Decentralized communication

How to send messages to unresponsive clients in a chat network

Isak Dunér Lundberg
Abstract

A communication solution to send messages between clients in a decentralized network even though the receiver may be unreachable at the time of transmission is presented. The solution is built on a decentralized cloud infrastructure and includes cryptically key pairs to perform client authentication. The network is resistant to failures and have redundancy and self-healing capabilities built in.

To reach this goal we analyze some existing systems and networks which forms the basis for this research. Some of these applications include Bittorrent, Bitcoin and Storj.
Acknowledgements

I would like to thank my mentor Pedher Johansson for the feedback during the work on this paper. I would also like to thank a friend of mine, Martin Lundgren who have been very helpful in my process. I would also like to give a shout out to all other students who during the spring of 2015 did their thesis as they in one way or another been a part of the journey.
Contents

1 Introduction 1
  1.1 Background 1
  1.2 Problem description 2
  1.3 Scope and limitation 2
  1.4 Disposition 2

2 Theory 3
  2.1 Decentralized networks 3
  2.2 Applications and systems 3
  2.3 Technical requirements 4

3 Method 7
  3.1 Search strategy 7
  3.2 Resource selection 8
  3.3 Framework 8

4 Result 11
  4.1 Storage and redundancy 11
  4.2 Information propagation 15
  4.3 Authentication and confidentiality 17

5 Design proposal 19

6 Conclusion 21
  6.1 Further research 21

References 23
1 Introduction

Communication is a fundamental need for human beings and the methods and accessibility for this have exploded in the digital age. Not only is communication a need but there is also a desire for confidentiality and knowledge that the conversation is private. There are solutions to this problem such as using decentralized networks and encrypt the traffic which makes sure you do not place your trust on a single server or node and also guarantee that only the intended receiver can read the message. There could however be flaws with these types of systems, they may lack some of the features users have grown to expect from a modern communication platform.

1.1 Background

Instant messaging (IM) is a rather popular tool for communication today [16]. These systems have different approaches in their implementation such as using a centralized (Skype [1], Facebook [2]) or decentralized (Bleep [3], Tox [4]) servers/networks for IM. The goal of an IM system is to provide near real-time communication between the participants such as one to one or multi-user conversations. Another feature that the modern IM application provides is the ability to provide information about presence. This means that the participants have the knowledge of the "online status" the individuals they communicate with have [16].

An increasing number of devices connected to the Internet are mobile devices and users availability or presence may change over time when changing devices or status of clients [20, 16]. When a client is unavailable it cannot receive any messages which means no communication between parties can occur. In such a case applications such as Skype and Facebook will keep the message stored on their servers and send it to the client when it comes back online [2, 1].

There are however problems with centralized solution and relying on a specific company. The users are bound to a set of terms of service and the lose control of their data. They are also a single point of failure which poses a problem an example of this was the shut down of Megaupload [16]. It was a clear demonstration that a third party (in this case a government) could use it's power to close down a communication platform that it deemed a problem. Another potential problem became apparent in 2013 by E. Snowden who leak slides from the intelligence organisation NSA that reviled the secret PRISM program. These carefully outline the extensive surveillance the NSA was carrying out on services such as Skype and Facebook [5, 6].
1.2 Problem description

This thesis focuses on introducing a feature to decentralized IM services that is rarely seen or even non-existent. The ability to communicate with a potential unavailable receiver which is a feature some big centralized applications employs as discussed in Section 1.1. The research question posed is as follows:

**Find a solution to transmit messages between irregularly connected clients in a decentralized network.**

In other words the ability to send a message to someone who is offline and then disconnect and still expect that this message will reach the intended recipient without sending the same message again. The problem boils down to three different components as follows:

- Storage and redundancy
- Information propagation
- Authentication

The background to these sub problems will be discussed in Section 2.3.

1.3 Scope and limitation

The goal of this thesis is to present a possible solution to research question posed. However some parts of the required solution could be analysed in great depth. For this reason the purpose is to create a broad picture of a solution rather than going into very fine details. As discussed in Chapter 2 the result will consists of all the aspects of the problem but on a relative shallow level.

1.4 Disposition

Chapter 2 discuss which components and aspects are important and some fundamental knowledge about relevant real world applications and networks. In Chapter 3 the data gathering method is presented and also the method used for selecting material. Chapter 4 then explain the result of the different components while Chapter 5 presents potential design in form of a scenario where two people, Alice and Bob, tries to communicate with each other asynchronously. This design proposal answers the research question and it’s sub problems. Finally Chapter 6 summarize the work and discuss what relevant future work could be conducted.
2 Theory

A starting point to break down the problem is to look at existing designs for systems and networks that are in one way or another related to the fundamental parts of the problem. As such decentralized networks are interesting, but the rise of cloud computing have also lead to the creating of decentralized versions such as Storj [37] and Box2Box [20].

A lot of research have been done on cloud computing where users can outsource their computing and storage to servers. The main benefit for the user is a more convenient and hassle free user experience where the service provider takes care of the technical details [32]. With this in mind this research can be leverage for solving the research question purposed for this thesis simply because the the areas are very similar.

Interestingly these decentralized cloud networks are using other existing technologies such as employed in Bittorrent (see Section 2.2.1) and Bitcoin (see Section 2.2.2) for sending data and authenticate users [37, 38, 20].

2.1 Decentralized networks

Decentralized networks function with many nodes that are connected and are collaborating to achieve its goal. [40] A popular example is the BitTorrent protocol (Section 2.2.1) which demonstrates the efficiency of decentralized network compared to a centralized one. The reason for this efficiency is that the cost of sending data is distributed among several nodes and even the receiver becomes a distributor when the data have been received. Therefore such a network is less likely to be bottle necked [28]. Decentralized networks are also more robust against system failures such as in the event an access point/node gets shut down [20].

2.2 Applications and systems

The design proposal was formed by looking at existing applications and network solutions that are running in the wild. This means that one should be somewhat familiar or the very least understand the basics of these systems. There for there is a brief explanation of them bellow.

2.2.1 Bittorrent

Bittorrent is a protocol for distributing large files in a decentralized manner. Torrent files are used to locate files which is distributed by peers. A "peer" is a user/client that is currently seeding or downloading the file and is connected to the swarm of clients sharing the same files. The torrent files link to a tracker or several trackers, these are used to store information
about the peers such as IP addresses so new clients can join the swarm. [7]

Trackers are a centralized entity which could be a problem, therefore some newer Torrent clients employ the use of a distributed hash table (DHT) instead. A DHT does the same job as a tracker but is decentralized so no single server is a point of failure for the network (more on this in Section 4.2.1) [8].

2.2.2 Bitcoin

Bitcoin is a decentralized peer to peer network for digital money. The so called cryptocurrency allows it’s user to send and receive money without ever trusting a single entity or person. The network verifies all transactions via a public ledger that is called the blockchain. This ledger contains all transactions that have ever been transmitted and each transaction is authenticated with the help of digital signatures from the sending address [9].

Each full node in the network or so called miners are using CPU power to perform work. This work is scanning for values generated by different hashes. When the work for a block have been done a block cannot be changed without redoing all the work for that block and all blocks that comes after [9, 25].

2.2.3 Storj

Storj is a decentralized cloud storage platform which aims to make it impossible to censor or monitor it’s user while at the same time providing a service with no downtime and high security. The network is made up by users that want to sell storage capacity that they are not using and users who buy storage space that they need. The data is spread on many nodes to mitigate risks of downtime and malicious clients [10, 37].

The network uses the cryptocurrency Florincoin [11] which works much like Bitcoin (Section 2.2.2) does. The cryptocurrency is used as payment for storage and transferring of data between users and the network. [10, 37].

2.3 Technical requirements

A centralized solution such as Skype uses servers to store the messages while users are offline [1]. Considering this the problem of storage is naturally something that needs to be solved which is also why looking at cloud based decentralized networks is a good idea as their purpose is to store data [10, 20].

One inherit danger of decentralized storage is the availability of the data or even the absolute loss of data. This is a result of letting anyone act as a host which means many nodes will have far from optimal uptime. There for redundancy is a very important aspect to consider and is tightly integrated with the storage problem [20].

Another problem that comes up in decentralized systems is information propagation in the network. One part of this is organising the network, since there is no central server or node the network needs to be self-organized where each node make it self available to the network [8, 21]. The other part is the ability to locate specific files or data sets as this is the purpose of for example Bittorrent (Section 2.2.1) or Storj (Section 2.2.3). Evidently these
problems show up in a number of research papers which points to importance of this aspect [28, 21, 15, 19].

As highlighted by Sowmiya and Adimoolam (2014) **authentication** is an important aspect for these types of systems. Only the owner (or in the case of this thesis the receiver) of the data should have access to it [33]. There are also privacy concerns to think about a user should not have to reveal their identity to the public network but still be able to access their files as argued by Irvine D. [17]. Another interesting aspect of the authentication question is of course one brought up in the Bitcoin system (Section 2.2.2). Namely proof of ownership or that you are the real person spending the Bitcoins [25]. This translates to verifying the source of a message sent so that you know who you are talking to.

If these requirements, storage and redundancy, information propagation and authentication is solved, all basic and necessary parts would exist for the research question to be properly answered.
3 Method

There are several frameworks and different ideas to consider when selecting a comprehensive method for a literature review. For example these three methods: "Theoretical background" which aims to give a foundation for the research question when writing a journal article. "Thesis literature review", which constitutes a Chapter in a thesis and lastly "stand-alone literature review" which have the purpose to review already existing literature within the subject area without bringing in new data [26].

Sørensen (1993) argues that you can divide your research approach into four different categories. Where on one side you have a theoretical vs empirical approach which then on the other can be divided into a analytical or constructive result. Furthermore that it is better to have both a theoretical and empirical approach rather then going for one or the other [34].

Another important point is that a review should further expand the knowledge in an area and as such close gaps that exists. But when doing so it is vital to look into the future and identify new areas that can be expanded which will provide academics and humanity as a whole with greater understanding of the subject [36].

For a review to be scientifically rigorous, Okoli (2010) argues for a number of steps that are necessary. An important part of these are to establish the background and outline how to find and select the material that will be used. It is necessary to limit the data to what is relevant but at the same time make sure that nothing important is missed. Which is why one need to be explicit with what criteria to use when selecting different sources and information [26]

3.1 Search strategy

A systematic search should mean that a comprehensive list of information is found and one can estimate this by not getting new concepts within the information found [26].

Webster and Watson (2002) argues that a high-quality paper consists of source material from many different types of sources. Further more that only focusing on a small set of geographic locations or well established publications leads to an incomplete review [36]. One should outline a protocol for location source materials for a consistent result [26], and on that note lets examine what that would look like.

Google scholar should be used as the primary search engine to find material and excluding patents from the searches. The IEEE explore database mainly used for extracting the articles and using the university login to gain access to the content. The search keywords:

Decentralized AND network AND/OR ( chat OR instant message OR cloud OR storage OR "application name" )
Furthermore searching for white papers about different systems using a regular google search with the following search term:

"application name" AND white AND paper

AND and OR in this context should be read as Boolean operators, "application name" should be replaced with the relevant name for the program/network such as Bitcoin, Bit-torrent, Storj.

3.2 Resource selection

There may be hundreds to thousands of articles on any given area of interest so it is important to select the most relevant ones as it would be impossible to review every single one in depth. That’s why an inclusion and exclusion criteria should be constituted to aid the process. The criteria should limit the scope of literature to review to bring it down to a manageable level [26].

Inclusion criteria

Only source material written in English will be considered, articles and white papers that focuses on concepts and systems that are working in practice, IE there are real world implementations.

Exclusion criteria

Source material with only a theoretical idea, untested models. Information that are not accessible online.

3.3 Framework

No method analysed really fit the problem for this thesis and for that reason a custom framework and method was used. However it draws inspiration from different approaches as described in Section 3.

Expanding on Chapter 2 the result needs to include the three categories storage and redundancy, information propagation, authentication. These should be answered by drawing knowledge from existing systems and analyse data and reviews that discuss them. As Section 3 suggests it’s good to use a spectrum of information hence should the use of both articles and white papers be included as outlined in Section 3.1.

By basing the research on information that is grounded in real systems there is some accountability that it actually works in reality. This means that the categories of the research question posed in Section 1.2 can be fully answered. Which in turn means the research question itself will be answered as suggested in Section 2.3.
As seen in figure 1 the work flow starts with identifying the problem and then extracting the pieces out of this problem and creating the theory which this framework is build upon. This will steer the way for establishing the result which will present all critical components for the research question to be answered.
4 Result

This Chapter presents and summarizes a number of methods and techniques used in modern systems which constitutes the results of the review. As mentioned in Chapter 2 the three sub problems to the research question posed in Section 1.2 where storage and redundancy, information propagation and authentication. These areas are outlined below using the method specified in Chapter 3.

4.1 Storage and redundancy

As discussed in Section 2.3 an analysis of decentralized cloud storage solution is a good and natural way to get a better grip of the subject. The Storj project [37] is such a network paired with it’s application MetaDisk [38] make up a supposedly working solution. There for this Section will be about dissecting some of the techniques used to get a clearer picture of how a decentralized cloud can function and more specifically how they store data. Other applications that also store information in a decentralized manner are Bitmessage [35] and Bitcoin which uses the blockchain technology invented by Satoshi Nakamoto (2008) [25].

4.1.1 Blockchain

Blockchain as used in the Bitcoin cryptocurrency is a way to store information, in this case information about the digital currency with the intent to stop double-spending and keeping track of users balance [25].

All the coins are stored in form of transactions inside a so called public ledger which is distributed to all the (back end nodes) miners in the network [27]. Furthermore this public ledger is divided into “blocks” of transactions which are locked using a SHA-256 hash [25] more details on this later.

A transaction here is simply the information stored in the blockchain which for Bitcoin means the senders address, the receivers address and the amount. It also includes a digital signature to prove ownership of the coins from the senders account and the corresponding public key to verify the signature [22].

All the transactions (or information) are broadcasted into the network and spread to all nodes, in reality some nodes may not get a specific transaction but this will be automatically corrected for with help of the chaining of blocks. In any case the transactions are collected and a new block is created but for the block to be added to the public ledger a proof of work needs to be completed [25].

This proof of work is what secures the network from being overtaken by a large malicious attacker. In essence a node or miner needs to find a hash for the block that starts with a number of leading zero bits. The more zeros the harder the problem becomes, when the
solution is found the node will inform the network about the new block and add it to its own ledger. Since the hash also includes the hash from the previous block it is tied to it and there can only ever be one block after another. If there is a conflict, two nodes find the solution for the hash at the same time a split in the blockchain will occur. However, the network is setup to favor the longest chain so when the next block is created it will be tied to one of the “forks” and the other will now be discarded. When the network then again comes to a consensus and longest chain is present in all nodes, all transactions that are confirmed are now present, which is why it didn’t really matter if every transaction reached all nodes (as explained before) [25].

Essentially this proof of work is a one-CPU-one-vote type of system as argued by Nakamoto (2008) which makes it much harder for an adversary to manipulate the network than say basing the voting on IP-addresses. Because of this need to perform extensive CPU calculation an attacker would need access to the majority of the networks CPU power to be able to change past transactions. Furthermore the older the block is, the harder it will become to change it since all blocks are chained, which means that all the work for every single block chained to it needs to be recalculated [25].

![Illustration of blocks with transactions chained after each other](image)

**Figure 2:** Illustration of blocks with transactions chained after each other

To get a better picture let’s examine figure 2, Block 1 represents the initial block, the first ever created. Block 2 is the appended after block 1 after some node have finished the proof of work and generated the hash. Block 3 is the appended to block 2 using the same method and so on.

The blockchain technology is used in other places as well, one of which is the Storj and MetaDisk project. Here instead of storing Bitcoins the information stored is metadata about files that are located in their decentralized cloud. The reason for not storing whole files is simple, it is unfeasible to send large files into a blockchain type of storage. As each node in the network stores every single block ever created storing large files would create a lot of bloat and slow down the network [37, 38].

Furthermore Wilkinson and Lowry (2014) purposed to store merkle roots (see Section 4.1.2) instead of metadata to increase the scalability of the system [37]. Merkle trees is something that also was purposed by Nakamoto (2008) to be used for Bitcoin when blocks get old for the same reason of saving storage space [25].
4.1.2 Merkle tree

In 1979 a man named Ralph Merkle wrote a paper [24] that first introduced the concept, hence the name. A merkle tree is simply a binary tree where each node consists of a hash sum of it’s two children. These hashes goes from the root (merkle root) down to the leafs (merkle leafs), the leafs are the actual information that is hashed. The number of possible pieces of information that we can store is the power of two such as we can denote it to $N = 2^n$[13].

Figure 3: Illustration of a merkle tree

Figure 3 shows a more clear picture of what is going on. The leafs is the information that will be hashed, in the case of for example Bitcoin this information is in form of transactions [25]. The next level contains four nodes where each is the hashed value of the two child leafs. Above that there is two nodes which is represents the hash sum of the two hashes in the nodes bellow (the child nodes). Finally the root node is a hash sum of it’s two child nodes [13].

The concept is pretty straight forward but it does have some interesting properties. As suggested by Becker (2008) the merkle tree is even safe against the power of a quantum computer which make it safe for future technological advancements [13].

Merkle trees are in short a cryptographic summary of the information stored and can be used to prove that no data have been manipulated. Furthermore it can also be used to prove that a node is actually storing a specific piece of data such as in the case of MetaDisk [38].
This is done by issuing challenges to the client where they need to perform a hash calculated on one or several or even all leafs to generate the markle root. If the merkle root does not match the one originally generated for the files it means that the data is not stored in it’s entirety, in other words it may be corrupted, deleted or manipulated [37].

### 4.1.3 Distribution and partitioning

When storing data in general there should be methods used to prevent against failures and loss of information. Centralized storage solutions often employ a RAID (redundant array of independent disks) array systems to protect against disk failures [39]. They may also use N-way data replication or erasure coding to provide redundancy and availability [41].

A decentralized system also needs redundancy as the same problems exists for these systems and on top of that potential unreliable clients, we can take advantage of the schemes used in centralized systems to achieve this protection [39].

An easy way to provide redundancy is a simple replication of the data and distribute it on different nodes. This is an N-way data replication scheme, for example three copies are made and stored on three different nodes. This provides the system of a fault tolerance of two, as in two nodes can lose the data and the system is still able to function normally without any data loss. This can be scaled to N number of copies providing N-1 fault tolerance with the trade off of using N times the required storage capacity [41].

In contrast a data decimation strategy would split the information into N pieces and store them separately. This how ever does not provide any redundancy, in fact all nodes would have to be reachable to receive the entire data stored [39]. The benefit though is less network workload for each node [41].

### M of N

An efficient strategy is for example an M-of-N erasure coding scheme [41]. The data objects are divided into \( m \) fragments and these are stored in \( n \) fragments where \( n > m \), these can then be distributed to different nodes. An important part of erasure coding is that the original data can be reconstructed from any \( m \) fragments [31]. This means that for example a 3-of-5 split can facilitate a fault tolerance of two while at the same time provide a storage usage of only 1.6 times the original file. Compared to the N-way method which ensures the same fault tolerance it consumes quite a substantial less storage space [41].

Another important aspect of a storage system is availability where the data is not only safe from deletion but can be accessed at any time. As Rodrigues and Liskov (2005) argues the M-of-N erasure coding scheme can be used to provide an availability level of four nines, that is 99.99%. Furthermore they argue that a higher availability isn’t really needed. The reason behind this is that other systems that the Internet relies on does not have a higher expected availability. In fact in many cases the number is only three nines (99.9%), there for aiming to get an even higher value will not increase the availability to the service anyway [31].

More importantly erasure coding also requires less bandwidth to restore any lost fragments or nodes to provide the full redundancy level again. As bandwidth is most likely a bigger obstacle to scalability than disk space it provides a pretty good solution all around [31].
Secret sharing

Secret sharing is a erasure coding scheme specifically used in the field of cryptography [39]. Beimel (2011) describes it as a game that involves a dealer with a secret and $n$ number of participants and a collection $k$ of subsets of participants named "access structure". The secret sharing scheme for $k$ is when the dealer distribute parts of the secret to the participants so that some specific criteria is met as follows: any combination of the participants in $k$ can recreate the secret from its parts. No one outside $k$ can reveal any information about the secret [12].

The security in this system is that the information is spread to all the participants or nodes and $k$ number of them need to collude to gain access to the information stored. This scheme can there for be used to store information even without the use of encryption keys based on the assumption that an attacker would not be able to control or fool the subset of $k$ nodes to gain full access [29].

Self healing

Every system needs maintenance over time, a decentralized one is no exception too this. However because of this decentralization the problem becomes more difficult as there is no central point that have control. Therefore the system should be self maintained and be able to detect errors and correct form the automatically [39].

A recover mechanism needs to be in place to handle a node that leaves and/or joins the network. As a node leaves the data it stored needs to be copied from existing nodes (as they have copies of the data) and sent to nodes that are still left in the network. The problem here though is to distinguish a temporary disconnect from a complete departure from the network. As there is no way to make such a distinction another solution is needed. A possible to detect a departure would be to employ a timeout timer that counts the time $t$ between connections. If this time $t$ exceed some threshold $h$ the node would be considered dead [31].

To account for corrupt data and or malicious nodes the merkle tree (see Section 4.1.2) could be used to perform an audit of the stored information as Wilkinson (2014) suggests [37]. If the data is found to be missing it is simply replicated from a node storing an authentic version [31].

4.2 Information propagation

As discussed in Section2.3 information propagation is a required feature for the posed research question. In this Chapter we outline information about how the nodes can find and talk with each other.

A property of decentralized networks that can be leverage when sending data and especially coupled with the redundancy described in Section 4.1.3 is the fact that the information is stored on many nodes. Therefore every $N$ times redundancy the system have it will add $N$ times of locations for that information. Ideally the nodes are located on different network locations and hence will increase the transfer speeds [37].
4.2.1 Distributed hash table

A Distributed Hash Table (DHT) is a Peer-to-Peer (P2P) network that is used for storing contact information to nodes. Examples of use cases for a DHT service is among others, content distribution, distributed file systems and distributed DNS [18]. For example they are actively running in the chat network Tox [4] and file sharing network BitTorrent [8].

Every node in the network has a network unique identifier and these ”node IDs” are chosen at random with the help of a SHA1 hash (160-bit) to ensure no collisions. The node compare it’s ID with other nodes ID to determine the distance or closeness to that node. Node ID’s that are closer gets saved more frequently than IDs far away, this creates a table of increasing detailed information about nodes that are close and only knowledge about few if they are far away [8].

Every routing table maintains this routing table in form of a binary tree where the leafs are ”buckets” containing node ID’s and their contact information, such as IP address and UDP port. Each bucket has a set limit of nodes it can store and can be slip to create further levels in the tree [23].

The table is kept up to date by performing a ping request every $k$ minutes, if a response is received the node is considered good. If how ever no answer got back the status changes to ”questionable”, in this case it is advisable to send a ping request again in the next refresh cycle. A bad node is one that have not responded to many ping requests in a row and will then be discarded from the routing table and replaced with another good node [8].

When first joining the network the node needs to get the address to at least one node, after this connection has been established the node will try to find the nodes with the closest ID by requesting a find node message. Nodes receiving find node message will answer by sending back information about good close nodes in it’s own routing table [8].

The BitTorrent networks DHT network have a get peers request which asks for the location of ”peers” (nodes and locations that are sharing a specific torrent) by sending an ”infohash” of the torrent and the ID of it self. The infohash is a hash of the torrent and can be used to determine which DHT nodes got information about the peers that share a specific piece of data. If the node that gets the get peers request have the information about the peers it sends that back, otherwise it returns the closest nodes it can find in it’s own routing table [8].

4.2.2 Blockchain

The underlying foundation for the Blockchain was discussed in Section 4.1.1 and as described there the entire network maintains a ledger or big set of data that have a perfect replica on every node in the network.

When a Bitcoin node first joins the network it queries some DNS servers to get the contact information for nodes in the network. Once a node have joined the network it will ask it’s neighbors for other nodes and receive announcements from new nodes joining. These are placed in a contact table and can then be used when a new connection needs to be established. The node will try to keep $n$ number of active connections to other nodes and if the number of connections goes below $n$ a new connection is established by using the contact table. There are no real way to leave the network so if an node disconnects it’s neighbor will keep the contact information for many hours before they are discarded [14].
As for propagating the actual blockchain information such as transactions and blocks another method is used. Considering that the size of this information can get considerably large (for this use case) the messages are announced rather than sent in its entirety [14].

The announcement works by sending a hash of the block or transaction to the nodes neighbors (active connections) if the receiving node does not have any knowledge of the block or transaction a getdata message is sent back. This will in turn result with the data being sent to the node. When the data have been received it will perform a verification of the block or transaction. If it is deemed valid the node will announces the new block or transaction it’s neighbors and this continues till the entire network possesses the information [14].

### 4.3 Authentication and confidentiality

To authenticate something we need a way to digitally sign data and then be able to verify that signature and at the same time make sure that signature is only valid for that specific piece of data [30]. Fortunately such technology exists, for example used in Bitcoin when signing transactions [25].

A public, private key scheme is used where the private key is used to sign a hash sum of a previous transaction and the public key of the receiver of Bitcoin. This signature is now tied to these specific information and can be verified with the help the senders public key [25]. This process is shown in figure 4 and as seen all transactions nicely tie into each other creating a long chain.

![Figure 4: The Bitcoin transaction signing process](image)

It relies on a key pair one that is public and can be shared with anyone, and a private one that should be kept secret. The keys can be used to encrypt data and the encryption needs to satisfy the concept of "trap-door one-way function", such that the key used for encrypting the message cannot then be used for decrypting it. How ever the other key not used for
encryption should be able to decipher the data and generate the original content [30].

Let's assume the two users Alice and Bob want to exchange a signed message with each other. Alice who has a message she wants to sign and send can use her private key to create a signature of the message. Alice now has both a signature and the original message but the signature contains the message which means the original message can be discarded and the signature can then be sent to Bob. However, by using Bob's public key Alice can encrypt the signature so that only Bob can open it which means we can introduce confidentiality into the communication [30].

Alice now transfers the encrypted signature to Bob who now are able to open it with his private key and reveal Alice's signature. It is assumed that Bob knows that he is talking to Alice (this information could be encoded with the encrypted data) and can therefore access Alice's public key. This public key can then be used to open the signature and reveal the original message [30].

Bob now has a message signature pair that also match Alice's public key and must therefore have been created by the use of Alice's private key. This can therefore be regarded as proof that Alice was the original sender, in other words, we have authenticated the message [30].

Even though the introduction of confidentiality isn't required as discussed in Section 2.3 it does have some benefits as the participants now can perform a private conversation without anyone snooping.
5 Design proposal

By now we have a good idea of what components can be leveraged in the use for solving the research question posed in Section 1.2. Let’s consider a scenario where we can combine these parts and at the same time the life of a message. First of all we need to assume that there is an underlying chat network that is already in use and that the users already have established the knowledge about each other. We also assume we have two users Alice and Bob, who want to communicate. For this scenario Alice wants to send a message to Bob who is currently offline, after the message is sent Alice will disconnect from the network. After a undefined amount of time Bob then connects to the network again and is able to receive the message sent by Alice.

Alice begins with employing the encryption and authentication scheme described in Section 4.3 and also includes the public key for Bob in readable format so this can later be used to authenticate Bob when he wants to retrieve the message. Alice the divides the message in the fashion of M-of-N erasure coding (see Section 4.1.3) and distributes the shards ideally as close to Bob as possible. The nodes can be found using the DHT protocol as described in Section 4.2.1, Alice also place metadata about the location of the message shards on the DHT network or alternatively the blockchain could be used (see Section 4.1.1 and 4.2.2).

At this point Alice have delivered all required data to the network and can now disconnect safely. While the message is stored in the cloud the network employs the self healing methods described in Section 4.1.3 in case any node storing information would abruptly disappear. The message should probably be time stamped and have an expire date in case Bob never shows up to collect the information.

Bob now decides to connect to the network and perform a look up with help of DHT protocol or alternatively blockchain to find any messages that are bound to him. He then gets the location of the shards and the nodes storing them challenges Bob to authenticate him self. Since the nodes have Bob’s public key stored together with the shards, Bob can use he’s private key to sign some piece of data that the nodes can verify (see Section 4.3). If the nodes verify that it really is Bob who are trying to access the files they send him the data and can now safely delete all shards from the network (and free up the space).

It is probably advisable that Bob sends back an acknowledgment that the message was received so that Alice knows her message got to Bob. Finally the last step for Bob is to simply to decrypt Alice message and enjoy the asynchronous communication.
6 Conclusion

This thesis have focused on solving the problem of sending messages between clients in decentralized networks that could be offline and still let the user expect that the message will be delivered. That have been solved by dissecting the problem into three different sub-problems as discussed in Section 2.3 and then handled individually. A possible solution was found by combining a decentralized cloud (see Section 4.1) with information propagation found in the Bittorrent network (see Section 4.2.1) and using the backbone of Bitcoin the blockchain (see Section 4.1.1 and 4.2.2) technology. Add in the cryptology scheme of public and private keys (see Section 4.3) which makes it possible to both authenticate and ensure privacy. With all these parts solved we have a message storage and delivery system that should work.

6.1 Further research

There are many aspects left out of this paper which would be interesting to analyse further. These aspects may be important but not vital for an initial functional design as purposed in Section 6. In any case further information would strengthen the trust and validity of the design.

First of all potential attack vectors is an important part to look further at. Some aspects of malicious nodes and attacks to the network have been discussed but this is an area where one never can be finished. The security aspect of any system is rather an arms race than anything else, therefore it is always important to analyse the current threats and possible exploits.

A node ranking system is probably not vital but nevertheless an interesting aspect as it cloud provide the network with knowledge of which nodes are historically better than others. With this of course it is important that such a system cannot be gamed as to give a false ranking for a node.

What motivates a user to run a network node? Any decentralized systems that relies on voluntary users needs to have some form of incitement or it would collapse. Question is simple is there anything other than just the benefit of using it that can be introduced to the system?

A deeper analyse of the systems scalability is needed, some of the sub parts have potential scalability issues which is a valid concern. A system and network that aims to have a long life time needs to scale well as new users joins.

Finally this paper is only theoretical even if real systems and methods have been analysed no actual tests was carried out. An important part to move forward is of course to construct a prototype and run the system with real users. Without this step we cannot guarantee that
it would work, another aspect is of course that a system design is pretty useless if it never gets used.
Bibliography


[26] Schabram K. Okoli, C. A guide to conducting a systematic literature review of information systems research. 2010.


[34] C. Sørensen. This is not an article - just some thoughts on how to write one. 1994.


