University of Glasgow | School of Computing Science

Building Applications on the SAFE Network

David Brown

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

Level 4 Project — March 28, 2018

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: _____ Signature: _____

Contents

Chapter 1

Introduction

The SAFE Network is a decentralised data storage and communications network that provides a secure, efficient and low-cost infrastructure for everyone [1].

1.1 The SAFE Network

The SAFE Network[1] is an open-source project being developed by a company in Scotland called Maidsafe[2]. Their aim is to build "The World's First Autonomous Data Network". An "Autonomous Data Network" in simple terms is "...a network that manages all our data and communications without any human intervention and without intermediaries" [2]. The network is comprised of *vaults*. A *vault* is a simple program that anyone can run on their computer. Together, all the *vaults* that comprise the SAFE Network work together to store and serve data. As anyone can run a *vault*, the network can store and serve data in a decentralised manner. Owners of *vaults* are compensated with a cryptocurrency called *Safecoin* which encourages more *vaults* to join the network. A global network that facilitates the decentralised, highly redundant and secure storage of data creates exciting new opportunities for developers.

1.2 Aims

The aim of this project is to not only explore the new avenues for software development that the SAFE Network facilitates, but also comment on the societal impact such a network could have. To gain experience in building applications for the SAFE Network an application called SAFE Wiki has been created. SAFE Wiki aims to provide *permissionless* and decentralised access to Wikipedia, facilitated by storing an archive of it on the SAFE Network. ZIM[3] is a file format that provides a convenient way to store content that comes from the internet. A ZIM file is a self-contained entity that can hold "copies" of entire websites, such as Wikimedia content, for the purposes of viewing them offline. SAFE Wiki will be able to write the ZIM files to the SAFE Network and then provide the capability for anyone to browse them.

Through building the application a perspective can be gained on the architectural and developmental challenges in working with such a new project. With a working product it will then be possible to draw conclusions on whether or not having Wikipedia hosted on the SAFE Network will be useful to people.

[1] https://safenetwork.org
[2] https://maidsafe.net

1.3 Motivation

1.3.1 Technical Impact

Traditional software architectures that are commonly associated with the internet, such as client-server, cannot be used with the SAFE Network. Thus different architectural approaches must be taken when building the software that interacts with the network. It is always stimulating to work with new technology and the SAFE Network definitely promises that opportunity. This report aims to not only convey the experience of working with the SAFE Network, but also outline any possible flaws and issues with both adoption and practical usage.

1.3.2 Cultural Impact

The right to liberty and the unobstructed access to information is one of the most important rights a human can have. Throughout history, a common tactic of oppressive governments is to block access to information. By doing this they try to break down a culture or to control people. The most prominent example of this was the Nazi Book Burning Campaign [4]. The goal of this campaign was to destroy any literature or information that could subvert the ideologies that Nazism is built upon.

Lor and Britz propose that a true "Knowledge Society" cannot be achieved without freedom of information[5]. A "Knowledge Society" is defined by Bindé in "Towards knowledge societies: UNESCO world report"[6] to be a society in which the dissemination of information (knowledge) is open and collaborative. Specifically, the SAFE Network ensures the freedom of access to information. This is a crucial reason why bringing Wikipedia to the SAFE Network makes sense. The benefits to society when citizens are permitted the liberty to seek and consume new ideas cannot be overstated.

> Article 19, Universal Declaration of Human Rights[7]: *Everyone has the right to freedom of opinion and expression; this right includes freedom to hold opinions without interference and to seek, receive and impart information and ideas through any media and regardless of frontiers.*

Chapter 2

The SAFE Network

2.1 Decentralisation

The decentralisation of data storage is one of the core features that the SAFE Network promises. The internet today is very fragile in that most users don't own or control their own data. When you upload a file to Dropbox[1] or OneDrive[2] that file exists solely on the servers that those organisations have control over. Once an organisation has data they can do with it what they wish, acting within the bounds of overcomplicated privacy polices to manage user data. Not only can this incur privacy infringements but it can lead someone into the false sense of security that their data is safe. If someone managed to hack Dropbox or there was a catastrophic failure at the datacenter, a user has no assurances that their data is safe. On a much larger scale companies like Amazon provide AWS[3], an enterprise grade cloud-computing platform. If AWS were to fail, or be the target of a hack, many of the worlds biggest websites would cease to function. This is because of the centralisation of resources. Services like AWS are not easy to target, but they are a single and identifiable part of the equation. A chain is only as strong as its weakest link.

Trust is at the core of decentralisation. Centralised control of resources requires trust in the facilitators of that resource. You have to trust that the resource is free of corruption and indeed trust that it won't be in the future. Companies readily change polices upon acquisition and managerial changes so this trust has to withhold over the course of time. Decentralised models of governance can be used to alleviate these problems. Through autonomous governance, entities such as the SAFE Network can ensure equality to all participants. It achieves this through a system of "trust-less" cooperation, *vaults* on the network do not inherently trust other *vaults*. Every action on the network must reach a quorum before it is considered valid by the *vaults*. This autonomous self-governance is what decentralises the "trust" you must impart on the network.

2.1.1 Peer to Peer vs Client Server

Centralisation of data and computing power is a natural consequence of the client-server architecture that has formed around the internet. It requires trust in the server you are interacting with. When a user want to upload data that trust in the entity becomes a big consideration. The client-server architecture forces centralised governance and inequality between the participants in the network. A Peer to Peer (P2P) network encourages the decentralisation of power. In a P2P network, participants are often of equal standing meaning no node in the

[1] https://www.dropbox.com
[2] https://onedrive.live.com
[3] https://aws.amazon.com/

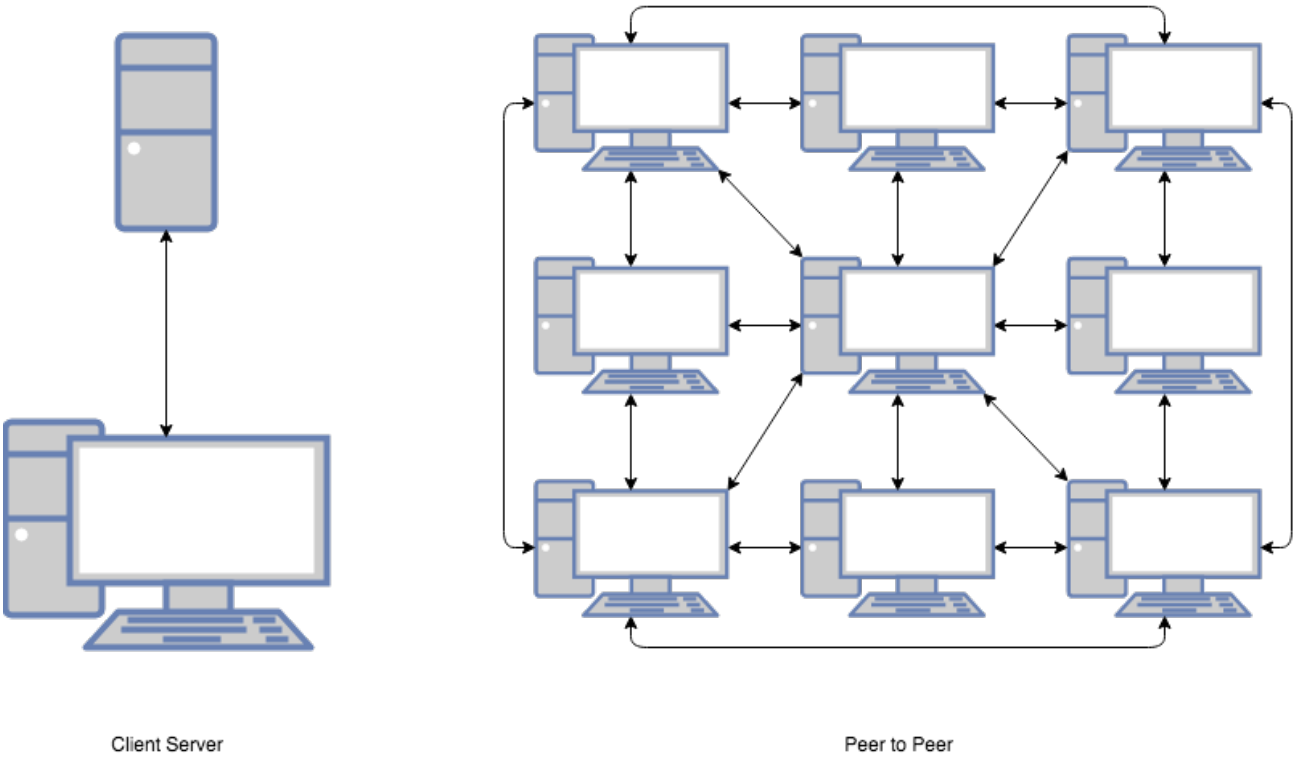

Figure 2.1: Client-Server vs Peer-to-Peer Network

network has more authority than any other node. Thus for a true decentralised network you have to have a P2P architecture to support it.

The SAFE Network thus has to be built around the P2P architectural model. Nodes that comprise the SAFE Network are called *vaults*. *Vaults* are responsible for both storing and serving data, they work together through an autonomous system of governance. Some *vaults* in the network do have more authority than others, these *vaults* are called *elders*. To become an *elder* a *vault* must first prove it is trustworthy and only after a quorum is reached between other vaults can it become an *elder*. A *vault* of this status has more voting power than other vaults, using this power to reach agreements with other *elders* and *vaults* on all network decisions. This decentralised self-governance scheme is crucial to the autonomy and reliability of the network.

2.1.2 BitTorrent

The first stable version of the BitTorrent[8] protocol was released in 2001. Since then it has become one of the worlds most popular means of file sharing, accounting for %3.5 of global internet traffic at the time of writing[9]. In a *permission-less* environment users are allowed to freely share files with one another. As there is no centralised body controlling who has access to what data, the system has been widely used for the piracy of copyrighted material. The SAFE Network aims to solve many of the same problems that BitTorrent does. One of which is moving away from the traditional client-server architecture of file sharing. In BitTorrent, *peers* form what is known as a *swarm*. A *swarm* is all nodes that aim to download a full copy of a piece of data. Data is broken down into discrete chunks, each with a unique hash that allows nodes to uniquely identify each piece of the original file. A node in the *swarm* is known as a *peer* when they don't hold a complete copy of the file. A node in the *swarm* is referred to as a *seed* when they do hold a complete copy of the file. The "resting state" of this network is when all nodes in the *swarm* are *seeds*. Nodes in the *swarm* offer the file chunks that they have to

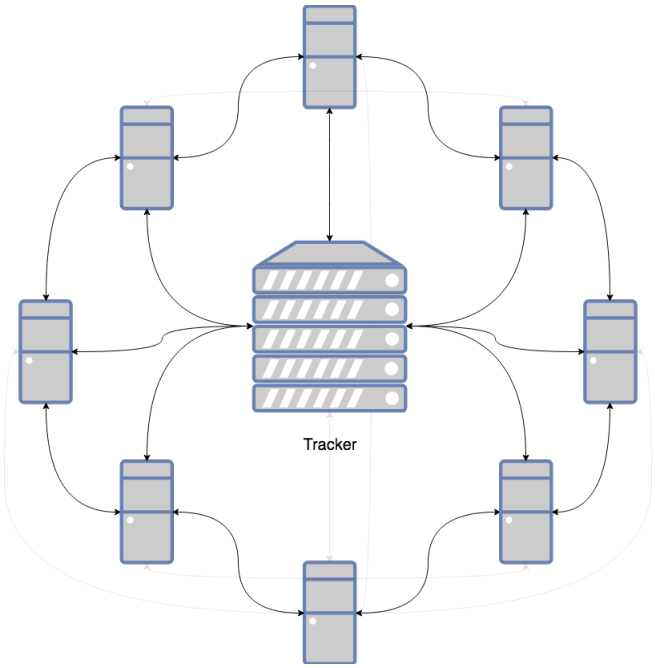

Figure 2.2: Topology of tracker based swarm in BitTorrent

other nodes and then use P2P routing to send the file chunks between one another.

In BitTorrent there is no central server to attack (disregarding a *tracker*, there are *tracker-less* solutions available). The topology of a *tracker* based *swarm* can be seen in Figure 2.2. Nodes can leave and rejoin the *swarm* whenever they want, as long as at least one node has a copy of a specific chunk then all nodes in the *swarm* can spread the data and become *seeders*. This level of data redundancy is a huge benefit to BitTorrent over a traditional client-server model of sharing files.

In the client-server model of file sharing, the owner of the server incurs great cost in the hosting of the file. They have to pay for the management, storage and network costs that are associated with serving that data. For large companies this is often a negligible cost that is not a prohibiting factor in hosting the data. For smaller organisations however (especially non-profits) this server cost can be a big problem. This is a primary reason why many Linux distributions so often provide BitTorrent links to download the operating system. By using BitTorrent they can offload the cost of file sharing onto their users. This works on a "good-samaritan" basis where a node should aim to serve as much data as they themselves have received from the *swarm*. For the vast majority of users this cost is negligible and can act as a "good-will" gesture to help support projects.

Data transfer speeds are a major benefit of using BitTorrent and other P2P file sharing methods. When a node is acting as a *peer*, their download speed is limited to the summation of the upload speeds of the nodes they are connected to. This often means that in a well established *swarm*, download speed is only limited by the nodes internet connection. The more nodes that join the *swarm*, the faster and more resilient to failures the network gets. This is juxtaposed to a client-server model wherein a single connection to the server must be shared by all the nodes wishing to download the data. Thus download speeds are limited by the resources of that single server.

BitTorrent may solve many issues surrounding the distribution of files, but falls short of solving the decentralisation of the internet as a whole. The main limitation being that data on BitTorrent is not mutable. Once a file has been spread to a *swarm* it is an immutable entity that cannot be changed. This means it is extremely challenging to use BitTorrent for dynamic content such as websites. Immutability is a big drawback depending on the use case, thus the option to support both Mutable and Immutable data is beneficial. The other attributing factor is that data only exists inside *swarms*, a node cannot interact with data without first joining the relevant

swarm. So the discoverability of data is also an issue. These are all points that the SAFE Network offers solutions to.

2.2 Serverless Architecture

The idea behind a serverless architecture is to move as much computation and functionality to the client as possible. As time passes, the computational power of user workstations gets faster. This computational power goes wasted for the most part. While browsing the internet, interacting with Facebook for instance, a computer actually does very little in terms of processing the information a user sees. Facebook serves to the browser a thin-client which can then make requests to Facebook for the data that is needed. Thus there is untapped potential on the client to perform more work locally instead of doing it on a server.

There are drawbacks in offloading work to the client. The first issue, especially with websites like Facebook, is privacy. When Facebook processes data on their servers, they can assure that only data a user is allowed to see is sent to the client. If all data had to be processed locally it would introduce new challenges in data protection and security. Another drawback is that of mobile devices. On laptops, smartphones, tablets and IoT devices, power consumption is a major problem. Thus by using the client-sever model, electricity expensive computation can be offloaded to the server and reduce power consumption on the client. This is in combination with reducing network traffic to the client, which for mobile devices is a crucial factor in battery life. Thus computation and power consumption on mobile devices means that client-server makes more sense than the serverless architectural model.

Vaults serve a similar purpose to *peers* in BitTorrent. Note that they do not serve a similar function to *seeds*, the SAFE Network aims to never keep a complete copy of a single file stored on one *vault*. The SAFE Network in this capacity can then facilitate file sharing just like BitTorrent. Additionally what the SAFE Network has is the ability to route requests throughout the entire network. All *vaults* in the network have the knowledge required to find any chunk of data. This is different to a node in BitTorrent because it can only find a chunk of data within the *swarms* it knows about. As this dynamic routing exists, the SAFE Network has a form of DNS that can be used. Another major difference is that the SAFE Network is capable of mutable data. This means that the SAFE Network is fully capable of supporting dynamic websites, forums, email and other such applications. A user can open a browser that is capable of connectivity with the SAFE Network and browse the *internet* just as they would normally.

Vaults cannot process data thus SAFE Network applications must process data locally and only use the network as a storage and transport medium. Thus the serverless architectural model is a good fit for the SAFE Network. This style of building websites and applications has been around for a long time. With the advent of JavaScript and other such technologies, it was possible to run code locally through the browser without needing the server to do any processing. A good example is online mini-games, the code runs locally and there is no processing required on the server. The JavaScript/Flash/Java/... is served to the client and the processing is performed locally. Another example of this are online "office suites", they are very powerful programs that can be ran through the browser. They depend heavily on the processing power of the client to provide an experience similar to that of a desktop application.

Unless the SAFE Network is just used as a component in the stack, it forces the serverless architecture model to be used. This introduces challenges in how applications are designed and built. As development is entirely focused on the client, development time on the back-end can be eliminated. Instead of designing websites and applications the *traditional* way, they must be developed as fat-clients. Websites will become heavier, requiring more care and optimisations. Messy and slow JavaScript is common on the internet today, mostly due to the abundance of computing power that exists. As discussed previously, battery life is an important consideration when offloading work to the client. This means that applications/websites that follow the serverless architectural

model must be well optimised for power consumption. A new technology called Web Assembly is gaining traction and could help alleviate these issues.

2.2.1 Web Assembly

Web Assembly is an assembly-like language that can be compiled to from C, C++, Rust, etc, and then ran inside web browsers. It allows code to be written in high-level languages (that aren't interpreted like JavaScript) and then ran on the client with *near native* performance. This has big implications for the internet as a whole, not just the SAFE Network. A technology like Web Assembly could therefore be extremely useful when building websites that use the serverless architectural model. A big strength of Web Assembly is in the processing of data. Optimised and high performance code to process data can be more easily written in languages like C than in languages like JavaScript. Since the SAFE Network serves raw data to the client, the use of Web Assembly to process that data could be of huge benefit.

2.2.2 Serverless Fat Clients

Although not all fat-clients follow the serverless architectural model, applications that are designed to be serverless are inherently fat-clients. Hence because of the points mentioned previously, the SAFE Network encourages (almost requires) the fat-client style of architecture to be used for software development. As web technologies mature they become more suitable for the development of fat-clients. Instead of having to download desktop applications to a device, new technologies allow powerful applications to be built and delivered through the web. If delivering fat-client applications through the web was not possible, users would have to download apps for popular services like YouTube, Facebook, Twitter, etc. Hence the advent of new web technologies, such as Web Assembly, is an enabler in the success of the SAFE Network.

2.3 Ownership of Data

Accessing the SAFE Network is *permission-less*. What this means is that users don't need to go to a central body that controls the network to ask for (register) an account. A user simply connects to the network and is allowed to create one. All users of the network are equal, there is no concept of "admin" accounts. When a user uploads a piece of data to the network, it can either be "public" data or encrypted data. Note that all data stored by *vaults* is encrypted, if the data is "public" then it simply means that the decryption key is publicly derivable so anyone can access the data.

Ownership of data is one feature that the SAFE network provides as apposed to a system like BitTorrent. All nodes in BitTorrent have equal rights to data, so a node can't simply delete or edit a file once it has been spread to the *swarm*. In the SAFE Network, the ownership of data is more clearly defined. When data is uploaded it is split into chunks and distributed across different *vaults*. That data has an identifiable "owner", the account that originally uploaded it to the network and has the relevant decryption keys. A special type of data called Mutable Data can be edited and controlled by its "owner" while still being stored and managed by the network. The "owner" maintains "control" over that data.

In the SAFE Network users "own" and control their own data, nobody else can edit or tamper with it. This does however incur issues surrounding the distribution of illicit and copyrighted content. One could imagine a user uploading the latest Hollywood blockbuster and sharing the link to it. As data cannot be deleted that file will be available forever. A solution to this could be through the use of a *master-key* to the network. This would be a key that would allow complete access to the network and unlock the ability to delete data. Such a key would be

beneficial in removing illicit content but completely invalidates the principles of the SAFE Network. This issue of control has made a huge impact upon BitTorrent. Numerous court cases and law suits have been issued since its inception due to its use for illicit content distribution. This is a situation of "you can't have your cake and eat it to". It is impossible to ensure complete security of user data and then undermine that with the ability of a central body to tamper with it. Like BitTorrent the SAFE Network is a tool. Users will do with it what they wish. The mitigation against those who wish to use a tool for illegal activities should not impact upon those who follow the law.

2.4 Architecture of the SAFE Network

The SAFE Network is still in active development. At the time of writing, the SAFE Network is currently on its second alpha revision (Alpha 2) out of a planned four. Thus the architecture of the SAFE Network is still subject to rapid change. Chapter 3 explains the architecture of the SAFE Network at the time of writing.

Chapter 3

The Architecture of the SAFE Network

The internet is constantly growing and changing. Changes in technologies slowly permeate throughout the network as if by osmosis. Governmental policy can have a large impact on how people interact with the network, whether that be Turkey blocking Wikipedia[10] or the US abandoning Net Neutrality. This area is where the SAFE Network starts to deviate greatly from the *traditional* internet. The SAFE Network is a "Autonomous Data Network". To have access to the SAFE Network means to have access to all of it. A government cannot curate access data. This is made possible by the architecture of the SAFE Network.

3.1 Vaults and Clients

The SAFE Network is comprised of *vaults*. A *vault* is a singular program/application that a user runs on their computer, whether that be a server hosted in a datacenter or a desktop computer. A *vault* is given a set amount of storage by the user which it then uses to *farm* data. For a given *vault* to join the network, it must pass a "Proof of Resource" test. This test is used to validate that the *vault* has enough bandwidth and CPU power to be able to adequately perform its job. Similar to how a real world farmer looks after their crop/animals, a *farmer* (*vault*) on the SAFE Network looks after data. Understanding that nomenclature is quite useful in understanding the function a *farmer* (*vault*) serves. When a *vault* successfully serves the data it is storing, it is rewarded with *Safecoin*. A cryptocurrency hosted on the SAFE Network. Reading data from the network doesn't incur any cost, it is only when writing data that a user (*client*) has to expend *Safecoin*. A user doesn't need to run their own *vault* to interact with the network, all users can connect through the use of a lightweight *client*.

The only time a user interacts with their *vault* is through configuration before startup. The most notable configuration being the allocation of storage for the *vault* to use. Once *vaults* start communicating with each other, humans have no control over their operation. The network autonomously votes and decides many factors which includes everything from where data should be stored to how much value a single *Safecoin* has. This is the autonomy of the network, it does not accept governance by humans and *vaults* cooperate for the good of the entire network.

3.2 Immutable and Mutable Data

Similar to BitTorrent, data is broken down into chunks. Each chunk of data that is stored on the SAFE Network is at most 1MB in size and has a unique 256-Bit Address. This allows every chunk of data to be uniquely identified and helps *vaults* to decide who stores the data. Data stored on the SAFE Network can take one of two forms.

It can either be *Immutable Data* or *Mutable Data*. A Mutable Data structure is a *key value* storage mechanism that allows for the storage of one thousand entries. An Immutable Data structure only stores a single "value", its address derived from the hash of the binary data it contains. An Immutable Data structure can itself only be 1MB in size, but through the use of a *data map* (Section 3.2.1) this limit can be subverted. As their names imply, Mutable Data can be freely mutated whereas Immutable Data cannot. It is this property of Immutable Data that eliminates duplication on the network. For example, if Bob uploads a picture to the network he is presented with the address of that file (the address of a *data map*) and will have the relevant keys to access it. If Alice then uploads the exact same picture the data is not duplicated, she is simply presented with another *data map* to the data. If either Bob or Alice chooses to "delete" the picture, then they simply "forget" how to access their *data map*. This means that if someone has access to a piece of data then that ability will never be revoked.

3.2.1 Self Encryption and Data Maps

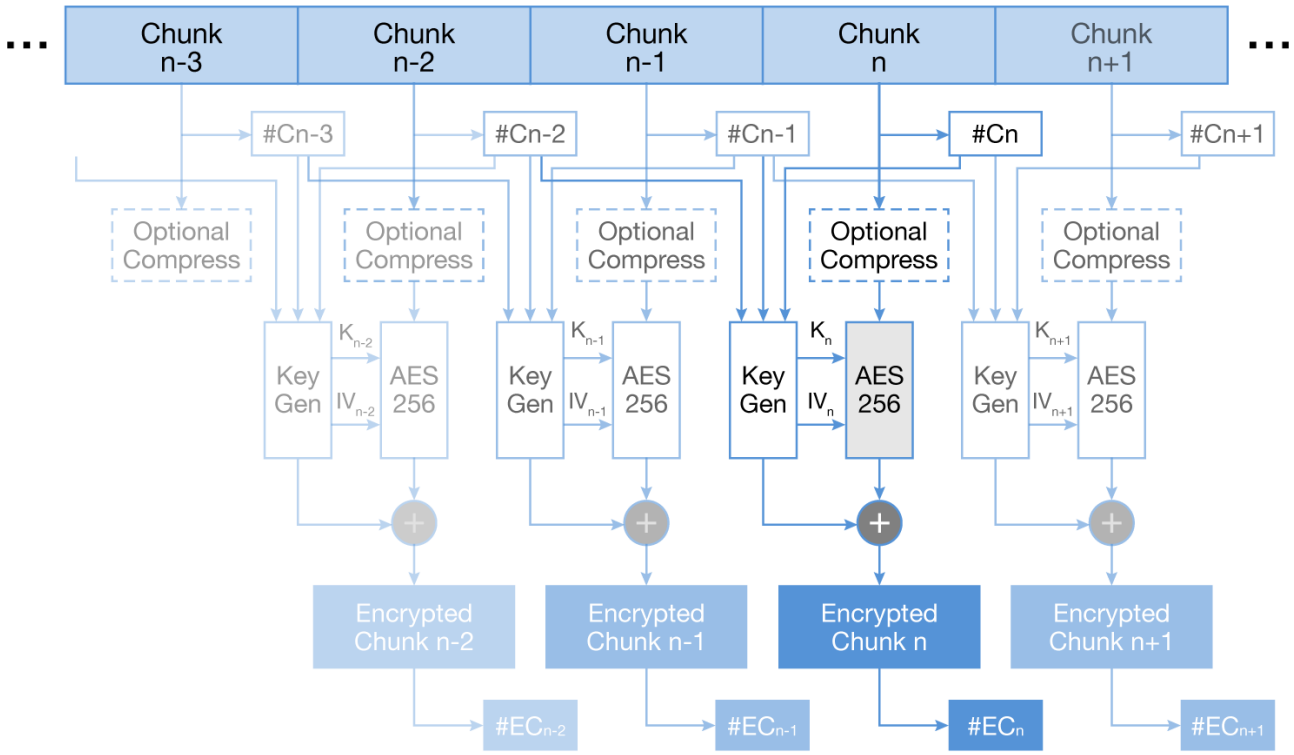

Figure 3.1: The *Self Encryption* process [1]

Immutable Data that is stored on the SAFE Network goes through a process called *Self-Encryption*[11]. During *self-encryption* data is broken down into chunks of a specified size (The SAFE Network uses 1MB chunks). To be able to reassemble and read the data a structure known as a *data map* is created during the process. The *data map* contains several pieces of information:

- chunk_num u32: Specifies how many chunks of data are within the Data Map
- hash Vec<u8>: Post-encryption hash of chunks
- pre_hash Vec<u8>: Pre-encryption hash of chunks

[1] Sourced from https://safenetwork.wiki/en/Encrypt

- source_size u64: The size of the original piece of data, before any encryption has taken place

The crucial structures of this *data map* are the two vectors that store the pre-encryption and post-encryption hashes of chunks. What happens first is a piece of data is broken down into chunks and then hashed. The hash of the chunks is stored in the 'pre_hash Vec<u8>' of the *data map*. This list is important as it defines the original piece of data before being encrypted. Each chunk then goes through an encryption process using AES 256. The key used for each chunk is the pre-encryption hash of one of the other chunks. The chunks then go through another step that further obfuscates the data. This step involves XOR'ing the data with the pre-encryption hash of other chunks. A final hash is then taken of each chunk and stored in the 'hash Vec<u8>' of the *data map*. A diagram illustrating *self-encryption* can be seen in Figure 3.1.

Self-Encryption is a generic process, there is nothing about it that specifically ties it to the SAFE Network. *Self-Encryption* is used to obfuscate data for storage, it is only with access to the *data map* that you can make sense of the data. It is because of this obfuscation process that *vaults* cannot distinguish what the chunks they are storing actually contain. It is just "garbage" data to them. This means chunks can be freely distributed across the network with the assurance that only the person with access to the *data map* can read the original data. To secure access to the *data map* it can be encrypted using either a symmetric key or an asymmetric key-pair. With different key-exchange mechanisms users can then freely share access to *data maps* with one another.

3.2.2 Address Collision

The SAFE Network stores chunks based on 256-Bit addresses. Although the probabilities are unfathomably small, collisions in the post-encryption hash of chunks from *self-encryption* is possible. The hash algorithm used currently is *SHA3-256*[12] which was first published in 2015. *SHA3-256* is an incredibly "safe" hashing algorithm to use, papers like "Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3"[13] do propose methods on how to "break" the algorithm but their findings indicate that even with Quantum computers an attack would be practically unfeasible. The question as to why use *SHA3-256* when stronger algorithms exist within the same family is however curious. NIST themselves currently recommend[14] that for generic hashing applications *SHA3-512* is the minimum algorithm that should be used. This brings into question why Maidsafe have chosen a less secure algorithm. If the choice came down to performance then this is an oversight that could hurt them in the long run. Moore's law[15] says that as time passes computers become more and more capable. If the SAFE Network truly wants to exist for a long period of time, choosing *SHA3-256* as the hashing algorithm the entire network is based upon seams like a naive approach. The suggestion to use the strongest algorithm that exists today is thus highly encouraged, a switch to *SHA3-512* seams like a logical step. Especially given the fact that currently the network doesn't handle hash collisions, it simply writes over any collisions that occur.

3.2.3 True Immutability of Data

Once Immutable Data has been written to the network it can never be changed or deleted. The chunks of data that are the result of *self-encryption* will be stored by the network indefinitely. This means that to "delete" data more accurately means to "forget how to access it". The implication of this is that once data is uploaded, access to the *data map* is the only thing that stands between being able to access the data and it being lost forever. Not having any mechanism to actually delete data does pose problems.

3.2.4 Mutable Data

A Mutable Data Structure facilitates a powerful "permission scheme" to data and is controlled by the user who creates it. Unlike Immutable Data, the address of Mutable Data doesn't have to correspond to a hash. For Immutable Data, the location of the "data" is actually the address of the chunk that contains the *data map*. The hash of the original file is the address of that chunk. There is no such limit on Mutable Data and hence can be stored at any empty address. As the name implies, the special thing about Mutable Data is that it can be mutated.

The permission scheme around MD is what gives it its utility. The "owner" of the MD can specify exactly who can perform what actions to the MD structure. The generic actions that can be performed are: insert, update, delete and change permissions. To give an example, a user could have an email address on the network that has an "inbox" MD structure. Everyone would then be granted the permission to "insert" to the MD, all other permissions would only be granted to its "owner". Using asymmetric encryption they could then encrypt the "inserts" to the structure so that only the "owner" of the MD is able to read the entries. As MD structures don't need to live in a chunk thats address corresponds to the hash of the chunk, they can be at custom locations. So to find the "inbox" for someone is as simple as taking the SHA3-256 hash of their username/email-address and then "inserting" messages to the corresponding MD.

It is this permission scheme that allows complex and dynamic applications to be built whilst giving users the ability to still control their "own" data. As MD structures can be mutated, they can be used to direct clients towards files that are Immutable Data. For instance, if a user visits a website the MD that points to the HTML, CSS, etc, can be found at the SHA3-256 of the URL. As it is a MD the "owners" of the website can simply update this index to point to newer revisions of the website files when they wish to update the website.

3.3 Disjoint Sections

The unique address of every 1MB chunk of data is used to determine what *vaults* are responsible for storing it. Maidsafe's innovation was in the creation of what are called *Disjoint Sections*. These *sections* are groups of *vaults* that are responsible for a certain range of the 256-Bit address space. By default, the network requires a minimum number of *vaults* to sustain the network. At the time of writing this is eight *vaults*. These eight *vaults* form a complete *section* and are responsible for the storage of the entire 256-Bit address range. As more *vaults* join the network, this *section* will grow in size and then eventually split into two new *sections*. There are numerous requirements that have to be met before a *section split* is allowed.

After a split, *sections* are then responsible for half of the 256-Bit address range that they were before. As more complete *groups* of eight *vaults* join the network, it continues to split and each *section* is therefore responsible for less data. An important thing to note is that the SAFE Network doesn't assign 256-Bit addresses based on proximity. In a given *section* two *vaults* could be located on different continents. This property helps the integrity of the network by ensuring *vaults* in a given *section* are not located close to one another. Otherwise the network could be open to simple attacks. For example, start eight *vaults* on a single computer to form a *section* then switch them all off. The loss of an entire *section* could result in data loss. If a significant number of *vaults* leave the network then *sections* have the ability to merge with other *sections* to ensure the stability of data is maintained. In Figure 3.2 four *sections* comprised of four *vaults* can be seen, the address range that each *section* is responsible for can be seen beside them. In the diagram four *vaults* make a *section* instead of the traditional eight, this is just to make the diagram easier to process.

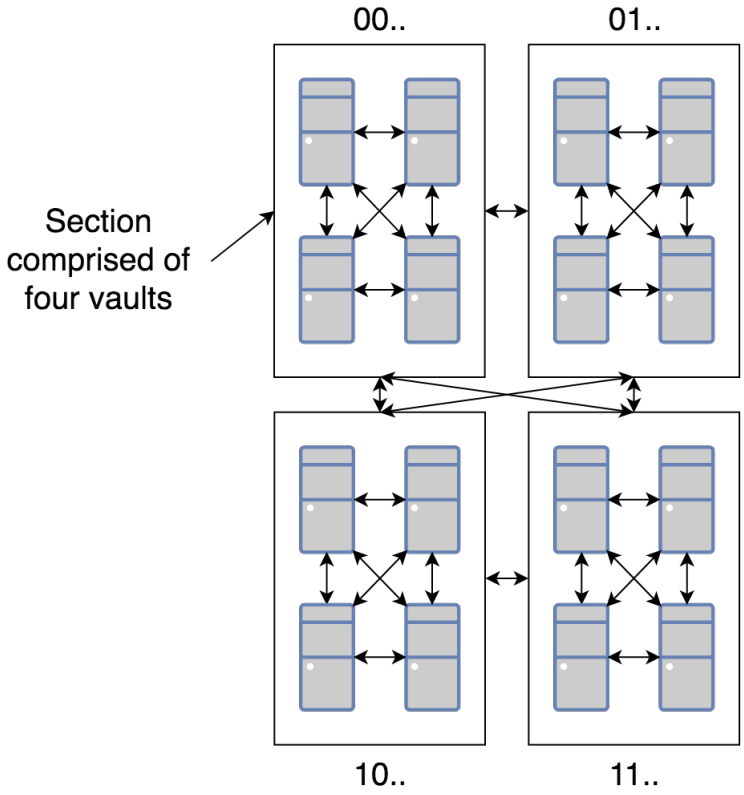

Figure 3.2: Four sections of a SAFE Network. You can see the address range each section is responsible for.

3.4 Proof of Resource

The *proof of resource* (PoR) test is used to validate the effectiveness of a *vaults* ability to store and serve data. PoR is used during certain events, such as a *vault* joining the network, to validate that it can adequately perform its job. *Vaults* will further be asked to perform a PoR during random network events, this means that *vaults* are continually tested to check that they haven't lost the ability to serve the network.

3.5 Personas

Vaults can be characterised as having different *personas*. One such *persona* is the *Data Manager*. A *Data Manager* is responsible for the curation of data chunks within a given *section*. When data is stored on the network, it is replicated across multiple *Data Managers* in a *section*. At all times the network aims to keep a minimum number of copies of a chunk of data, if a copy goes missing then it is replicated to a *Data Manager* to ensure that data is stored redundantly. Hence within a given section, there will be several *vaults* storing identical chunks of data. Each having full knowledge of the chunks of data that the other *Data Manager's* hold. This scheme means that no *vault* will ever hold the single copy of a chunk of data.

Another important *persona* is that of the *Client Manager*. A *Client Manager* is responsible for storing the account data for clients that fall within its address space. It is also responsible for liaising with the rest of the network on behalf of that account. When a user creates an account on the SAFE Network the data is stored on the network just like any other piece of data. It has a given 256-Bit address and contains information like: how much *Safecoin* it has, the number of chunks of data that has been uploaded, etc.

3.6 Accounts

Accounts on the SAFE Network are not inherently special in that they are stored along with other pieces of data on the network. There is no centralised body or organisation that is needed to grant access to the network. An account on the SAFE Network is derived from two parts, an *Account Secret* and a *Account Password*. The *Account Secret* identifies where the account is stored then the *Account Password* is used to encrypt and decrypt that information. With these two components a user can gain access to a piece of data that represents their account. This data structure contains everything needed for an account:

- The Safecoin balance of the account
- Address(es) of *data maps* that the account has decryption keys for
- Decryption keys for the aforementioned *data maps*
- Other account information

An important caveat that comes from having a single set of credentials is that once lost, a user cannot access their account again. This is not a real problem for "professional" users as most make use of password managers and other such tools. At some point though even the most "pro" user could lose track of a password and with that their data is gone. There is no chance of recovery. This could involve losing irreplaceable data which can be heart-breaking for individuals and bankruptcy causing for companies. Thus keeping credentials safe is extremely important. This could be a prohibiting factor to a lot of users however, some may not want to take the risk.

3.7 Crust and Encryption

Crust is the secure routing layer designed and built by Maidsafe to provide the secure communications backbone of the SAFE Network. *Crust* allows for reliable P2P connections and provides encryption for all traffic. Several transmission protocols can be used, falling back to UDP from TCP (for example) if required. Encryption at this level means that data in transit is always encrypted, no matter what.

When a client connects to the network they do so through the use of a *Proxy Node*. A *proxy node* is a *vault* that is used to liaise between a client and the network at large. The *proxy node* is used to hide the clients IP address from the rest of the network. Deeper into the network, *vaults* communicate directly with this *proxy node* and not with client. Hence through the use of a *proxy node*, the real world identity of the client is well hidden from the rest of the network. This means that a given *vault* cannot detect that the data it is sending is going to a certain geographical location. The topology of how a client connects to the SAFE Network can be seen in Figure 3.3.

3.8 Network Incentives, Safecoin and Farming

Safecoin[16] is the cryptocurrency of the SAFE Network. When a user creates an account on the network, they are given a *Safecoin* wallet. This wallet can be used to securely store the *Safecoin* that belongs to the account. Akin to *Bitcoin*, *Safecoin* is a cryptocurrency that can be sent between users in a permission-less manner. *Safecoin* can be sourced in a number of ways including through *farming* and by simply purchasing them with another currency. The ultimate purpose of *Safecoin* is to incentivise people to run *vaults*. Running a *vault* is costly, so the reward of *Safecoin* is used to incentivise the participation of nodes in the network. The expectation

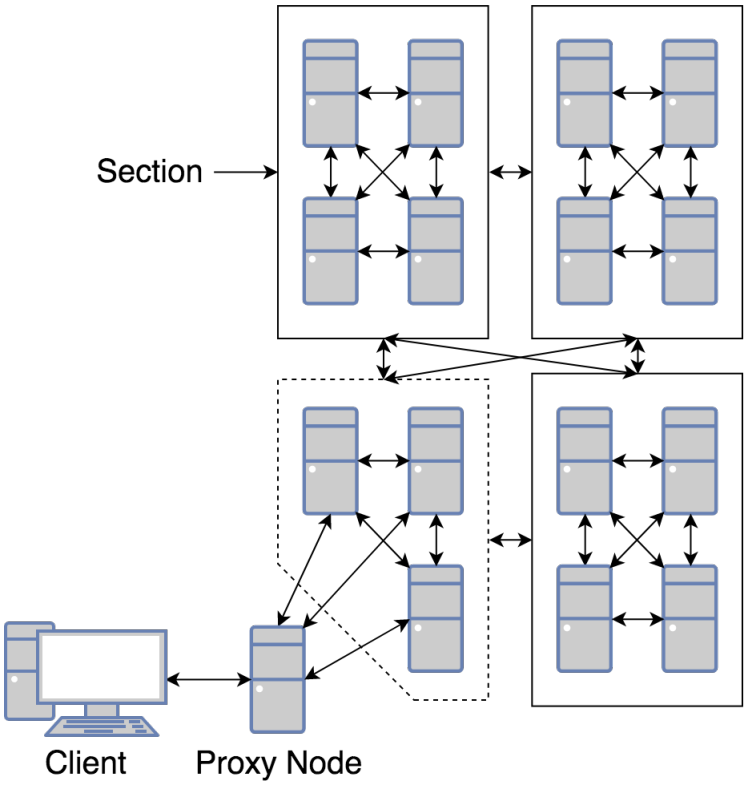

Figure 3.3: A client connecting to the SAFE Network through a Proxy Node

is that as the cost of CPU/storage falls with time, the value of *Safecoin* will increase. Value in this context is how much data each coin facilitates the storage of.

When a client requests a chunk of data, the *vault* that successfully returns that piece of data will be given the opportunity to earn *Safecoin*. The probability of being awarded this attempt to earn *Safecoin* is determined by the *farming rate* of the network at that specific moment in time. The *farming rate* is used to balance supply and demand of data on the network. The SAFE Network tries at all times to keep a minimum amount of network capacity free, this is around %30. When capacity starts to fall below this threshold then the *farming rate* will increase, meaning *vaults* have the opportunity to earn more *Safecoin*. This works in the opposite direction too. If the network grows so that there is an overabundance of capacity then the *farming rate* will decrease meaning *vaults* earn less money. This constantly varying *farming rate* is hence used to balance network resources and de-incentivise users from starting new *vaults* when they are not needed.

The utility of *Safecoin* is not tied to whatever its "exchange price" is. Overtime, the storage "buying power" of each *Safecoin* should increase as computing power and storage becomes cheaper. *Safecoin* is in its infancy and indeed has not yet been implemented, which means it is subject to changes from what has been described above.

3.9 Quorum and the Datachain

As the network acts as an autonomous entity there has to be some method for a given *vault* to reach consensus with other *vaults*. This problem is what cryptocurrencies aim to solve through processes such as mining. *Mining* is essentially the network reaching consensus upon what has happened. In the case of *Bitcoin*, every time a block is *mined* it is cryptographically linked to the block that came before it. As this *Blockchain* grows in size the

consensus on past transactions becomes stronger. The SAFE Network needs a similar mechanism on how to reach consensus. Analogous to a *Blockchain*, the SAFE Network has a *Datachain*.

The Datachain is used to help insure the integrity of the network and can be used to help rebuild it in the case of a catastrophic failure. For any action on the network to be valid, whether this be the storing/mutation of data or a *vault* joining a *section*, there has to be a corresponding *group signature*. This *group signature* is stored in the *Datachain* that all *vaults* in a *section* have. In order for an action to be considered valid, a *section* has to reach a quorum. For a network where the minimum *section* size is eight, a quorum would be five out of the eight vaults voting in agreement. This means that in a given *section*, several *vaults* could be acting as "bad parties" but network integrity wouldn't be lost. The closer two *sections* are in 256-Bit XOR address space the more they know about the data the other is storing. They will have access to the portion of the *Datachain* that is used by that *section* to record data writes and mutations. This way a given *section* can help to verify that a neighbour is acting as a "good party" and that data being stored there has not been tampered with. The further away two *sections* are in 256-Bit address space the less they know about each other. This means that as the number of *sections* increases, the influence a given *section* has over the network decreases. Eventually resulting in no *section* having a complete overview of the entire network.

A protection mechanism exists for when a *vault* tampers with data after it has been recored in the *Datachain*. When a client requests a given piece of data, a single *vault* is chosen to return that chunk of data. Alongside the data that is returned, a number of acknowledgements from other *vaults* in the *section* must be returned too. This way, a client can then verify the data they receive against the acknowledgements from the other *vaults* in order to ensure that the data is valid.

The development of the *Datachain* is still very active, thus implementation details are subject to rapid change.

3.9.1 Node Age and Churn

For a *vault* to vote on network activity it has to have proven itself reliable. A *vault* cannot just join the network and start voting in network decisions. When a new *vault* announces itself to the network, it is issued with the POR test that was discussed in Section 3.4. If it passes then as long as the assigned *section* reaches a quorum the *vault* will join that *section* and it will be recorded in the *Datachain*. The new *vault* is very "young" in the eyes of the network and as such cannot be trusted. It is not allowed to vote in group actions and is responsible only for the storage and transmission of data.

Churn is used to continuously "rotate" vaults round different *sections* on the network. This means that in a given time frame, a *vault* will not be responsible for the same 256-Bit address range. This important feature helps to ensure that it is very difficult to track down where data is stored in order to erase it or corrupt it. During *churn*, young *vaults* with a lower *node age* will be chosen more frequently than older *vaults*. The *vaults* chosen are assigned to new *sections* to which they must provide another PoR test to be allowed to join. If the new *section* reaches a quorum then the *vault* joins and its *node age* is incremented. Thus, trust must be earned by acting as a "good party" in the network over time. Only when a *vault* reaches a certain *node age* does it become an *elder*. An *elder* is a node which has the highest possible *node age*, meaning it has has proven itself to be a reliable party over the course of time. When a *vault* is an *elder*, it gains the voting rights that lead to the construction and maintenance of the *Datachain*. If a *vault* acts out of order then its *node age* can be decremented or eliminated entirely. Trust must be earned.

Node ageing and *churn* are hence essential security features of the network and make it very difficult for an attacker to have any choice in the *section* of the network they wish to attack.

3.10 Opportunistic Caching

If a *vault* is involved in the routing of a chunk of data that it sees frequently, it has the ability to cache that chunk. Owing to the architecture of the SAFE Network, content cannot "go down" due to high traffic. The network will respond autonomously to the higher volume of traffic and dynamically increase its potential to serve that content.

Another benefit of this caching is geographical proximity. A good example is that of a website that has support for multiple languages. Due to caching, chunks that correspond to particular languages will be cached in the *vaults* nearest to where they are being consumed. For instance, the chunks that correspond to Japanese content will more likely be cached in *vaults* near Japan as that is where they are being fetched from the most.

Chapter 4

Implementation of SAFE Wiki

SAFE Wiki is an application that facilitates the storage and browsing of ZIM files on the SAFE Network.

4.1 Kiwix

Kiwix first launched in 2007 as a way to browse the internet "offline". It achieves this through the use of ZIM files which are suitable for storing most HTML based content. One of the primary goals of Kiwix is to allow users to browse Wikipedia and other projects from the Wikimedia foundation without an internet connection. Whether this be in the middle of the ocean, Africa or even inside North Korea. Since the initial launch, different versions of the software have been released. Versions support many different platforms including: iOS, Android, Windows Phone, FireFoxOS, macOS, Windows and Linux. A user opens the app and then through a file explorer (or other means) selects the target ZIM file. Kiwix then presents the user with an almost "web browser like" experience. With resources like Wikipedia it looks uncannily like the "real" thing. Users have the ability to follow hyperlinks around the website (ZIM file) and search for pages. The London page of Wikivoyage can be seen in *Figure 4.1*. With such a fantastic history behind the project, Kiwix was seen as the natural foundation to build SAFE Wiki upon. Kiwix is inherently a fat-client style of application as all processing is done on the client. Hence building upon an already fat-client application made perfect sense considering the points made in Chapter 2.

4.1.1 Kiwix JS

Kiwix JS is a JavaScript variant of Kiwix, originally part of the Evopedia project it presents Kiwix in the form of a browser extension. This extension has support for many different environments (FireFox, Chrome, Edge, etc) due to the portable nature of JavaScript.

As the SAFE Network is still very much in its infancy the developer API's reflect this. At the time of writing the only API's that are ready for use are the *Node.js API* and what they call the *DOM API*. The *DOM API* can be used to build websites to interact with the SAFE Network whereas the *Node.js API* facilitates the development of desktop applications. Both of these require the usage of JavaScript, hence forking Kiwix JS to build SAFE Wiki made logical sense.

Kiwix JS as it stands has support for Wikimedia and StackOverflow ZIM files (although others may work, just not supported). This meant that through SAFE Wiki it would be possible to not only browse Wikimedia content but also content coming from StackOverflow. The content that users would be able to browse would

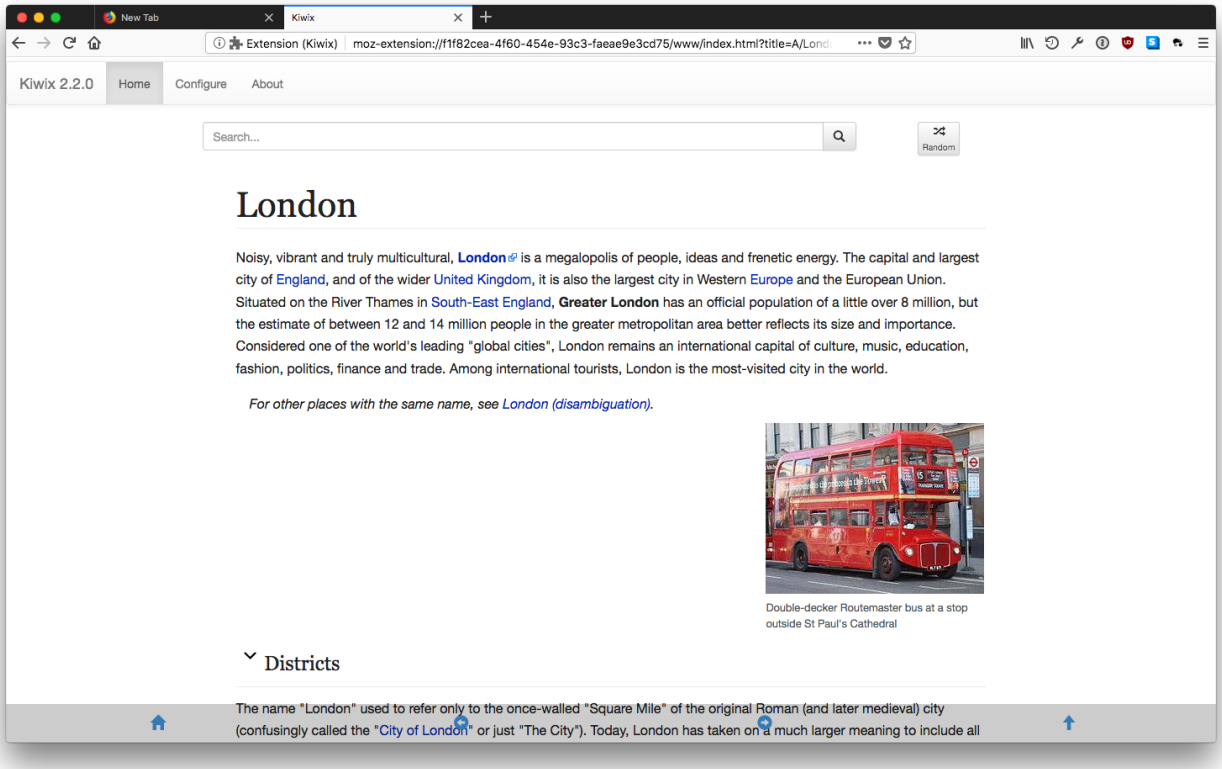

Figure 4.1: Kiwix-JS running in FireFox

be static, ZIM files are not mutable. The ZIM files being static does however bring its own benefits when this immutability mixes with the architecture of the SAFE Network.

4.2 Static versus Dynamic Content

When the idea to “build a Wikipedia on the SAFE Network” was first conceived, we were very well aware of the fact that it might just forever be a “tech demo”. Getting enough users to start contributing content, and building an environment where strict moderation could occur, would have been a fools-errand given the time permissible for this project. It just wouldn’t have been possible to build a full wiki system and do it justice.

It is with that realisation that lead to the discovery of Kiwix. Instead of trying to build a wiki system on the SAFE Network and convincing users to start contributing, it would be possible to bring Wikipedia (and other sites) to the SAFE Network. The content would not be editable by users but it would be there for consumption. An important thing about this approach is that by the end of the year there would be a way to store a browsable copy of Wikipedia on the SAFE Network. In its entirety. Not just a simple throwaway “tech demo” but a tool that people might actually be able to use.

Websites like Wikipedia only work because of their user base. When a user edits an article this change is logged and anyone can review the changes made. As there are lots of users moderating Wikipedia, anything that is grievously wrong is likely to be flagged and addressed quickly. If someone is acting as a “bad-party” and editing pages wrongfully they can be blocked based on IP address. A simple example is a school, it does not need explaining that school children can be rather silly sometimes. This behaviour can result in the vandalism of some Wikipedia pages. In such cases Wikipedia has the moderation tools to block the IP address that belongs

to the school (from making edits) and prevent any further vandalism. On the SAFE Network, this approach is impossible. A user could simply create another account and vandalise an open wiki all they want. It is for reasons such as this that building a dynamic wiki (with adequate moderation techniques/tools) would have been very difficult. A static mirror of Wikipedia was however very achievable.

A static version of Wikipedia might at first seam quite rigid, but in the context of the SAFE Network it makes sense. An organisation like Wikimedia, or a trusted third-party, can upload ZIM files to the network with the assurance that users will know it came from them. It will then exist on the network as an un-censorable mirror (or archive) of whatever source the ZIM file came from. Everyone that has access to the SAFE Network can browse it, as data is immutable that file can never be deleted. Ensuring that the ZIM file is available for the entire life of the SAFE Network.

4.3 Electron

Electron[1] allows you to "Build cross platform desktop apps with JavaScript, HTML, and CSS". Being able to produce an application that was cross platform was very important. The SAFE Network is not platform specific so SAFE Wiki should not have been either. As Kiwix JS is built upon web-technologies, Electron seamed like the obvious answer as to how to pull Kiwix JS outside of the web browser. Electron combines *Node.js* and *Chromium* into a single environment that can be deployed to the three main platforms: Windows, Linux and macOS. As there exists a *Node.js API* for the SAFE Network it meant that a single application could be built. An application that could handle both the publishing of ZIM files and facilitate the browsing of them. The decision not to use the *DOM API* was because of file uploading. To facilitate the upload of large files, (The ZIM for Wikipedia with images is $>$70GB) SAFE Wiki needed to be a desktop application.

Electron and *Node.js* were unfamiliar technologies when development started. Making Kiwix JS run as an Electron application was hence quite a challenge. After a few months of work Kiwix JS was running in a desktop application. Indeed the creators of Kiwix had a similar idea of bundling Kiwix JS into an Electron application some time ago. They were pleased when contacted about the early success of this project. What was now SAFE Wiki (which can be seen in Figure 4.2) could browse ZIM files from local storage and maintained all the functionality of Kiwix JS.

4.4 Developing with the SAFE Network

The SAFE Network was exceedingly difficult to work with, this was down to the lack of documentation and developer resources. Being only at "Alpha 2" they still have a long way to go before a true "1.0" release of the product. This development roadmap indicates that they are holding off on writing proper documentation until closer to the full release. The only saving grace in this matter was the Developer Forum[2]. The developer forum is a very lively place with constant chatter, everyone pitches in and shares ideas. All the information needed to build SAFE Wiki was contained within the forums. This was not an optimal way to find the knowledge needed, it made work very slow and much harder than it needed to be. The lack of documentation and *cannon* knowledge on certain topics resulted in the creation of several forum posts to help support this project. Users could not have been more helpful and most queries posted were resolved within a matter of days.

Development first starts with how to orchestrate the connection to the network. During development a fork of the Beaker Browser project called SAFE Browser[3] was used. SAFE Browser takes the form of a web browser.

[1] https://electronjs.org

[2] https://forum.safedev.org/

[3] https://github.com/maidsafe/safe_browser

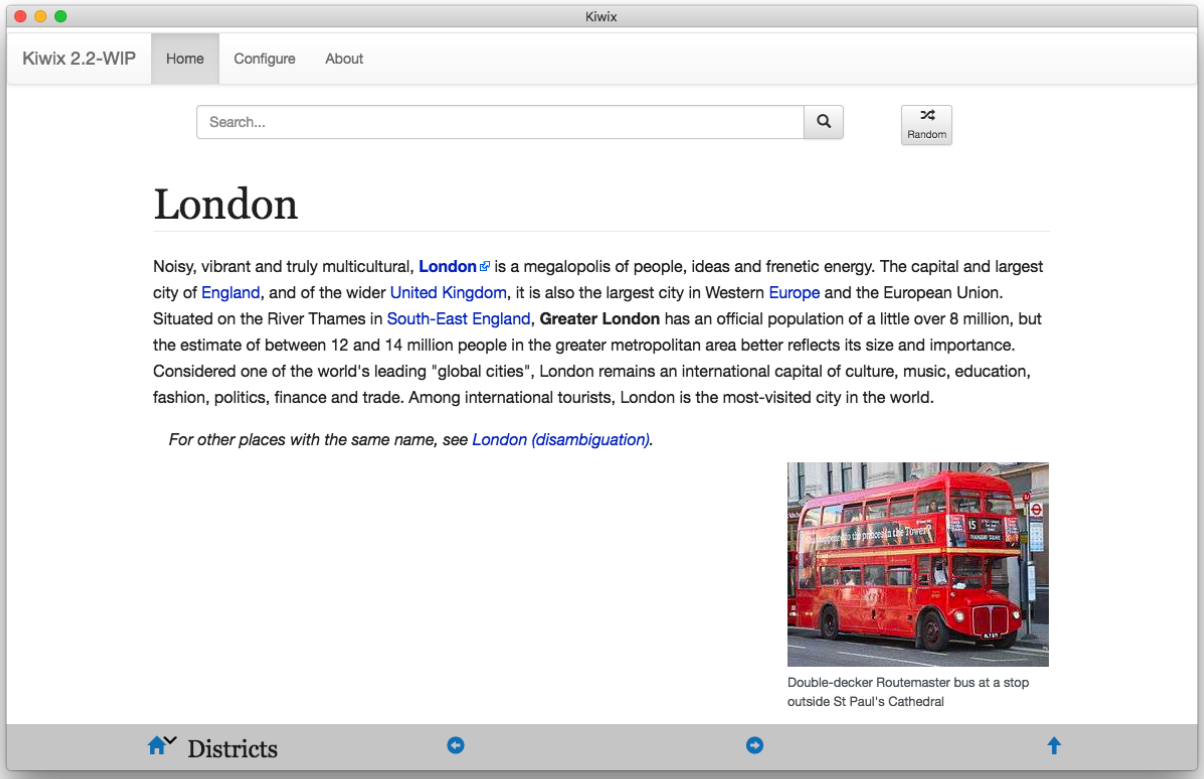

Figure 4.2: Kiwix JS as an Electron App

The browser facilitates authentication and the browsing of any websites hosted on the SAFE Network. Standalone applications, like SAFE Wiki, can use IPC to communicate with SAFE Browser in order to gain authentication. Once authenticated, communication directly with the SAFE Network is possible. Thus SAFE Browser doesn't need to be used as a "middle-man" for network connectivity. Currently Maidsafe are working on the successor to SAFE Browser called Peruse[4].

There are currently three ways to run an application against the SAFE Network.

- Alpha 2 Network: This public network is hosted and ran by Maidsafe themselves. It is the "official" network for early adopters to host websites and run applications against. As it is a public network, it is not optimal for developmental work.
- Mock Routing: Mock Routing is a system that is included in both SAFE Browser and Peruse. What it does is spoof the underlying network to the client through the use of a local database. This means that the client thinks it is talking to a real network while in actuality it is talking to a database. This is a very reliable way of developing locally, although it doesn't give the full experience of how an application/website would work with a "real" SAFE Network. What this method does offer is simplicity. A binary of SAFE Browser can be downloaded that has mock routing already switched on, meaning a single download is all that is required to begin development.
- Local Network: Sadly it was very long into development before it was discovered just how easy it was to run a "real" SAFE Network locally. The process isn't as simple as downloading the binaries and clicking

[4]https://github.com/joshuef/peruse

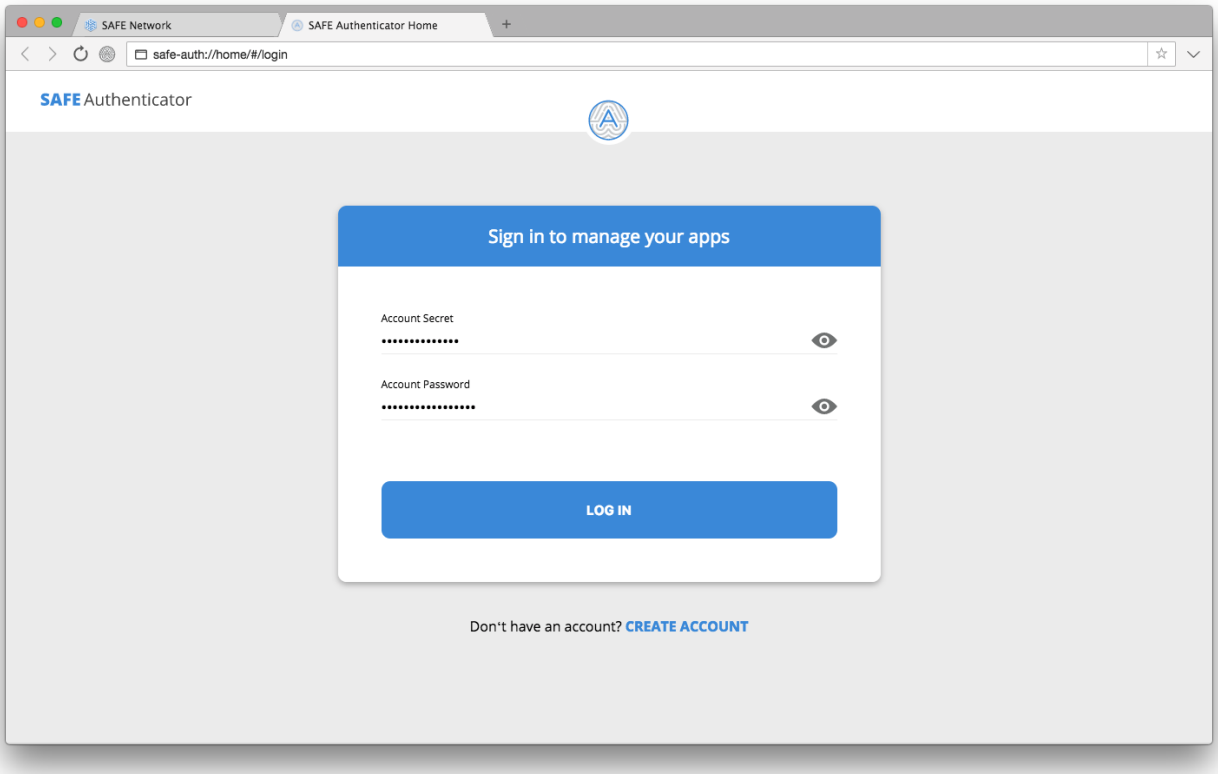

Figure 4.3: Login screen of the SAFE Browser

run, but it is not difficult. One has to download and compile the "safe_vault"[5] from Maidsafe's GitHub. This is a *vault* that makes up the nodes of the SAFE Network. Once configured, several *vaults* can be started and they will automatically connect to one another. Once the "min_section_size" number of *vaults* has been reached, the network is ready for use. The "min_section_size" setting is used to configure the minimum number of *vaults* required to form one complete *section* (default is eight). It is possible to set this number lower (e.g. two), which makes running the *vaults* on one machine much easier.

4.4.1 Web Hosting Manager Example Application

Maidsafe themselves provide a number of Electron example applications[6]. Looking through the code and how they worked was very helpful in learning how the *Node.js API* worked. A big challenge for this project was just trying to figure out how SAFE applications should be designed, how they should authenticate themselves with the network and such. Design patterns for how to do a lot of these things will be established and grow naturally as more developers start working with the SAFE Network. For the purposes of this project, the style of how the "Web Hosting Manager"[7] example app worked was very appealing. Web Hosting Manager is an application that can be used to upload websites to the SAFE Network. As such, it uses almost all of the API for numerous purposes which was brilliant for learning. Looking through the code it became apparent that there was a lot of repeating code across the example applications. It might be best described as "boilerplate" code. This led to the decision to simply "fork" the internal workings of Web Hosting Manager into SAFE Wiki instead of repeating the same code as Maidsafe themselves had done. SAFE Wiki's interaction with the SAFE Network is simpler

[5] https://github.com/maidsafe/safe_vault
[6] https://github.com/maidsafe/safe_examples
[7] https://github.com/maidsafe/safe_examples/tree/master/web_hosting_manager

so only the core functionality was taken. Most notably was the code for reading local files to then upload to the network. As mentioned previously there really are no guidelines on how applications should be built, so by forking this code it meant that development could follow the style that Maidsafe themselves had intended. Proper attribution has been added to any and all source files that are not of my own creation, this includes files from Kiwix JS. Most files have seen significant changes to them as development progressed, the complete history of the changes can be seen on the SAFE Wiki GitHub page[8].

4.5 Authentication

For an application to have connectivity with the SAFE Network it has to be authenticated. Regardless of whether it is a website or a standalone application. Communication from SAFE Wiki to the SAFE Browser for authentication was one of the most difficult parts of this project. Getting SAFE Wiki itself to run on all supported platforms was trivial, it just worked straight away. Getting SAFE Wiki to communicate with the SAFE Browser on all platforms was extremely difficult. Luckily, a community member had published an example SAFE Network Electron app called "safe app base"[17]. This is a modified version of the application from the "Electron Quick Start" guide[18], which made understanding how it worked very easy. The app itself is very basic, all it does is ask the SAFE Browser for authentication then creates a new Mutable Data structure and prints it to console. Through trying the application it was discovered that it did not work on macOS. What would happen is the SAFE Browser would successfully authenticate the application but the application itself would never receive the response to say it had been. It was deduced that the issue was regarding how URI Schemes are registered across the system. The mechanics of how this works differs across the platforms so what works on one operating system may not work on another. Differences on how the application is ran also has an impact. What may work when running the application from terminal (through the "electron" command) might not work when the application has been packaged as a binary. Indeed there are even differences depending on which Electron package you use to bundle/package the application.

This issue with authentication was a big setback because if a simple example application didn't work then it would prove difficult to implement SAFE Wiki successfully. To help solve this issue a forum post[19] was created to discuss it with the community. The best approach would be to fix the example application before attempting the implementation in SAFE Wiki. Conversation with numerous people lead to a fix being created[20], which got merged into the "safe app base" GitHub. The working "safe app base" can be seen in Figure 4.4. Problems with URI Schemes cropped up later on in development too, resulting in another forum post[21]. This time the issue was with support on Ubuntu. Thanks to the help of the creator of "safe app base", the issue was solved quickly.

4.6 NFS Emulation

To support the storage of ZIM files on the SAFE Network, SAFE Wiki makes use of the NFS emulation support that the *Node.js API* has. This "emulation" is just a wrapper around Immutable and Mutable Data structures that makes working with "files" much easier. In SAFE Wiki nomenclature there is the concept of a *ZIM folder*. This "folder" is really a Mutable Data structure that is emulated as a folder through NFS. Within this folder are placed the ZIM files that a given user uploads.

[8] https://github.com/DaBrown95/safe-wiki

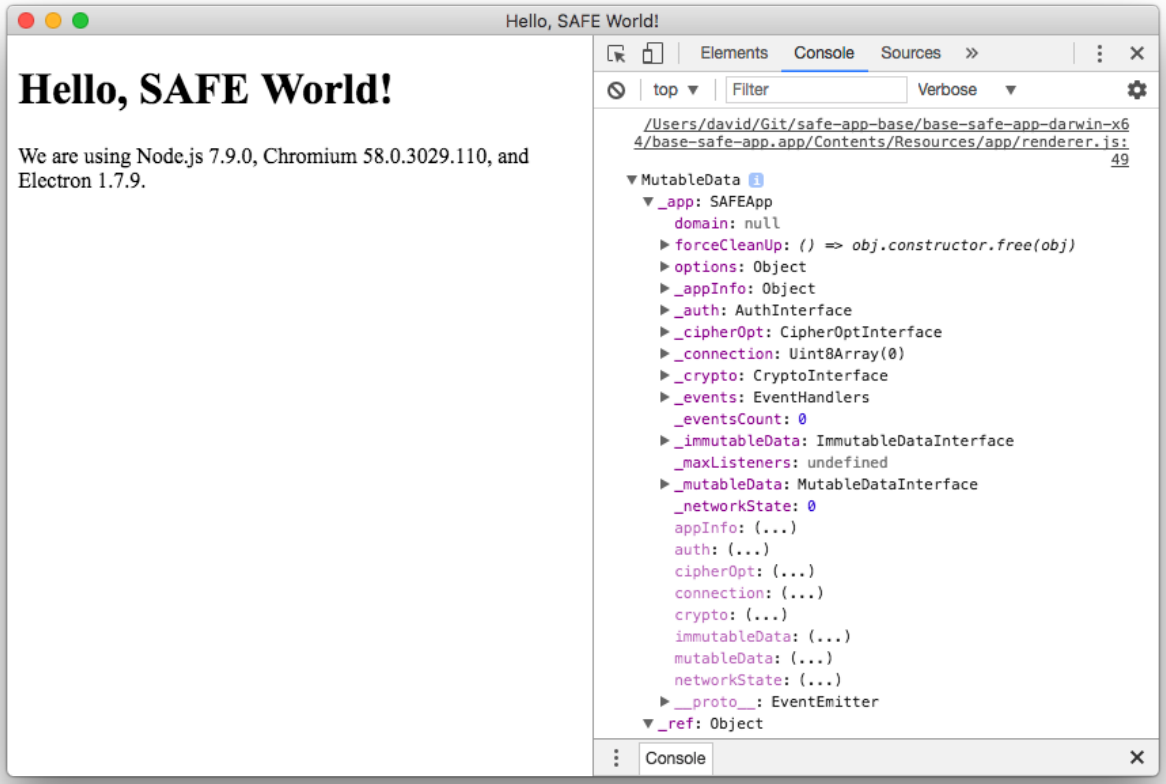

Figure 4.4: SAFE App Base with newly created Mutable Data structure

4.6.1 ZIM Folder

Every account on the SAFE Network has a number of Mutable Data structures by default that are called *containers*. These *containers* are similar to a "home folder" on a traditional OS in that they give applications structure (guidance) on where to store things. Such containers include: _public, _downloads, _music, _pictures, _videos, etc. The ZIM folder that SAFE Wiki uses is stored within the _public container because data stored within there can be "un-encrypted" or "public" data. Within the _public container is placed a key-value pairing where the key is "zim" and the value is the address of the Mutable Data structure that is the ZIM folder. When a user creates a ZIM folder they must specify a name. That name is then hashed to give a unique 256-Bit address which is where the ZIM folder is then stored. Thus through the name of a ZIM folder another user can locate the ZIM files that are stored within it.

4.6.2 ZIM Files

Once a user has created a ZIM folder, they are then able to upload ZIM files to the network. When a user uploads their ZIM file they give it a name, this name is important. This name is the "file name" of the file. Meaning that within a given ZIM folder, the ZIM file is stored against the name the user specifies.

To access a ZIM file, all a user has to provide SAFE Wiki with is the name of the ZIM folder and the name of a ZIM file within that folder. This approach means it is easy to share access to ZIM files, as names can be

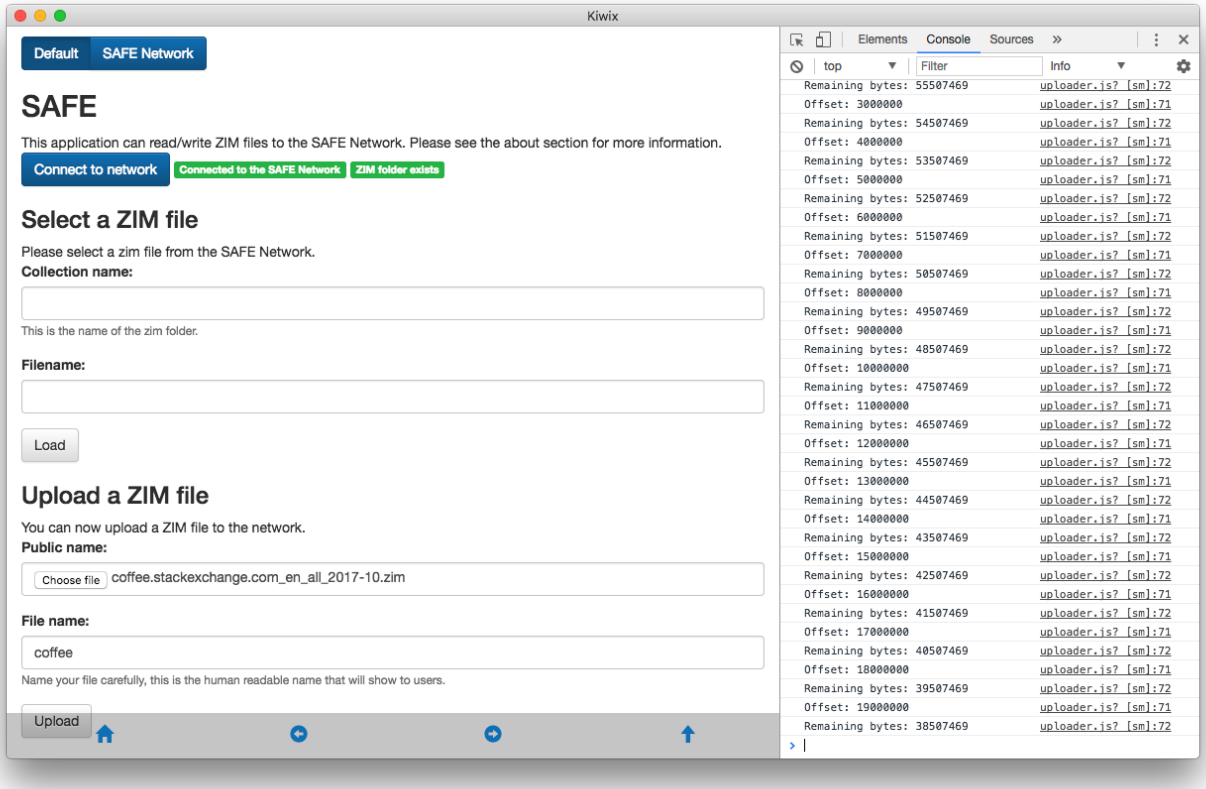

Figure 4.5: Uploading the Coffee StackOverflow ZIM file to the SAFE Network

human readable they are as easy to share as website URLs. The resolution of the 256-Bit address of the ZIM folder is through hashing. As the ZIM folder was stored at the address corresponding to the name specified by its "owner", the address is then derivable by anyone else that knows that name. The way this is envisioned to be used is that the name of the ZIM folder can correspond to the originator of the content and then the filenames can follow on logically from that. For example, "Wikimedia" could be the name of the ZIM folder and then "Wikipedia" could be the name of a ZIM file within that folder. As things are organised like this it then becomes logical to derive the location of other ZIM files. A user can deduce that to get to "WikiVoyage" is as simple as "Wikimedia" and "WikiVoyage".

4.7 Reading ZIM Files

An important feature facilitated by *data maps*(Section 3.2.1) is that it is possible to read arbitrary bytes from files without having to download all data chunks. For ZIM files this is important, it is illogical for SAFE Wiki to have to download the entire Wikipedia so that a user can browse a single article.

When a user chooses to read ZIM files from the network, the read requests that are usually directed to local storage are instead diverted to the SAFE Network. This approach, being modular in design, means that the original functionality of Kiwix JS is maintained. A user can still choose to read ZIM files from local storage if they wish. The code to perform reading from the SAFE Network is shown in Figure 4.7. The `zimFolder` that is a Mutable Data structure is first emulated as a folder using NFS, then the target `file` is fetched by using the `filename` specified by the user. To then read the required bytes is simple. In Figure 4.6 the discrete reads from the network can be seen in the console.

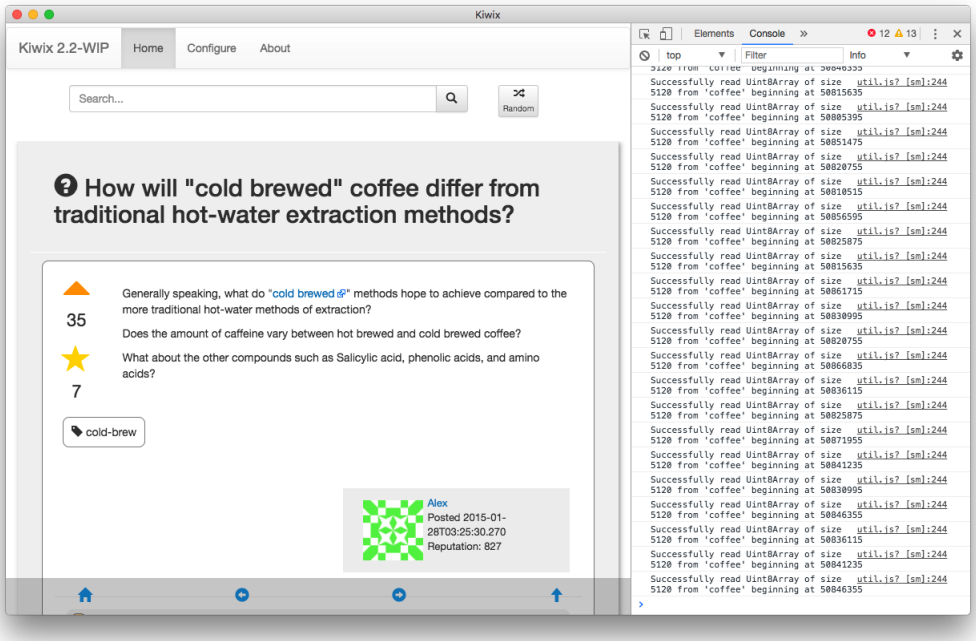

Figure 4.6: Browsing a page from the StackOverflow Coffee ZIM file on the SAFE Network

```
readZim (zimFolder, filename, begin, size) {
  return new Promise(async (resolve, reject) => {
    try {
      const nfs = zimFolder.emulateAs('NFS')
      let file = await nfs.fetch(filename)
      file = await nfs.open(file, CONSTANTS.FILE_OPEN_MODE.OPEN_MODE_READ)
      let data = await file.read(begin, size)
      file.close()
      resolve(data)
    } catch (error) {
      reject(error)
    }
  })
}
```

Figure 4.7: Code to read a ZIM file from the SAFE Network

Chapter 5

Evaluation

5.1 Experimentation with Local Networks

During software development it is important to be able to adequately test and deploy software internally on a local network. Although *Mock Routing* is an option, it is only possible to gain a true perspective on how an application functions when it is ran against a real SAFE Network. A network that is both stable and functioning as the global and public SAFE Network would.

5.1.1 Method

After SAFE Wiki was fully working, some un-formal experimentation on how well a local SAFE Network can handle the uploading of ZIM files was performed. To do this, a number of *vaults* were setup on different computers on a local network. The computers used included a desktop, laptop and a Raspberry Pi[1].

It was very easy to setup the local network, *vaults* could successfully connect to each other and form *sections* without any further configuration. The network was indeed working *autonomously*. Regrettably however, the Raspberry Pi did not work well and consequently it was no longer used. It is difficult to say whether this was down to network connectivity or limited hardware resources. What can be deduced however is that a *vault* running on a machine like a Raspberry Pi 3 is not viable at this time.

ZIM files were repeatedly stored on the network and then browsed from different computers, including computers that didn't have a *vault* running on them. The number of *vaults* was increased to the point that network *splits* started to happen. During testing, the number of *vaults* needed before a *split* happened did vary. There was however a lower bound on the number of *vaults* required. The SAFE Network has a `split_buffer` that is used in combination with the `min_section_size` to help decide whether a *section split* should happen, currently set to `3`. The equation can be seen in Figure 5.1. For a *split* to happen, this condition must be met for every *section*.

$$number\ of\ vaults = min\_section\_size + split\_buffer \tag{5.1}$$

The `min_section_size` is a configuration option that dictates the minimum number of *vaults* required to form one complete *section*. By default this value is set to `8`, however using this number with a local network

[1] https://www.raspberrypi.org/

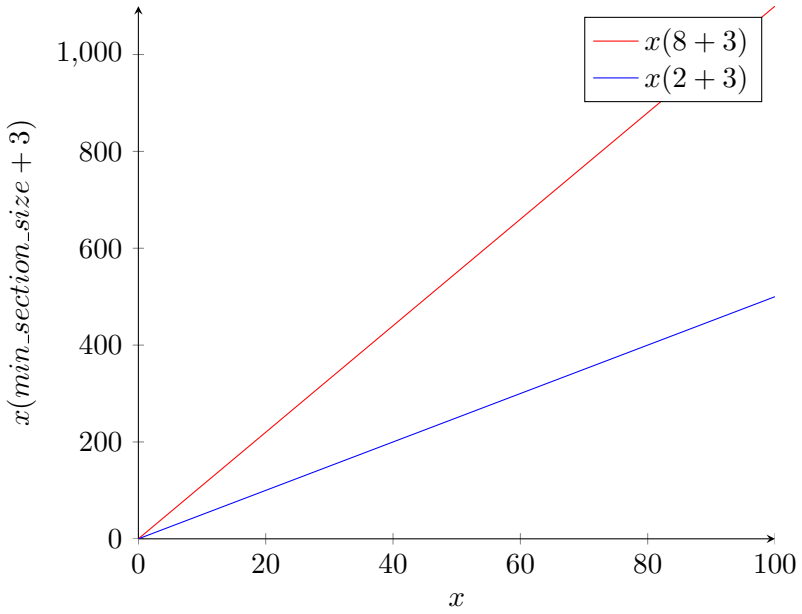

Figure 5.1: Compares how `min_section_size` impacts the number of *vaults* needed to form a given number of *sections*.

is problematic. Using the equation in Figure 5.1, to have two *sections* on a local network you would need a minimum of `22` *vaults*. To run this number of *vaults* on one machine is difficult (anecdotally the network is extremely slow if you do). Hence to facilitate a greater number of *sections* in a local network, it is advisable to set the `min_section_size` as small as possible.

5.1.2 Results

What was notable about this experimentation is that it is extremely easy to run a local SAFE Network. For developers and researches this facility is of the utmost importance. What proved to be troublesome though was using a small `min_section_size`. It was found that when using a number such as `2`, the network was unstable. If *vaults* left or joined after data had begun to be stored, the network would cease to function and would simply lock-up and not respond to clients. The explanation for this is that the network isn't designed to be fully functional with such a small `min_section_size`. This means that in practice it is difficult for a single person to have enough computing power to run a local network with a significant number of *sections*. To have two *sections* with a `min_section_size` of `2` requires a minimum of `10` *vaults*. Figure 5.1 shows the significance of the difference between a `min_section_size` of `2` and `8`.

To be able to run enough *vaults* on one computer to adequately test an application against a stable and local SAFE Network is difficult. The ability of the network to be more stable at lower `min_section_size` values, or to have a formally specified and endorsed minimum value that can be used, would be very beneficial.

5.2 Discussion

5.2.1 Ownership of Data

As discussed in Section 2.3, ownership of data is an important aspect of the SAFE Network. "Users own their own data" is a feature that the network strives to ensure. It is through exploiting these features that developers

can create interesting and new applications.

When developing for the SAFE Network it is very important to understand that Immutable Data is truly immutable. Once someone has access to a *data map*, the data it leads to is then available to them forever. There are no permissions to the data that can be revoked. This means that once a key or access to a *data map* is leaked or discovered, that data is available publicly forever. This interacts in odd ways with data de-duplication. Imagine a scenario where a film studio stores an archive of their movies on the SAFE Network. Each film would have its own encrypted *data map* that only the studio has the key for. If somehow this file was leaked, a user could upload it to the SAFE Network with an un-encrypted *data map*. Owing to *self-encryption*, the new *data map* would point to the exact same chunks that were stored on the network by the studio. Meaning there would now be two *data maps* that point to the exact same data, an encrypted one and a public one. At this point the studio would have no means of restricting access to the data, even though their encrypted *data map* is still secure.

Thus when users are using the SAFE Network they have to be extremely careful when managing the keys to their data. Once leaked, there is no method of revoking access. Thankfully most of the key management to such data is handled by storing them inside a users account on the network. If the account credentials are compromised however, there is no way to "change a password" or recover an account. The data that the account "owns" is leaked forever. Thus the suggestion to further encrypt the data on the client before uploading it to the network, and storing that key through separate means, is highly suggested.

Immutability of Data

As mentioned in Section 3.2.3, data on the SAFE Network is immutable. Once written to the network a chunk of data can never be removed. It is the access to data through *data maps* and encryption keys that orchestrates the access to it. What this means in practical situations is that once someone has access to data, they have access to it forever. There is no way to revoke that right. For SAFE Wiki, this is brilliant. Once a ZIM file is uploaded then a user can access that information forever.

Sometimes users do want want to securely delete data, so to not have this ability is a problem. Although not implemented yet, there is talk and an RFC[22] to bring deletable Immutable Data to the network. However to allow "Immutable Data" to be deleted is an interesting concept, one can't really call it "Immutable" if it can indeed be deleted.

5.2.2 Cost Benefit

A huge benefit to anyone wanting to build a website or application is how much it costs to store data on the SAFE Network. Traditionally on the internet, a user would need to continuously pay for the upkeep of a server. Depending on how much traffic the website gets, the costs associated with running a server can increase rapidly. On the SAFE Network, a user only has to pay *Safecoin* when writing to the network. That means that once published, a website or application will be available forever and not incur any additional costs.

For SAFE Wiki, this means that once a user pays to upload something like Wikipedia ("only" 75GB in size) it is then available to everyone for free and forever. It doesn't cost anymore money than it cost to store the file originally. This means that it is possible to publish a resource like Wikipedia and incur no running costs. As the ZIM file is stored on the network as immutable data, it is available for consumption by all and forever.

5.2.3 Alternative Business Models

Like any new technology, the SAFE Network opens up many opportunities that didn't exist before. In SAFE Network nomenclature, *vaults farm* data. The safe and reliable storage (*farming*) of data is rewarded with *Safecoin*.

Exploiting Consumer Resources

One could envision an application that instead of charging users for access, allows them to become a *vault* that generates *Safecoin*. This *Safecoin* could then be sent back to the creators of the program and hence financially compensates them for the usage of their application. This model could be used to better make use of a consumers resources. When a user sits and watches Netflix on an entertainment system, there is very little strain on the resources of that device. Potential financial models can try to *exploit* this untapped power to the benefit of both the user and the provider of the application. Encouraging the creation of more *vaults* not only increases the utility of the SAFE Network but provides users with an entirely new way to pay for content. Offering the resources they have in exchange for access to services.

Micro-Payments

As *Safecoin* is so tightly integrated with the network it can be used in interesting ways. A possibility talked about in the *Safecoin* white-paper[16] is the facilitation of micro-payments. Traditional currencies and payment processing methods are too expensive to be used for quick and small payments. This issue does not exist for *Safecoin* which means it can be used to facilitate small payments. As the white-paper suggests, this could be used "to pay for films on a cost per frame basis, with the user only paying for what they watch". By using this system, artists and creators could cut out the "middle-man" for users consuming their content. If the financial benefits are significant then they may be drawn to the SAFE Network over alternative platforms like Spotify. Current platforms usually take a significant cut of profits so it may indeed result in higher revenue for them. How profitable a system like this would be remains to be seen. When Maidsafe gives a formal outline of how micro-payments will be implemented, it will then be possible to examine if they will benefit creators.

5.2.4 Privacy and Anonymity

Anonymity and Privacy are not mutually exclusive. Anonymity means *"Nameless; of unknown name; also, of unknown or unavowed authorship"*[23]. True anonymity is very important for many people around the world, especially when it comes to digital communications. The identities of individuals is what matters here. Being able to convey a message and be "nameless". Privacy means *"The state of being in retirement from the company or observation of others; seclusion"*[24]. Most would agree that people should be able to send a message/letter/email to someone and have that communication be private.

The SAFE Network provides guarantees of privacy. A user can upload pictures and documents with the assurance that only they can access them, it is private data. A user could also send a message to someone by inserting an encrypted entry in their "inbox", this is private communication. When a client requests a chunk of data from a vault, that vault knows something relating to the *account* that is requesting the data. However difficult it may be to tie that information to an individual, that information exists for a period of time and hence full anonymity is not insured.

The access to *data maps* is what gives users privacy. If a user doesn't have access to a *data map* then they cannot know what the data it leads to is. Once data is stored on the network, it is anonymous. Thus for the above

reasons the SAFE Network allows pseudo-anonymised interactions. In practically users can be relatively assured that their interaction is anonymous, they must however know that it is non ensured fully.

In some parts of the world people do not have the freedom to access information freely. A modern day example of this is that the Chinese government are still censoring access to information about Tiananmen Square[25]. The SAFE Network facilitates the uncensored access to information, which as discussed at the beginning of this report is crucial to developing societies. If a user can connect to the SAFE Network successfully they can interact with it fully, there is no avenue to censor or curate their interactions.

5.2.5 Building websites on the SAFE Network

The SAFE Network can do simple websites really well. The cost benefit of not having "running costs" is a big benefit to people. Some websites are not updated frequently and thus being stored on the SAFE Network would possibly save users years of running costs. It is when websites become more complex, that the SAFE Network becomes a very difficult product to suggest. A simple example is e-commerce. As there is no processing available on the SAFE Network a server to handle payment processing would still be required. That dependency almost invalidates the whole reason for using the SAFE Network. This could be answered by offering the use of cryptocurrency as an option to pay for goods, this approach wouldn't need a traditional server to handle payment processing. For wide-scale adoption, only accepting cryptocurrency is not a sensible option at this time.

5.2.6 Storing and archiving data

The storage of data is where the SAFE Network excels. Storing things like backups and archives makes perfect sense, once stored it cannot be removed. Thus the SAFE Network is a very attractive option when it comes to the long term and secure storage of data, this applies to organisations and to individuals. The reduced cost is very appealing too, a user could backup large amounts of data and have access to it indefinitely without having to pay a yearly or monthly fee. Organisations could use the SAFE Network to perform backups of critical systems and for other purposes. Long term storage with a one time fee is a very attractive prospect.

Companies cannot own third party data

If a user asked a company to delete their personal data, the company simply wouldn't be able to if that data was stored on the SAFE Network. All they could do is the equivalent of throwing away the key to a filing cabinet. Amongst many other reasons, this is why companies cannot own "third party" data on the SAFE Network. Instead of a user giving companies their data, they give the companies access to their data to facilitate the services they wish to access. This model means that services need to be built around the idea that the source of the data is ultimately in control of that data.

5.3 Future Improvements to SAFE Wiki

Future work for SAFE Wiki could take one of two approaches, development could continue on SAFE Wiki itself or another approach could be taken. An alternative approach to how SAFE Wiki could be further developed is outlined below.

5.3.1 ZIM Uploader

The first step would be to create a new application called "ZIM Uploader". This applications only purpose would be to facilitate the management of ZIM files on the SAFE Network. This means that a user that has no intention of uploading their own ZIM file doesn't need to see this piece of functionality. As this application only serves one purpose, it would be far easier to maintain than SAFE Wiki itself.

5.3.2 Kiwix JS Extension

Kiwix JS is a browser extension, using the browser as its run-time environment. Forking away from this approach perhaps fragments things more than they need to be. Instead of pulling Kiwix JS into a desktop application, through the use of the *DOM API* the functionality of SAFE Wiki could be brought to the extension. With this approach the user would have Kiwix JS installed inside their browser (SAFE Browser or Peruse) and be able to use Kiwix JS as normal. Within a SAFE Network environment, Kiwix JS would then have the ability to read ZIM files from the SAFE Network.

Extending Kiwix JS instead of maintaining a fork brings many benefits. The first and foremost is that there is the possibility of this added functionality being merged into the main branch of Kiwix JS. This would not only increase the awareness of SAFE Wiki but means that improvements (bug fixes etc) can be made to one single repository instead of constantly pulling changes across the two streams.

5.3.3 Website

Through the *DOM API* it would be possible to build a website that would facilitate the reading of ZIM files. This means a user could simply visit the website through a SAFE Network compatible browser and then browse ZIM files hosted on the network. This approach would deliver the internals of Kiwix JS through the browser to run on the client, providing users with an extremely easy method to access the functionality of SAFE Wiki. The drawback with this approach however is that the maintenance of another code base would be required, meaning the benefits described in Section 5.3.2 would not apply.

5.3.4 Suggestion

One can envision that in the future it would be possible for a user to download ZIM files from the SAFE Network for offline consumption. Thus maintaining the ability to browse the files locally, without an internet connection, is a key piece of functionality that shouldn't be lost. Thus the preferred and suggested approach is the one outlined in Section 5.3.2. It is Kiwix JS, a well established project, with the added ability to read files from the SAFE Network.

5.4 Unresolved Questions and Problems with the SAFE Network

5.4.1 Performance

How practical the SAFE Network will be for applications like video streaming and file downloads is unknown at this point. Speaking from experience, dealing with uploading large quantities of data to a local SAFE Network is slower than one might imagine. Although this observation is merely anecdotal, it will be interesting to see what

performance is like when there is a global SAFE Network. How *node ageing* and *churn* effect performance is also an interesting question to ponder. When *vaults* are moving between *sections*, those *vaults* no longer contribute to the resources of a particular *section* and actually consume them instead. Thus how these network events impact on the experience of the end-user is still to be seen.

5.4.2 Safecoin

The fee model and economy of *Safecoin* is not well established. The *Safecoin* white-paper[16] does give some indications as to how things will work but the implementation details are omitted.

A feature that was discussed in the white-paper is that "based on how much the application is used, the network will pay safecoins to the safecoin wallet address of the app creator. This provides a built in revenue stream for app developers, one that is directly proportional to how successful their application is." The details of how this will work is not specified.

In the white-paper it is said that *Safecoin* is awarded "to the successful node as data is retrieved from it (GETS), as opposed to when it is stored (PUTS)". What this means is that if data is never accessed, the *vaults* storing that data will never be rewarded for doing so. This means that the network has to support the data indefinitely and nobody will ever be incentivised for doing so. This means that the amount of *Safecoin* that was originally used to pay for the storage has to be great enough to cover storage and network costs for decades upon decades. A possible answer to this is that *vaults* are paid during *churn* when they send and receive data to one another, this is a mere guess however and is not specified in the white-paper.

5.4.3 Maidsafe has an information problem

The situation regarding documentation makes suggesting working with the SAFE Network extremely difficult. The main problem is the lack of clarity and "cannon" knowledge on different subjects. There is no central location where a developer could lookup a piece of information and get the hard facts as to what they are enquiring about. Information is spread thin and far across the forums and different blog websites. Sometimes it is hard to distinguish between opinions on how things should be and how they actually are. Indeed the wiki for the SAFE Network is incredibly out of date in some areas, some of it is just plain wrong. This is a problem that made assembling information for this dissertation very difficult.

Some very kind community members took it upon themselves to write the *A SAFE Network Primer* document. This document is one of the first unified resources of information that people can use to learn about the SAFE Network. This speaks to the enthusiasm of the community at large who believe in the SAFE Network and its ultimate utility. There is a lot of knowledge out there, it just needs to be brought into one central location that is an approved resource of knowledge. This lack of clarity can be explained, but not excused, by the infancy of the network. It is only in *alpha 2* and not a finished product. Going forward however this is something that needs to be addressed.

5.4.4 How Traditional Internet Businesses would Benefit

Suggesting that the SAFE Network is a viable avenue for existing businesses is difficult. A few basic details mean that their current financial models simply wouldn't work on the SAFE Network. Thus how traditional businesses could benefit from the architecture of the SAFE Network is a very difficult question to answer. Thus the suggestion is that many simply do not benefit. For most, their entire business is based around the collection and curation of user data and thus the SAFE Network is not commercially viable to them. Thus new models on

how to commercialise ideas will have to be thought of. The main barrier to this is the technical challenges in how to do so. The SAFE Network is incredibly rigid when compared to the flexibility of the traditional model of the internet. Thus there has to be the creative exploration of how to facilitate and incentivise existing services to move across to the SAFE Network.

5.4.5 A Full Prototype and Specification Doesn't Exist

There does not exist a fully working prototype of what the "1.0" release of the network will be. There are a lot of details unexplained, especially surrounding *Safecoin*. How the value of a *Safecoin* is calculated and how payments will work in practice is unspecified. This makes it extremely difficult to draw any sort of concrete conclusions as to the effectiveness of the network. At this time it is unknown whether Maidsafe will even be able to address the challenges that they still have to solve.

5.4.6 Future Work

The SAFE Network provides many exciting avenues for academic research. The title of this paper is "Building Applications on the SAFE Network" and that was the main focus of it. With such a broad topic it is difficult explore specific questions in as much depth that is needed. The following are proposals of specific areas that would benefit from explicit and deep academic research.

Safecoin

As discussed in Section 5.4.2, the success of *Safecoin* is crucial to the success of the SAFE Network. A deeper and formal analysis on the economic viability of *Safecoin* is needed to evaluate whether the SAFE Network is financially viable for users.

The Datachain

How Datachain's will ultimately impact the operation of the SAFE Network is still undecided. An interesting academic exercise could be to review the current proposals and to run simulations to give quantitative evidence as to their effectiveness. Maidsafe are currently implementing the current proposals and will be included in the *alpha 3* release of the network. Upon the successful release of *alpha 3*, the deeper analysis of how Datachain's function will be possible. This is all information that would be incredibly useful to Maidsafe.

Competitors

There does exist some competitors to the SAFE Network. An analysis on how the SAFE Network compares to these competitors would be an interesting topic to look at. An ideal candidate for comparison is Filecoin[2]. The incentive layer for Filecoin is far more clearly defined than in the SAFE Network (currently) so an evaluation between the two could be beneficial academically and to Maidsafe themselves.

[2] https://filecoin.io/

Chapter 6

Conclusion

Evaluating the SAFE Network in its current state does the project an injustice, it is not finished yet. Uncertainties in implementation details, especially *Safecoin*, brings into question how successful the project will be in the long run. Proper incentives for *vault* owners is crucial in the success of the network. The next big step for Maidsafe is the implementation of Datachain's (Section 3.9) and it will be very interesting to see how successful they are.

The SAFE Network ultimately provides an interesting new way to develop applications and services. Developers will have to re-think how they build applications to exploit the characteristics of the SAFE Network to their benefit. Although SAFE Wiki demonstrates that the secure storage and retrieval of data is already possible, the feasibility of running large websites and services that require extremely fast processing needs to be researched fully. The performance of how a world wide SAFE Network with dynamic growth and decay will only be known once it fully launches.

The passion exerted by the team at Maidsafe can only inspire confidence in that they will try their very hardest to achieve all of their goals. This is in-spite of all of the challenges that they still have to solve. The success of the SAFE Network is not only dictated by its success, but by the success of any projects that are built because of it. Whether this be a SAFE Network "2.0" or an entirely new project that Maidsafe has laid the foundations for.

6.1 Personal Reflection

This project was one of the most difficult I have ever undertaken. The lack of resources available made development extremely difficult and tiring at times. It took a good few months before I was even comfortable with the basic concepts of the SAFE Network. Indeed I discovered some of my knowledge was completely flawed while writing this dissertation.

I am very proud of the application that was developed, SAFE Wiki is a good example of what the SAFE Network can do. If and when the SAFE Network fully launches, I hope that some may find SAFE Wiki of actual use.

The biggest thing I learned from this project is that community in software development is important. Without the help of the community around the SAFE Network, this project simply wouldn't have been possible. When I first announced my project on the forums[1], it was exciting to hear the positive response from the community. A few even tried SAFE Wiki out for themselves! It is through this community involvement that I have been invited to speak at Maidsafe's first dev-conference, an extremely exciting opportunity to develop myself professionally.

[1] https://safenetforum.org/t/safe-wiki-wikipedia-on-the-safe-network/21493

6.2 Acknowledgements

This project would not have been possible without the support of my supervisor Inah Omoronyia, a huge thank you to him.

Maidsafe were gracious enough to invite us to their headquarters in Ayr, the hour we spent together was very illuminating.

To the community members that answered the many queries and questions that I had, I owe a great deal of thanks. Without their help this project would not have been possible.

The creators of Kiwix are truly inspiring people. The work that they do in delivering free educational content to the people who need it most is very noble. I hope to be able to contribute something back to their projects in the future.

Bibliography

[1] Jan. 2018. [Online]. Available: https://safenetwork.org/.

[2] N. Lambert, *Autonomous data networks and why the world needs them*, Oct. 2017. [Online]. Available: https://blog.maidsafe.net/2017/10/07/autonomous-data-networks-and-why-the-world-needs-them/ (visited on 01/16/2018).

[3] Jan. 2018. [Online]. Available: http://www.openzim.org/wiki/OpenZIM.

[4] United States Holocaust Memorial Museum. (2018). Book burning, [Online]. Available: https://www.ushmm.org/wlc/en/article.php?ModuleId=10005852.

[5] P. J. Lor and J. J. Britz, "Is a knowledge society possible without freedom of access to information?" *Journal of Information Science*, vol. 33, no. 4, pp. 387–397, 2007. DOI: 10.1177/0165551506075327. eprint: https://doi.org/10.1177/0165551506075327. [Online]. Available: https://doi.org/10.1177/0165551506075327.

[6] J. Bindé, "Towards knowledge societies: Unesco world report," 2005.

[7] U. G. Assembly, "Universal declaration of human rights," *UN General Assembly*, 1948.

[8] B. Cohen, *The bittorrent protocol specification*, 2008.

[9] Palo Alto Networks, *Application usage & threat report*. [Online]. Available: http://researchcenter.paloaltonetworks.com/app-usage-risk-report-visualization/# (visited on 03/18/2018).

[10] BBC. (Apr. 2017). Turkish authorities block wikipedia without giving reason, [Online]. Available: http://www.bbc.com/news/world-europe-39754909 (visited on 01/16/2018).

[11] D. Irvine, "Self encrypting data," 2010.

[12] M. J. Dworkin, "Sha-3 standard: Permutation-based hash and extendable-output functions," Tech. Rep., 2015.

[13] M. Amy, O. Di Matteo, V. Gheorghiu, M. Mosca, A. Parent, and J. Schanck, "Estimating the cost of generic quantum pre-image attacks on sha-2 and sha-3," in *International Conference on Selected Areas in Cryptography*, Springer, 2016, pp. 317–337.

[14] [Online]. Available: https://www.keylength.com/en/4/.

[15] R. R. Schaller, "Moore's law: Past, present and future," *IEEE spectrum*, vol. 34, no. 6, pp. 52–59, 1997.

[16] N. Lambert, Q. Ma, and D. Irvine, "Safecoin: The decentralised network token," Maidsafe, Tech. Rep, Tech. Rep., 2015.

[17] Mar. 2018. [Online]. Available: https://github.com/hunterlester/safe-app-base.

[18] Mar. 2018. [Online]. Available: https://electronjs.org/docs/tutorial/quick-start.

[19] Dec. 2017. [Online]. Available: https://forum.safedev.org/t/sample-safe-electron-app-doesnt-work-on-macos/1213?source_topic_id=1438.

[20] Dec. 2017. [Online]. Available: https://github.com/hunterlester/safe-app-base/commit/66563fe8d8a5438714fc224f168252615b5a479f.

[21] Feb. 2018. [Online]. Available: https://forum.safedev.org/t/uri-scheme-registration-on-ubuntu-linux/1438.

[22] [Online]. Available: https://github.com/ustulation/rfcs/blob/deletable-immutable-data/text/0000-deletable-immutable-data/0000-deletable-immutable-data.md.

[23] [Online]. Available: http://www.websters1913.com/words/Anonymous.

[24] [Online]. Available: http://www.websters1913.com/words/Privacy.

[25] J. Kaiman, *Tiananmen square online searches censored by chinese authorities*, 2013 6. [Online]. Available: https://www.theguardian.com/world/2013/jun/04/tiananmen-square-online-search-censored (visited on 03/18/2018).

[26] *A safe network primer*, 2018. [Online]. Available: https://maidsafe.net/docs/Safe%20Network%20Primer.pdf.