Harald Øverby. Hashtag erasure codes: From theory to practice. *IEEE Transactions on Big Data*, 4(4):516–529, 2017.

[KKE03] In-suk Kim, Yong-hyeog Kang, and Young Ik Eom. An efficient contents discovery mechanism in pure p2p environments. In *International Conference on Grid and Cooperative Computing*, pages 420–427. Springer, 2003.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, 2014.

[KMM+02] Mandar Kelaskar, Vincent Matossian, Preeti Mehra, Dennis Paul, and Manish Parashar. A study of discovery mechanisms for peer-to-peer applications. In *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)*, pages 444–444. IEEE, 2002.

[Li15] Keqin Li. Analysis of file download time in peer-to-peer networks with stochastic and time-varying service capacities. *Future Generation Computer Systems*, 42:36–43, 2015.

[LLC99] MultiMedia LLC. MS Windows NT kernel description, 1999.

[LT11] Hsiao-Ying Lin and Wen-Guey Tzeng. A secure erasure code-based cloud storage system with secure data forwarding. *IEEE transactions on parallel and distributed systems*, 23(6):995–1003, 2011.

[MJ09] Alberto Montresor and Márk Jelasity. Peersim: A scalable p2p simulator. In *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*, pages 99–100. IEEE, 2009.

[MJ18] Tomas Mikula and Rune Hylsberg Jacobsen. Identity and access management with blockchain in electronic healthcare records. In *2018 21st Euromicro Conference on Digital System Design (DSD)*, pages 699–706. IEEE, 2018.

[MLFR01] Ethan Miller, Darrell Long, William Freeman, and Benjamin Reed. Strong security for distributed file systems. In *Conference Proceedings of the 2001 IEEE International Performance, Computing, and Communications Conference (Cat. No. 01CH37210)*, pages 34–40. IEEE, 2001.

[MM02] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[Nak19] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.

[NS06] Beomseok Nam and Alan Sussman. Dist: fully decentralized indexing for querying distributed multidimensional datasets. In *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*, pages 10–pp. IEEE, 2006.

[PBV05] B Pourebrahimi, K Bertels, and S Vassiliadis. A survey of peer-to-peer networks. In *Proceedings of the 16th Annual Workshop on Circuits, Systems and Signal Processing, ProRisc*, volume 2005. Citeseer, 2005.

[Pes13] Yaniv Pessach. *Distributed storage: concepts, algorithms, and implementations*. 2013.

[PH] Amir H. Payberah and Seif Haridi. Kademlia: A peertopeer information system based on the xor metric.

[RD01] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 329–350. Springer, 2001.

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, 2001.

[RFI02] Matei Ripeanu, Ian Foster, and Adriana Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *arXiv preprint cs/0209028*, 2002.

[Rip01] Matei Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *Proceedings first international conference on peer-to-peer computing*, pages 99–100. IEEE, 2001.

[RNW+15] KV Rashmi, Preetum Nakkiran, Jingyan Wang, Nihar B Shah, and Kannan Ramchandran. Having your cake and eating it too: Jointly optimal erasure codes for i/o, storage, and network-bandwidth. In *13th {USENIX} Conference on File and Storage Technologies ({FAST} 15)*, pages 81–94, 2015.

[SMK+01a] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.

[SMK+01b] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, Hari Balakrishnan, and A Chord. A scalable peer-to-peer lookup service for internet applications. *Lab. Comput. Sci., Massachusetts Inst. Technol., Tech. Rep. TR-819*, 2001.

[Swa15] Melanie Swan. *Blockchain: Blueprint for a new economy*. " O'Reilly Media, Inc.", 2015.

[TR03] Dimitrios Tsoumakos and Nick Roussopoulos. A comparison of peer-to-peer search methods. In *WebDB*, pages 61–66, 2003.

[W+14] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[WB99] Stephen B Wicker and Vijay K Bhargava. *Reed-Solomon codes and their applications*. John Wiley & Sons, 1999.

[WDMS17] Kun Wang, Miao Du, Sabita Maharjan, and Yanfei Sun. Strategic honeypot game model for distributed denial of service attacks in the smart grid. *IEEE Transactions on Smart Grid*, 8(5):2474–2482, 2017.

[WG18] Karl Wüst and Arthur Gervais. Do you need a blockchain? In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 45–54. IEEE, 2018.

[YGM02] Beverly Yang and Hector Garcia-Molina. Improving search in peer-to-peer networks. In *Proceedings 22nd International Conference on Distributed Computing Systems*, pages 5–14. IEEE, 2002.

[YGM03] B Beverly Yang and Hector Garcia-Molina. Designing a super-peer network. In *Proceedings 19th international conference on data engineering (Cat. No. 03CH37405)*, pages 49–60. IEEE, 2003.

[ZN+15] Guy Zyskind, Oz Nathan, et al. Decentralizing privacy: Using blockchain to protect personal data. In *2015 IEEE Security and Privacy Workshops*, pages 180–184. IEEE, 2015.

Appendix

Appendix A: Proof of Eq. 3

In this setup, we have $\{T_1, ..., T_n\}$, which are uniformly random variables from $[0, T_{max}]$ and $T_1 \leq T_2 \leq ... \leq T_n$. We want to find the expected value of T_k. It is trivial that T_k is the $k-th$ smallest value in this sample. we can map these values into ${}_1, ..., U_n$ in unit interval $[0, 1]$ by setting $U_i = \frac{T_i}{T_{max}}$. The cumulative distribution function of T_k is calculated as:

$$\begin{aligned}\mathbb{P}\left(U_k \leqslant u\right) &= \sum_{i=k}^{n} \mathbb{P}\left(U_1 \leqslant u, \ldots, U_i \leqslant u, U_{i+1} > u, \ldots, U_n > u\right) \\ &= \mathbb{P}\left(\sum_{i=1}^{n} [U_i \leqslant u] \geqslant k\right)\end{aligned}$$

where $[U_i \leq u]$ is the Iverson bracket, which is defined as follows:

$$[U_i \leq u] = \begin{cases} 1 & if\ U_i \leq u; \\ 0 & otherwise \end{cases}$$

As all U_i are uniformly random, we have:

$$\mathbb{E}\left([U_i \leqslant u]\right) = \mathbb{P}\left(U_i \leqslant u\right) = u \tag{15}$$

The sum of n independent and identically distributed random variables equals in distribution to a binomial random variable with parameters n and u. Thus:

$$F(u) = \mathbb{P}\left(U_k \leqslant u\right) = \sum_{i=k}^{n} \binom{n}{i} u^i (1-u)^{n-i} \tag{16}$$

By using the above equations, we can calculate the expected value of U_k:

$$
\begin{aligned}
\mathbb{E}\left(U_k\right) &= \int_0^1 uF'(u)\mathrm{d}u = uF(u)|_{u=0}^{u=1} - \int_0^1 F(u)\mathrm{d}u \\
&= 1 - \sum_{i=k}^{n} \binom{n}{i} \int_0^1 u^i (1-u)^{n-i}\mathrm{d}u \\
&= 1 - \sum_{i=k}^{n} \binom{n}{i} B(i+1, n-i+1) \\
&= 1 - \sum_{i=k}^{n} \frac{n!}{i!(n-i)!} \cdot \frac{(i)!(n-i)!}{(n+1)!} \\
&= 1 - \sum_{i=k}^{n} \frac{1}{n+1} = \frac{k}{n+1}
\end{aligned}
$$

Now, we can use the transformation to calculate the expected value of T_k.

$$
\mathbb{E}\left(T_k\right) = T_{max} \cdot \mathbb{E}\left(U_k\right) = T_{max} \cdot \frac{k}{n+1} \tag{17}
$$

Appendix B: PeerSim Configuration Settings

```
# ::::: GLOBAL ::::::

# Network size
SIZE 500
```

This line denote size of the network (Number of Nodes). in this case, 500 nodes.

```
K 5
MINDELAY  50
MAXDELAY  100
SIM_TIME 500*60*60
```

K is the size of the K-Buckets, MINDELAY and MAXDELAY are the minimum and the maximum delay set for each channel and the SETTime is the total duration of the simulation, in this case 500 hours

```
TRAFFIC_STEP  (SIM_TIME)/SIZE
```

TRAFFIC_STEP denotes the step in which data is relocated in Kademlia, In this case every hour.

```
OBSERVER_STEP 100000
```

OBSERVER_STEP denotes the step in which the observer object is executed in order to gather the data related to the simulation, i.e., the average retrive time.

```
TURBULENCE_STEP  (SIM_TIME)/(20*SIZE)
```

TURBULENCE_STEP denotes the steps in which we have a turbulence (Nodes joining and leaving the network). This turbulence is defined using some probabilities which are defined later in this code.

```
STORE_STEP   (SIM_TIME)/SIZE*50
```

STORE_STEP denotes the steps in which data storage is performed in the network.

```
FINDVALUE_STEP (SIM_TIME)/SIZE*100
```

FINDVALUE_STEP denotes the steps in which a node requests a find value in the network.

```
simulation.experiments 1
simulation.endtime SIM_TIME*3
```

These variables are related to the PeerSim Engine and define how many times the simulation is repeated and the simulation end time.

```
protocol.0link peersim.core.IdleProtocol
protocol.1uniftr peersim.transport.UniformRandomTransport
protocol.1uniftr.mindelay MINDELAY
protocol.1uniftr.maxdelay MAXDELAY
protocol.2unreltr peersim.transport.UnreliableTransport
protocol.2unreltr.drop 0
protocol.2unreltr.transport 1uniftr
protocol.3kademlia peersim.KademliaProtocol
protocol.3kademlia.transport 2unreltr
#protocol.3kademlia.BITS 32
```

In this part, we define and set the variables for the protocols in our simulation. There are 4 protocols in the setup, IdleProtocol indicates the foundation of the system which stores the nodes. UniformRandomTransport is the environment in which the channels are defined and finally KademliaProtocol is the Kademlia hash table which is implemented over the nodes and channels defined by the other two protocols.

```
# ::::: INITIALIZERS :::::
init.0randlink peersim.dynamics.WireKOut
init.0randlink.k K
init.0randlink.protocol 0link
```

In this part, we define the topology of the system. We set our system as nodes connected by wired channels and each node is connected to k other nodes.

```
init.1uniqueNodeID peersim.CustomDistribution
init.1uniqueNodeID.protocol 3kademlia

init.2statebuilder peersim.StateBuilder
init.2statebuilder.protocol 3kademlia
init.2statebuilder.transport 2unreltr
```

Here we initialize the defined the protocols on our simulation.

```
# ::::: CONTROLS :::::

# traffic generator
control.0traffic peersim.TrafficGenerator
control.0traffic.protocol 3kademlia
control.0traffic.step TRAFFIC_STEP
```

In this part and the following parts which are under the "controls" section, we define the functions that will be executed during the simulation. In this specific part, we define the TrafficGenerator function which handles the relocation of data and set its variables.

```
# turbolence
control.2turbolenceAdd peersim.Turbulence
control.2turbolenceAdd.protocol 3kademlia
control.2turbolenceAdd.transport 2unreltr
control.2turbolenceAdd.step TURBULENCE_STEP
control.2turbolenceAdd.p_idle 0.5
control.2turbolenceAdd.p_rem 0.25
control.2turbolenceAdd.p_add 0.25
```

In this part , we define the turbulance in the network. The probabilities set in this part define the probability of "no change", "add a node" and "remove a node" respectively each time that the function turbulance is executed.

```
#store msg generator
control.4store peersim.StoreMessageGenerator
control.4store.protocol 3kademlia
control.4store.step STORE_STEP

#find value msg generator
control.5findvalue peersim.FindValueMessageGenerator
control.5findvalue.protocol 3kademlia
control.5findvalue.step FINDVALUE_STEP
```

In this part we set the store and retrieve messages and their time steps.

```
# ::::: OBSERVER :::::
control.3 peersim.KademliaObserver
control.3.protocol 3kademlia
control.3.step OBSERVER_STEP
```

In this part we set the KademliaObserver class to be executed. This class outputs the characteristics of the system such as the average store and retrieve time after a fixed time which is set in the beginning of the configuration file.

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Saeid Mousavifar**,
*(*author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   An Improvement for The Decentralized Privacy System Using Random Linear Network Coding,
   *(*title of thesis)

   supervised by Rune Hylsberg Jacobsen and Satish Srimara.
   *(*supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Saeid Mousavifar
15/05/2020