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 20, 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” [3]. 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 (which again works autonomously) stores data in a decentralised manner. Owners of *vaults* are compensated for their computers resources, which encourages more *vaults* to join the network. Increasing its reliability, storage capacity and performance. A global network that facilitates the decentralised, highly redundant and secure storage of data creates exciting new opportunities for developers.

1.2 Aims

In this project I aim to not only explore the technical benefits the SAFE Network provides, but also consider the societal impact such a network could have. For the project I have developed an application that I have called SAFE Wiki. SAFE Wiki aims to provide *permissionless* and decentralised access to Wikipedia, facilitated by storing an archive of it on the SAFE Network. ZIM[4] 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 I hope to explore 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.

1.3 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. Through this report I hope to not only convey my experience working with the SAFE Network, but also outline any flaws and issues I can see with both adoption and practical usage. This is in combination with exploring the new opportunities the network provides for softare development.

1.3.2 Cultural Impact

In my opinion, the right to liberty and the unobstructed access to information is the most important right we 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. To control people. The most prominent example of this was the Nazi Book Burning Campaign [5]. The goal of this 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[6]. A ‘Knowledge Society’ is defined by Bindé in “Towards knowledge societies: UNESCO world report”[7] 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 why I find the prospect of bringing Wikipedia to the SAFE Network to be such an exciting concept. 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[8]: *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

Decentralisation of data is the core benefit of the SAFE Network. As with many things in life, once someone has ownership or control of something they can either use that position of power for good purposes or for less desirable ones. The internet as it exists today is very fragile in this regard. When you upload a file to Dropbox[9] or OneDrive[10] 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 please, acting within the bounds of overcomplicated privacy polices to manage user data. Not only does this incur the obvious 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 their data is safe. On a much larger scale companies like Amazon provide AWS[11], an enterprise grade cloud-computing platform. If AWS were to fail, or be targeted, many of the worlds biggest websites would cease to function. This is because of the centralisation of resources. It is not necessarily an easy target, but it is a single identifiable piece of the equation that if removed, causes the whole thing to collapse.

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 between nodes. Vaults on the network do not inherently trust other vaults. Every action on the network must be 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 to store data.

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 you want to upload data that trust in the entity becomes a big consideration. The Client-Server architecture forces centralised governance, there is very much the idea of a central power 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 network has more authority than any other node. Thus for a true decentralised network you have to have a P2P architecture to support it.

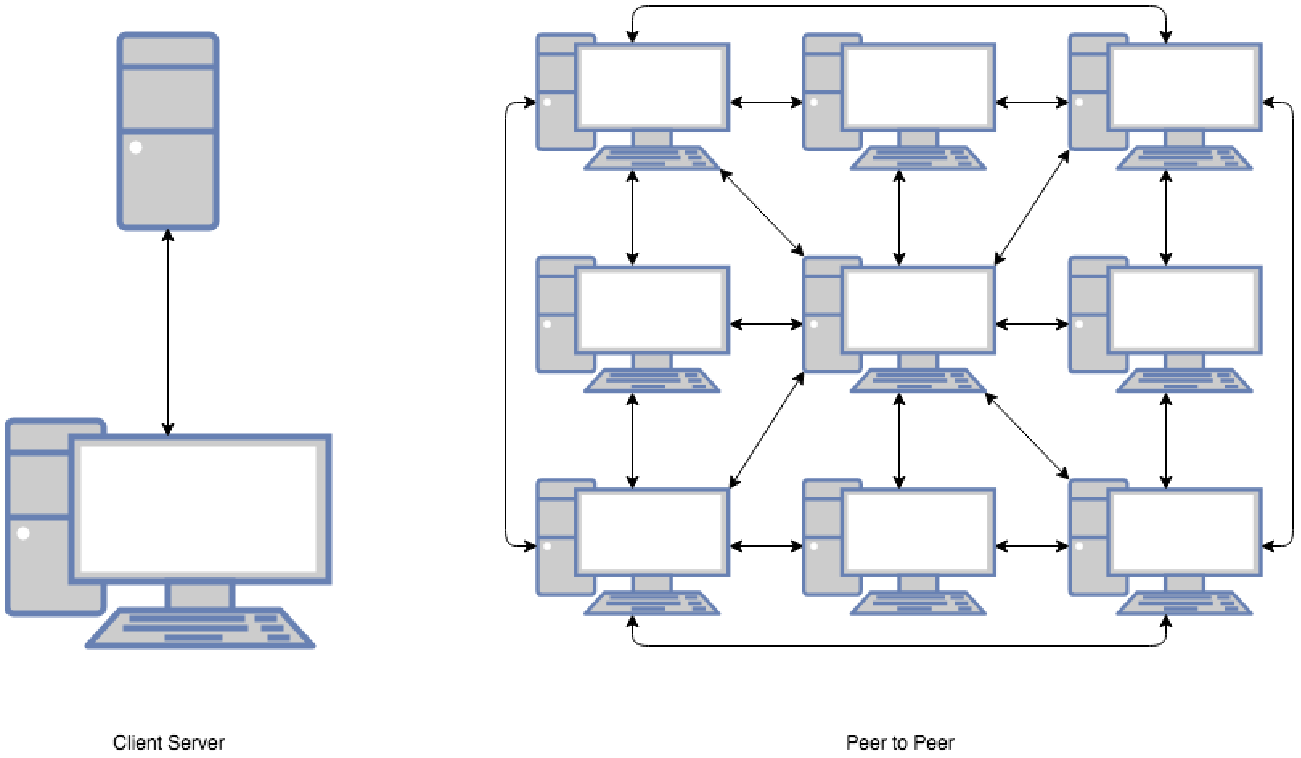


Figure 2.1: Client-Server vs Peer-to-Peer Network

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 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. Autonomy being a crucial pillar in the decentralisation of the network.

2.1.2 BitTorrent

The first stable version of the BitTorrent[12] 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[13]. 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. BitTorrent helps to solve many of the same challenges that the SAFE Network aims to. One of which is moving away from the traditional Client-Server architecture. In BitTorrent, peers form what is known as a ’swarm’. A ’swarm’ is all clients that aim to download a a full copy of a piece of data. Data is broken down into discrete chunks, each with a unique hash that allows clients to uniquely identify each piece of the original file. A client in the swarm is known as a *peer* when they don’t hold a complete copy of the file. A client 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 clients in the swarm are *seeds*. Nodes use P2P routing to send chunks of the file to other clients in the swarm that do not have it.

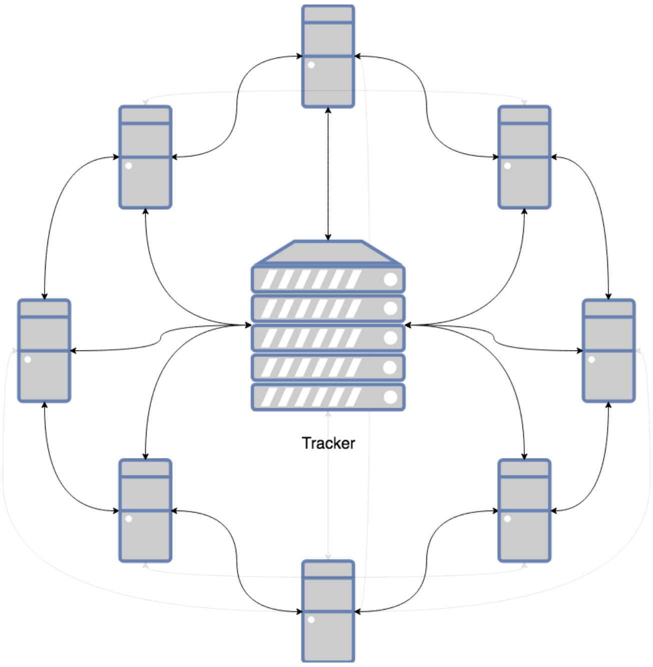


Figure 2.2: Topology of tracker based swarm in BitTorrent

In BitTorrent there is no central 'server' to attack (disregarding a *tracker*, there are *tracker-less* solutions available). You can see the topology of a tracker based swarm 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 clients 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 traditional 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 the network costs of sharing 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 sharing the file onto their users. This works on a 'good-samaritan' basis where if you download a file you should aim to have your *seed-ratio* hit at least 1 before leaving the swarm permanently (a nodes *seed-ratio* is how much of the file they serve to other users against how much they themselves have downloaded 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 downloading from. This means that in a well established swarm that your file will download as fast as your internet connection will allow. The more users 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 nodes wishing to downloading the data. Thus download speeds are limited by the resources of that single server.

There is a crucial aspect of BitTorrent that means lots of organisations cannot make use it. The issue is that of control. Once shared, a file cannot be easily removed from the network. Thus not having ownership of data on the network makes it unsuitable for some applications. Notably this means that the distribution of copyrighted content across the network is very difficult, once shared you don't have any control over that data. Licenses change and legal implications mean that copyrighted material is often withdrawn from public meaning

BitTorrent is unusable for such applications.

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*, you cannot interact with data without first joining the relevant swarm. So the discoverability of data is an issue. A given node within the network can't work out how to retrieve a chunk of data that is not located in the swarms it belongs to. 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/functionality to the client as possible. As time passes, the computational power of user workstations gets faster and faster. This computational power goes wasted for the most part. When you browse the internet, interact with Facebook for instance, your computer actually does very little in terms of processing the information you are seeing. Facebook serves to your browser a thin-client which can then make requests to Facebook for the data that is needed. Thus there is untapped potential on client side to perform more work locally instead of doing it server side.

There are drawbacks in offloading work to the client. The first issue, especially with websites like Facebook, is privacy. When Facebook processes the data on their servers, they can assure that only data you are allowed to view is sent to your client. If this data was 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 factor. Thus by using the traditional client-sever model you can offload the electricity expensive computation to the server and reduce power consumption on the device. This is in combination with reducing bandwidth to the client, which for mobile devices is a crucial factor in battery life. This means that when using the serverless architectural model in battery powered devices power consumption and network traffic must be minimised as much as possible. Not an easy task when large amounts of data need to be processed for rich and interactive content.

Vaults serve a similar purpose to *peers* in BitTorrent. Note that they do not serve a similar function to *seeds*, the network aims to never keep a complete copy of a single file in a single vault. The SAFE Network in this capacity can then serve a similar purpose to 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 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 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. You can open a browser that is capable of connectivity with the SAFE Network and browse the *internet* just as you would normally.

From the clients perspective, a vault only serves and stores data. No processing of the data can be performed on the vault. Thus SAFE Network applications must process all the data locally and only use the network as its storage 'back-bone'. Thus the *Serverless Architecture* model is a good fit for the SAFE Network. This method 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/... code is served to your 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.

The SAFE Network forces the *serverless architecture* architecture to be used unless you merely use the SAFE Network as a component in your stack. This introduces challenges in how you build and design applications. As you no longer have servers, you don't need to consider how your apps data will be served. This means that you can save time and cost in developing the 'back-end' to your software. Instead of designing websites and applications the *traditional* way, you develop them like you would a *fat-client*. Websites will become heavier, requiring more care and optimisations. Messy and slow JS is abundant in the internet today, mostly due to the abundance of computing power that exists. This hap-hazard way of coding cannot exist for *serverless architecture* websites or applications. 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 that is emerging that can help aid this is WebAssembly.

2.2.1 Web Assembly

Web Assembly is an assembly-like language that you can compile C, C++, Rust, etc, to and then run inside web browsers. It allows you to write code in high-level languages (that aren't interpreted like JS) and then serve it to users such that the code runs 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 architecture* model. A big strength of Web Assembly is in the processing of data. You can more easily write high performance code to process data in languages like C than you can in JS. Since the SAFE Network serves raw data to the client, the use of Web Assembly to process it could be a huge aid in increasing performance on lower power devices.

2.2.2 Serverless Fat Clients

A *Fat Client* is a computer (application) that can perform operations and tasks without relying on a central *server*. A Fat Client may still need to make periodic connections to a server but the vast majority of its functionality can be performed without *chatter* with the server. The concept of a Fat Client is juxtaposed to that of a Thin Client. A Thin Client is a lightweight computer/application that relies heavily on a server to have any sort of utility. It can perform some tasks locally but most are processed on the server before being sent to the client. 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 and more suitable for the development of fat clients. Instead of having to download desktop applications to your device, new technologies allow rich *serverless fat client* applications to be built and delivered through the web. This prevents users from having to change the way they browse the internet. If delivering *serverless fat client* applications through the web was not possible, users would be required to change to a model where they would have to download an app for Facebook, Twitter, YouTube etc. Hence I view the advent of new Web Technologies, such as Web Assembly, to be 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 means that the decryption key is publicly available so anyone can access the data. Vaults are still unable to determine the contents of the encrypted chunks they are storing.

In decentralised data storage networks tying ownership to real-world identities is difficult. In many cases there is simply no means to facilitate this. Ownership of data is one feature that the SAFE network provides as apposed to a system like BitTorrent. In BitTorrent, its not possible to uniquely identify the owner of a piece of data. If a user starts sharing data they might be the theoretical 'owner' but owing to the design of the protocol they have no ability to control the data. All users in a BitTorrent *swarm* have equal rights to data. In the SAFE Network, the ownership of data is more clearly defined. When data is uploaded it is split up into chunks and distributed across different vaults. Owing to the design of the SAFE Network that data has an identifiable owner, the account that originally uploaded it to the network. They can do with it what they wish, including deleting the data permanently. This contrasts with BitTorrent greatly wherein data cannot simply be deleted. To delete data in the BitTorrent network you would have to explicitly ask all nodes to delete the file, they have no obligation to do so.

Data that has been written to the SAFE Network is *immutable* without the credentials of the account that uploaded it. This is beneficial to users as it means they have the assurance that they 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. A scenario I can envision is users treating accounts as 'disposable'. One could imagine someone uploading the latest Hollywood blockbuster and then erasing the account credentials used to upload that file. Once lost, it is impossible to delete the data from the network. A solution to this could be through the use of a *master-key* to the network. This would be a decryption key that would allow complete access to all data on the network. 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 down to its use for illicit content sharing. This is a situation of ”you can't have your cake and eat it too”. 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. At the end of the day the SAFE Network is a tool. Like any other tool (BitTorrent) users will do with it what they please. The mitigation against illegal use of the tool should not impact on those who follow the law.

2.4 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* which is a cryptocurrency built into the SAFE Network. One could envision an application that instead of charing 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. One consideration of this approach however is that vaults don't get to choose what data they store, that is an integral part of the architecture of the SAFE Network. By following this financial model then it would be for the 'good of the whole', increasing the utility of the entire network and not just for one 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 the device. In the case of a games console, literally teraflops of processing power, advanced networking and storage facilities are going unused. Potential financial models can try to *exploit* this untapped power to the benefit of both the user and the provider of the application. One possibility facilitated by the SAFE Network is *farming Safecoin*. Not only does this increase 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.

2.5 Architecture of the SAFE Network

The SAFE Network is still very much 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. The network is thus still very much subject to rapid changes. 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 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, a Raspberry Pi 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'. This initial 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. Once *vault* is successfully storing data it is rewarded with *Safecoin*, which is 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 interact through the use of a *client*. To help increase privacy, a *client* connects to the network through an intermediary *vault* called a *proxy node*. This *proxy node* orchestrates the writing and retrieval of data on behalf of the *client*, hiding the *clients* IP address from the rest of the network.

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*. Once *vaults* start communicating with each other there is no intervention by humans. The network itself votes on and decides many factors. This includes everything from where data should be stored to how much value a *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.

3.1.1 Self-Encryption

Data that is stored on the SAFE Network is split up into small 1MiB chunks through a process called *self-encryption*. Each 1MiB chunk of data is then hashed to give a unique address in 256-Bit XOR Address Space.

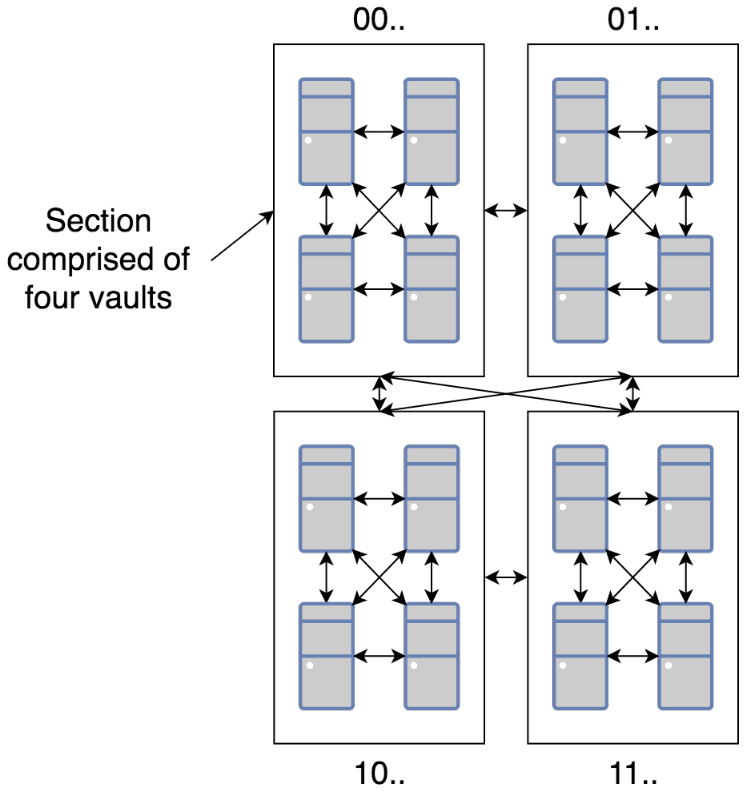


Figure 3.1: Four sections of a SAFE Network. You can see the address range each section is responsible for.

3.1.2 Disjoint Sections

The unique address of every 1MiB chunk from *self encryption* 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 XOR Address Space. By default, the network requires a minimum number of vaults to sustain the network. At the time of writing this is 8 vaults. These 8 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 and more complete *groups* of 8 *vaults* join the network, it continues to split and each *section* is therefore responsible for the curation of less and 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 very close together in 256-Bit XOR space but 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 each other. Otherwise the network could be open to simple attacks. For example, start 8 *vaults* on a single computer to form a *section* then suddenly switch them all off which could cause data loss. If a significant number of *vaults* leave the network then *sections* have the ability to join with other *sections* to ensure the stability of data is maintained. In Figure 3.1 you can see four *sections* comprised of four *vaults* each, you can see the address range that each *section* is responsible for. In the diagram four *vaults* make a *section* instead of the traditional eight, this is just to make the diagram easier to process.

3.1.3 Proof of Resource

The *proof of resource* (PoR) test is used to valid the effectiveness of a *vaults* ability to store and serve data and is the value proposition of *Safecoin*. The PoR is used to validate *vaults* joining the network but also during other network events. The process is as follows:

-

3.1.4 Personas

Vaults can be characterised as having different 'Personas'. The most *basic* persona that a vault can have is that of the Data Manager. A Data Manager is responsible for the storage of chunks within a section. Their job is vital to the stability of the network. When data is stored on the network, it is actually 'duplicated' across multiple Data Managers. At all times the network aims to keep a minimum number of copies of a chunk of data, if a chunk goes missing (say a vault goes offline) this chunk is quickly duplicated to another 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 Managers hold. The other persona a vault can take is that of the Client Manager. A Client Manager is responsible for storing the account data for clients. When you create an account on the SAFE Network, that data is stored like any other piece of data on the network. It has a given 256-Bit Address and contains the information like: how much Safecoin an account has, the number of chunks of data that has been uploaded, etc. As an account is a 256-Bit address it will fall within the domain of a particular section, the Client Managers in that section will then store the relevant data. As I will discuss in the section on Encryption, a vault does not know the IP address of the client that it is interacting with. The Client Manager thus doesn't know the IP address of the client it belongs to, it is just data and they cannot arbitrarily read the account data because it is encrypted.

3.1.5 Accounts

3.2 Crust and Encryption

Crust is the secure routing layer used by the SAFE Network. It was designed and built by Maidsafe to provide the secure communications backbone of the SAFE Network. Crust allows for reliable peer to peer connections and provides encryption for all traffic. I won't go into too much detail but some important points to realise is that Crust doesn't have a standard port required to function, it is capable of randomising ports. Several Transmission Protocols can be used, falling back to UDP from TCP (for example) if required. Encryption at this level means that Data on the network is always encrypted, data is only decrypted client side and whenever it is not on a clients computer it is fully encrypted.

Encryption is a very important aspect of the SAFE Network. Whenever data is stored on the network, it is encrypted. As mentioned above the only time data is unencrypted is when it is on a client. Data on the network exists as discrete 1Mb chunks, each with its own 256-Bit Address. When a file is uploaded to the network, it undergoes a process known as self-encryption. Self-Encryption is a pioneering technique developed by Maidsafe and is used to encrypt data. What happens is that when your file is broken down into 1Mb chunks, each chunk is encrypted with the hash of one of the other chunks. What happens then is a DataMap is constructed, this DataMap then contains the addresses of each of these individual chunks of data so that they can be retrieved. As data is stored on the network in this manner, you have a number of options on how to access it. You can choose to have data "unencrypted" or what Maidsafe calls "Plain", what this means is that any user that knows

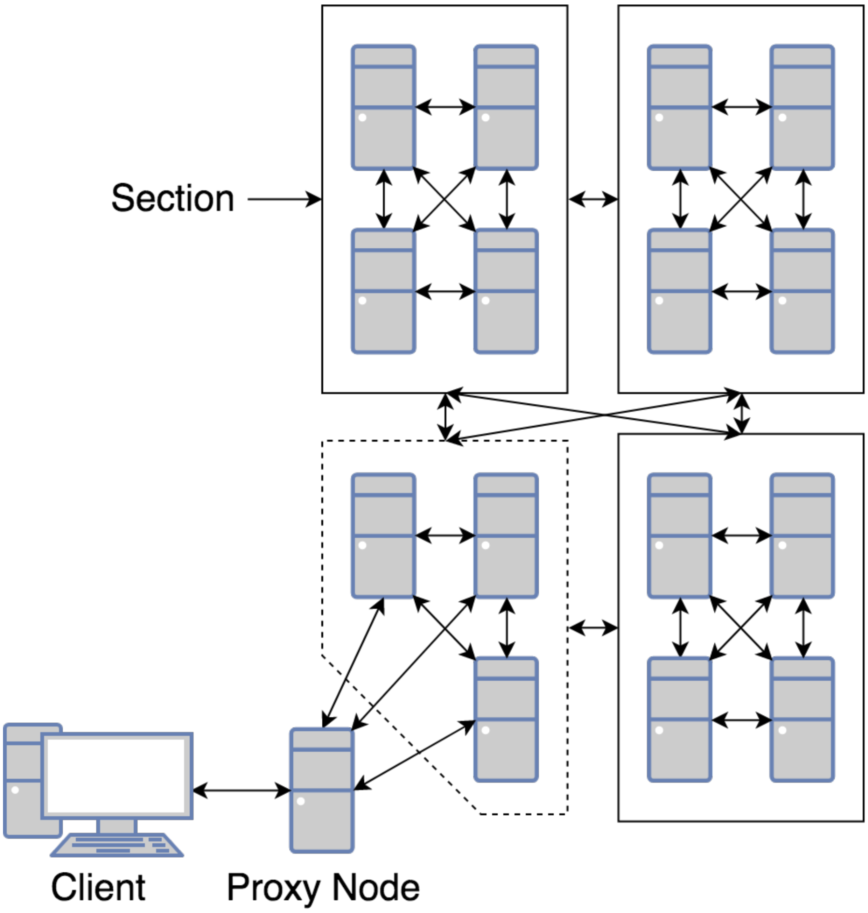


Figure 3.2: A client connecting to the SAFE Network through a Proxy Node

the address of the data (and the type-tag) can retrieve and read the data. The special thing about this is that the data is still fully encrypted on the network through self-encryption, a vault owner cannot decipher what the chunk of data holds. When anyone goes to access this data though, it is reassembled and you can read it. The two other types of encryption supported are Symmetric and Asymmetric. Having these options means that you can build applications in quite a flexible manner. A user can freely share the key to data and this opens up the possibility for interesting designs.

A system is also in place to protect a users identity as they connect to the network. This aspect of the SAFE Network is very important to my project. 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. When a user connects, the Proxy Node of course knows the the IP address of that client. Beyond the Proxy and deeper into the network all the vaults know is the XOR Address of the account being used (including other relevant data and public encryption keys). Hence by using a Proxy Node, the activity of the client is well hidden from the rest of the network. A given vault cannot detect that the data being retrieved is going to someone in a particular country etc. This means that clients can anonymously read and store data to the network without people being able to monitor the contents of that data. I will touch upon this later on when I discuss my project, this anonymity is very important. In Figure 3.2 you can see the topology of how a client connects to the SAFE Network.

3.3 Safecoin

Safecoin is the cryptocurrency of the SAFE Network, it is earned by farmers and spent by writing data to the network. The expectation is that as the cost of CPU/Storage falls with time, the value of the Safecoin will increase. As in, the amount of raw bytes that a given Safecoin would allow the storage of increases.

3.4 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 this case, financial transactions). 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 grows and grows. For Bitcoin and similar cryptocurrencies, to be able to undo a transaction/block you would need to have control of over %50 of the networks hash power. The debate on how easy it is to do that is a hotly debated topic that is outside the scope of this paper. The SAFE Network needs a similar mechanism on how to reach consensus. Analogous to a Blockchain, the SAFE Network has a 'Datachain'. This Datachain is used to help insure the integrity of the network and can be used to help rebuild the network incase of a catastrophic failure. For any action on the network to be valid, whether this be the storing 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 has. In order for an action to be 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. This means that in a given section, several vaults could be acting as 'bad parties' but network integrity wouldn't be lost. XOR Distance also comes into play in this process. The closer two sections are in 256-Bit XOR Address Space the more they know about the data the other section is storing. They will have access to the portion of the Datachain that is used by that section. This way, a given section can help to verify that a neighbour is acting as a good party in the network and that data being stored there has not been tampered with. The further away in 256-Bit Address Space two sections are then the less they know about each other. This means that as the number of sections increases, the influence a given a section has over the network decreases. Eventually resulting in no section in the network having an overview of the entire network.

A protection mechanism exists in the retrieving of data to account for the case 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 corresponding to a 256-Bit address. Alongside the data that is returned, a minimum 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, at the time of writing I have tried my best to summarise the current proposals. Things are subject to change as Maidsafe runs simulations and exams how things operate.

3.4.1 Node Age and Churn

A crucial part of the integrity of the Datachain is node ageing. In order for a vault to *vote* on network activity (this is the signatures that form the group signature) it has to have proved itself a reliable party. 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 Proof of Resource that we discussed earlier. If it passes the proof of resource then as long as the assigned section reaches a quorum on the new vault joining, then it joins that section. This node is very 'young' in the eyes of the network and as such is not trusted. It is not allowed to vote in group actions and is responsible only for the storage and transmit of data. A very interesting aspect of the SAFE Network is the concept of *churn*. Churn is used to constantly '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 vault is assigned to a new section, to which it must give another proof of resource. If the new section reaches quorum then the vault joins that new section 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 node reaches a certain node age does it become an *elder*. An elder is a node which has a high node age, meaning it has been up and running for a while and has proven itself to be a reliable party. When a node is an elder, it gains the voting writes that eventually lead to the construction and maintenance of the Datachain. Vaults that are not elders have no voting writes and essentially just do what they are told by the elders. 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.

3.5 Immutable and Mutable 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 (often abbreviated MD) is a key value storage mechanism that allows for the storage of 1000 entries at a maximum size of 1Mb. A MD has a 256-Bit address to specify its location. An Immutable Data Structure only stores a single 'value', its address on the network is derived from the hash of binary data it contains. An Immutable Data structure can itself only be 1Mb in size, but through the use of a Data-Map this limit can be subverted. I will talk more in depth about the Data-Map when we discuss my project, as its properties are very important to my application. As their names imply, Mutable Data can be freely changed and updated whereas Immutable Data cannot. Its clear to see that if you change the contents of Immutable Data then it will no longer correspond to the address at which it is stored. As mentioned previously, it is this property of Immutable Data that eliminates duplication on the network. If a user uploads the same file as another user they are simply presented with another *key* to access that data. If a user 'deletes' the data it will remain on the network as another user still maintains the key to access it.

3.6 Privacy and Anonymity

Chapter 4

SAFE Wiki

Ownership of data is very important. With verifiable ownership comes many avenues for interesting applications and is a trait of the network that gravitated me towards the application I decided to build. SAFE Wiki is an application that allows users to both upload and browse content that can be stored in a ZIM file. The ZIM file format allows you to easily store content from the web, one of its uses is in the distribution of Wikimedia based content.

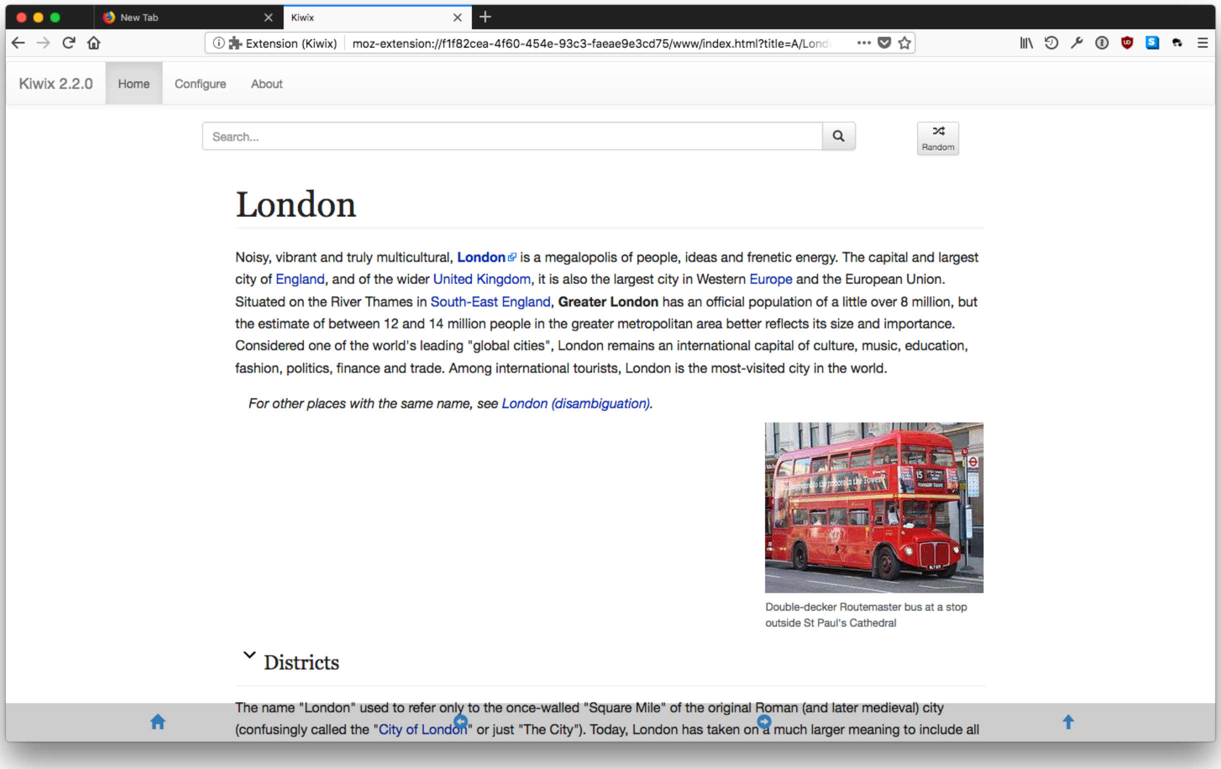
4.1 Kiwix

Kiwix was 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, *"the sum of all human knowledge"*, offline. Whether this be in the middle of the ocean, deepest Africa or even inside North Korea. Since the initial launch, different versions of the software have been released. Different versions support many different platforms, including: iOS, Android, Windows Phone, FireFoxOS, macOS, Windows and Linux. With such a fantastic history behind the project I saw Kiwix as the natural foundation to build my application on. Kiwix is inherently a *Fat Client* style of application. It does not rely on a network connection as that is the whole point of its existence. Hence building upon an already *Fat Client* application made perfect sense, considering the points made in Chapter 2. SAFE Wiki would do all the processing/reading of the file locally, using the SAFE Network only as a storage medium for the ZIM files.

Simply, Kiwix provides the means to browse the content within ZIM files. A user opens up 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. With the ability to follow links around the website (ZIM file) and search for pages. You can see what the London page of a Wikivoyage ZIM file looks like in *Figure 4.1*.

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 *Node.js* and what they call the *Web API*. Both of these require the usage of Javascript. Hence forking Kiwix JS and building SAFE Wiki on-top of it was a natural choice.


New Tab
Kiwix
Extension (Kiwix)
moz-extension://f1f82cea-4f60-454e-93c3-faeae9e3cd75/www/index.html?title=A/Lond
Kiwix 2.2.0
Home
Configure
About
Search...
Random

London

Noisy, vibrant and truly multicultural, **London** is a megalopolis of people, ideas and frenetic energy. The capital and largest city of England, and of the wider United Kingdom, it is also the largest city in Western Europe and the European Union. Situated on the River Thames in South-East England, **Greater London** has an official population of a little over 8 million, but the estimate of between 12 and 14 million people in the greater metropolitan area better reflects its size and importance. Considered one of the world's leading "global cities", London remains an international capital of culture, music, education, fashion, politics, finance and trade. Among international tourists, London is the most-visited city in the world.

For other places with the same name, see London (disambiguation).

Double-decker Routemaster bus at a stop outside St Paul's Cathedral

Districts

The name "London" used to refer only to the once-walled "Square Mile" of the original Roman (and later medieval) city (confusingly called the "City of London" or just "The City"). Today, London has taken on a much larger meaning to include all

Figure 4.1: Kiwix-JS running in FireFox

Kiwix JS as it stands has support for Wikimedia and StackOverflow ZIM files (although others may work, just not supported). This meant that if I could get SAFE Wiki working then it would be possible to not only browse Wikimedia content but also content coming from StackOverflow. The content that users could browse on the SAFE Network would be static, but with that comes some benefits.

4.2 Static versus Dynamic Content

When the idea to 'build a Wikipedia on the SAFE Network' first came to mind, I was very well aware of the fact that most likely nobody would use it. It would exist as nothing more than a tech demo with the slight hope that I would be able to continue developing it after I had finished my studies. Getting enough users to start contributing content, and building an environment where strict moderation could occur, would have been a fools-errand. Especially in the time permissible for this project. I just wouldn't have been able to build a full Wiki system on the SAFE Network and do it justice.

It is with that realisation that I came to discover Kiwix. Instead of trying to build a Wiki system on the SAFE Network and trying to bring users across, why not bring Wikipedia (and other sites) to the SAFE Network. This wouldn't be dynamic in anyway (in that users can edit the content) but it would be there for consumption. The storage of the data facilitated by all the features that the SAFE Network has to offer. An important thing about this approach is that by the end of the year I could have a working and browsable copy 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 any changes made. As there are thousands of users anything that is grievously wrong is likely to be flagged and addressed quickly. Similarly if someone is acting as a *bad-party* and editing pages wrongfully (think governments editing pages of staff members to make them appear in a better light) can be blocked based on IP address. This simple measure means that, in most cases, edits are tied to an IP address and hence can be blocked. A simple example is a school, I don't think it needs explaining that school children can be known for being rather silly sometimes. Resulting in the vandalism of some Wikipedia pages. As this is the case it is trivial for Wikipedia to block the IP address(s) that belongs to a 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 the ones outlined above that building a dynamic wiki (with adequate moderation techniques/tools) would have been very difficult and most likely a fools errand with the time I have allotted for this project. A static Wiki however, was very achievable.

A static version of Wikipedia might at first seam quite rigid, but in the context of the SAFE Network it makes sense. As the network has a concept of *ownership of data*, a ZIM file that has been uploaded can be directly tied to an account. The ZIM file belongs to someone (an account) in a verifiable *cryptographic* manner. An organisation like Wikimedia, or a trusted third-party, can then 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 access it, the only person that is allowed to modify (delete) the file is the holder of the account used to upload the file. As long as you trust the source of the ZIM file, you can *trust* that the information contained within it came from them.

The combination of Kiwix and the SAFE Network makes total sense within this context. A strong pedigree, and the foundation to read and process the ZIM files (static content), meant I could build a tool that could be useful to many people around the world.

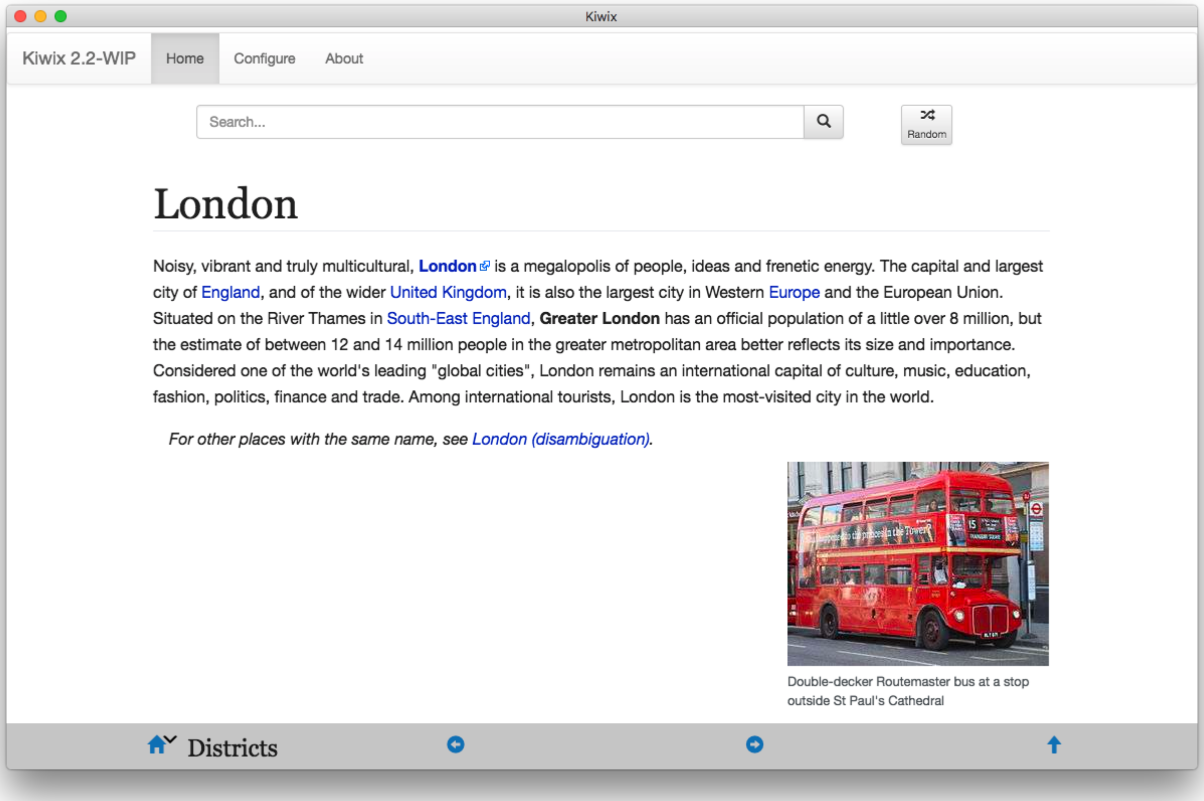
4.3 Electron

Electron 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 to me. The SAFE Network is not platform specific so SAFE Wiki shouldn't be 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 environment. Under the hood, 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 this meant that a single application could be built. A single application that could handle both the uploading and deletion of ZIM files on the network and also the browsing of them. The decision not to use the *Web API* to make SAFE Wiki was born out of the requirement of file uploading. To facilitate the upload of large files, (The ZIM for Wikipedia with images is >70GB) I really needed to build a desktop application.

Electron, and indeed Javascript, was very much a new world to me. Beyond simple web development I really haven't done anything note worthy in Javascript. Making Kiwix JS run as an Electron application was hence quite a challenge. After a couple of months of work though I managed to get it running. It was a case of 'completely broken' then one fix lead to 'completely working'. What was now SAFE Wiki (which you can see 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

Overall I found the SAFE Network exceedingly difficult to work with. I truly have never seen such poor attention to documentation and developer resources. My guess as to why this is is because they are still in development.



London

Noisy, vibrant and truly multicultural, **London** is a megalopolis of people, ideas and frenetic energy. The capital and largest city of England, and of the wider United Kingdom, it is also the largest city in Western Europe and the European Union. Situated on the River Thames in South-East England, **Greater London** has an official population of a little over 8 million, but the estimate of between 12 and 14 million people in the greater metropolitan area better reflects its size and importance. Considered one of the world's leading "global cities", London remains an international capital of culture, music, education, fashion, politics, finance and trade. Among international tourists, London is the most-visited city in the world.

For other places with the same name, see London (disambiguation).

Double-decker Routemaster bus at a stop outside St Paul's Cathedral

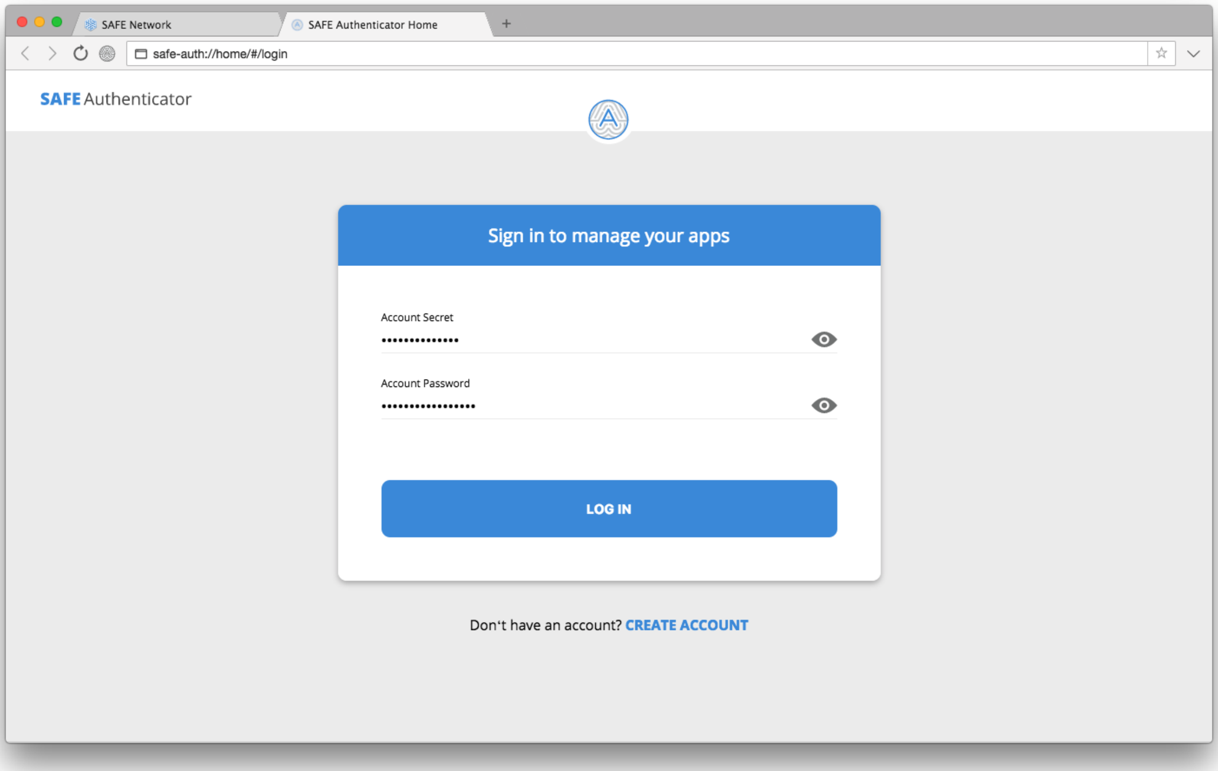
Figure 4.2: Kiwix JS as an Electron App

Being only at 'Alpha 2' they still have a long way to go before a true '1.0' release of the product. From that perspective I can forgive them. The only saving grace in this matter was the Developer Forum. The Developer Forum is a very lively place with constant chatter and everyone pitching in and sharing ideas. All that I needed to know about how to develop (work with) the network was contained in the forum. This was not an optimal way to find the knowledge I 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 me creating several posts of my own. I was amazed at how quickly people joined in the discussions and I usually had a resolution to my post within a matter of days.

Development first starts with how to orchestrate the connection to the network. During my project I used SAFE Browser, a fork of the Beaker Browser project. SAFE Browser takes the form of a web browser. Through it you can authenticate yourself with the network and browse any websites that are hosted on the SAFE Network, just like you can with a 'traditional' web browser. If you build a standalone application, like SAFE Wiki, you can use IPC to communicate with the SAFE Browser to gain authentication with the network. Once you have authentication you can communicate with the network directly meaning you don't need to use SAFE Browser as a middle man. Currently Maidsafe are working on the successor to SAFE Browser called Peruse, I don't have much experience with it so I can't comment on its utility.

To develop with the network you need to have some way of running your own 'development' SAFE Network. There are currently three ways of achieving this.

- Alpha 2 Network: This network is a public network 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,



Sign in to manage your apps

Account Secret
••••••••••••••

Account Password
•••••••••••••••••

Don't have an account? CREATE ACCOUNT

Figure 4.3: Login screen of the SAFE Browser

it is not optimal for developmental work.

- Mock Routing: Mock routing is a technique that is baked into the 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 local database. This is a very reliable way of developing locally, although it doesn't give you the full experience of how your application/web-site would work with a 'real' SAFE Network. What this method does offer is simplicity. As it is built into the SAFE Browser, that is the only download you have to make. You just start SAFE Browser (with mock-routing support turned on) and you just work with it as you please.
- Local Network: Running a local SAFE Network is my preferred choice. Sadly it was very long into development before I 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 run, but it is not difficult. One has to download and compile the 'safe_vault' from Maidsafe's GitHub. This is a *vault* that makes up the nodes of the SAFE Network. Once you have it configured, you then start up several vaults and they will automatically connect to one another. Once you have reached the 'min_section_size' you set, then you can reliably start using the network for development. The 'min_section_size' setting is used to configure the minimum number of vaults required to form one complete *section* on the network.

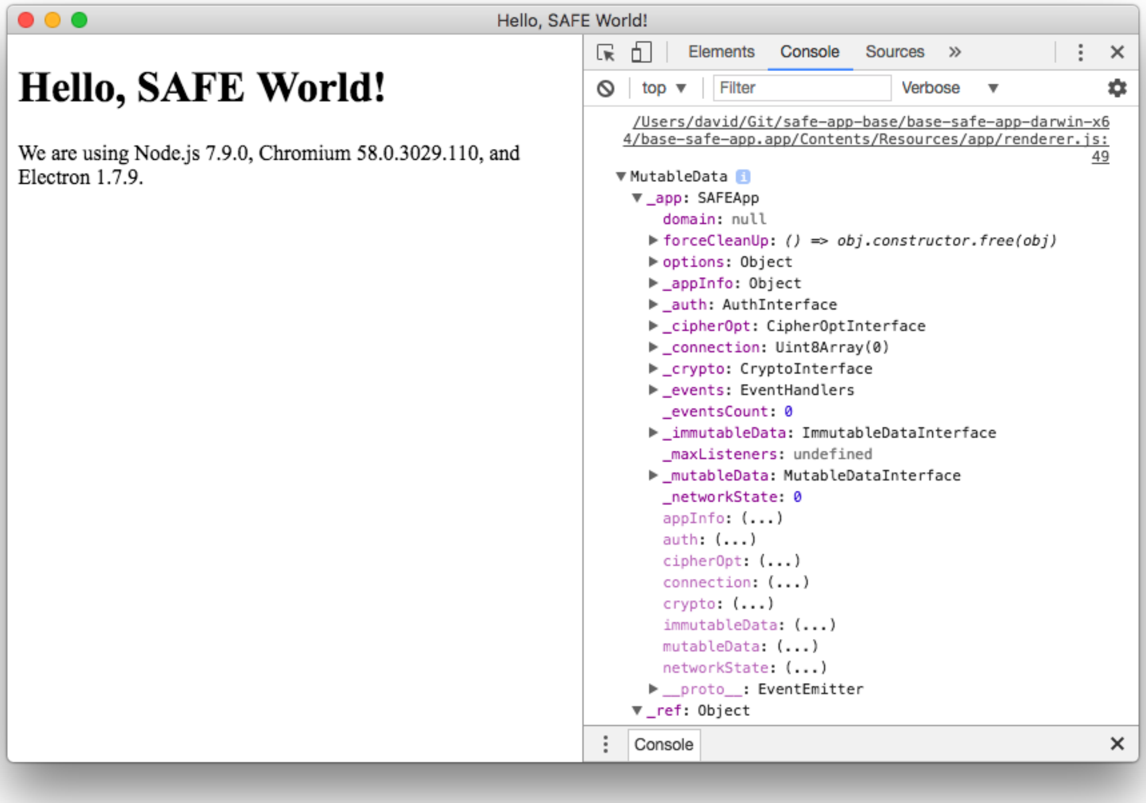
4.4.1 Web Hosting Manager Example Application

Maidsafe themselves provide a number of Electron example applications[14] . Looking through the code and how they worked was very helpful in figuring out how the *Node.js* API actually worked. A big challenge for me was just trying to figure out how SAFE applications should be designed, like 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 and more developers start working with the SAFE Network. For my purposes, I really liked the style of how the ‘Web Hosting Manager‘[15] example app worked. 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 meaning it was easy for me to learn how the vast majority of it should be used. One thing I noticed across all the example apps was repeating code. I don’t think it could be defined as ‘boilerplate’ necessarily but nonetheless it was repeating code. This led me to the decision to simply ‘fork’ the internal workings of Web Hosting Manager into SAFE Wiki as I would just be repeating code the same as Maidsafe has done. What I brought into SAFE Wiki was essentially the inner workings of the application. My application uses the SAFE Network in a far simpler way, so I only took the parts that would be useful in my application. Most notably was the code for reading local files to then upload to the network. By adapting the code from inside Web Hosting Manager I was able to use the SAFE Network in a way that is closest to what Maidsafe intended. As mentioned previously there really are no guidelines on how applications should be built, so I thought the most proper way would be to follow what Maidsafe themselves had done. 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 I developed my solution and my own style of doing things, the complete history of the changes can be seen on the SAFE Wiki GitHub page.

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 my application to the SAFE Browser for authentication was one of the most difficult parts of this project. As Electron allows (encourages) cross-platform development, what worked on a Windows computer might not work on Linux or macOS (the platform I worked on). Getting SAFE Wiki to run on all supported platforms was trivial, it just worked straight away. Getting SAFE Wiki to communicate with the SAFE Browser for authentication proved to be extremely difficult. Luckily, a community member had published an example SAFE Network Electron app called ‘safe app base’[16]. This application is a modified version of the application from the ‘Electron Quick Start’ guide[17], 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. I discovered though that on macOS the application didn’t work. What would happen is the SAFE Browser would successfully authenticate but the application would never receive the *auth-uri* needed to communicate with the network. I managed to deduce 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 you run the application 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 was a big setback for me because if this simple example application didn’t work then it would prove difficult to implement my own application. To help solve this I created a forum post[18] to discuss the issue with the community, I thought it best to fix the example app so that I would learn in the process and it would help other users. After some conversation with numerous people I managed to deduce how to solve the problem, I made the fix myself myself[19] and it got merged into the ‘safe app base example application. You can see the working application in Figure 4.4. Problems with URI Schemes cropped up later on in development too, resulting in another forum post[20]. This time the issue was with support on Ubuntu, thanks to the help of the creator of ‘safe app base’ I managed to get this issue fixed quickly. Resulting in me confirming support across Windows, macOS and Linux.



Hello, SAFE World!

We are using Node.js 7.9.0, Chromium 58.0.3029.110, and Electron 1.7.9.

```
/Users/david/Git/safe-app-base/base-safe-app-darwin-x6
4/base-safe-app.app/Contents/Resources/app/renderer.js:
                                                     49
▼MutableData
  ▼_app: SAFEApp
      domain: null
    ▶forceCleanUp: () => obj.constructor.free(obj)
    ▶options: Object
    ▶_appInfo: Object
    ▶_auth: AuthInterface
    ▶_cipherOpt: CipherOptInterface
    ▶_connection: Uint8Array(0)
    ▶_crypto: CryptoInterface
    ▶_events: EventHandlers
      _eventsCount: 0
    ▶_immutableData: ImmutableDataInterface
      _maxListeners: undefined
    ▶_mutableData: MutableDataInterface
      _networkState: 0
      appInfo: (...)
      auth: (...)
      cipherOpt: (...)
      connection: (...)
      crypto: (...)
      immutableData: (...)
      mutableData: (...)
      networkState: (...)
    ▶__proto__: EventEmitter
  ▼_ref: Object
```

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

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 both Immutable and Mutable Data Structures on the network 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.

4.6.1 ZIM Folder

Every *account* on the SAFE Network has a given 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. Note this doesn’t mean that data is not encrypted within vaults, it still goes through the process of self encryption, it just means the keys necessary are publicly derivable. Within the _public container then is a key value pairing where the key is ‘zim’ and the value is the XOR Address of a Mutable Data structure that is the ‘zim folder’. When a user creates a ‘zim folder’, they must specify a name for that folder. That name is then hashed to give an address which is a unique 256-Bit XOR address, the address at which the Mutable Data Structure will be stored on the network. Thus through the name of the ‘zim folder’, another user can locate the zim files uploaded by any other user.

4.6.2 Immutable Data, Data Maps and ZIM Files

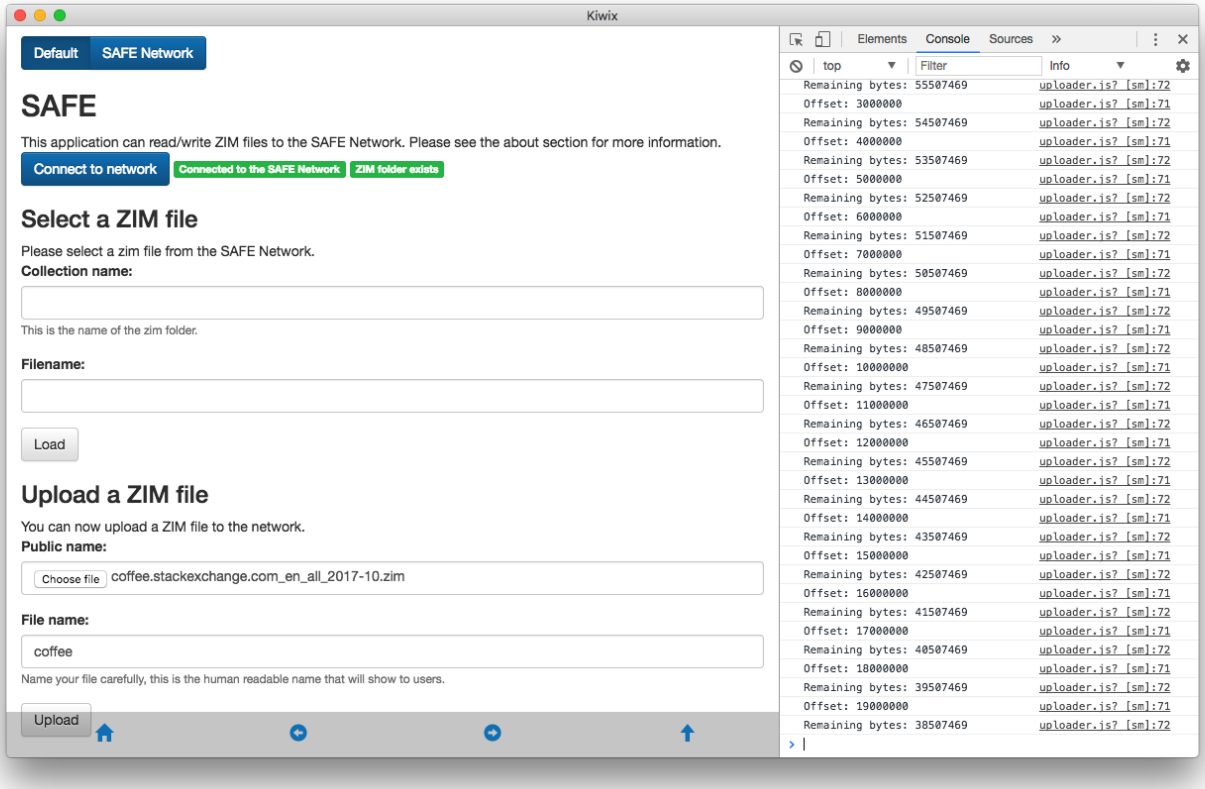
Once a user has created a ZIM folder, they are then able to upload ZIM files to the network. This is achieved through the use of the NFS emulation support of the *Node.js*. The target ZIM folder is emulated as a folder using NFS, then a ZIM file is placed within this folder. As the ZIM folder itself is a Mutable Data Structure, it allows for key-value pairings. In this case, the key-value pairs are ZIM file name to Immutable Data.

At all addresses in 256 XOR Address space, the SAFE Network is capable of storing a 1MiB chunk of data. This is regardless of whether or not the chunk is Immutable or Mutable Data. Immutable Data is best used for storing binary data but at only 1MiB in size you have to split large amounts of data across different chunks. This is handled by self-encryption through the use of ’Data Maps’. When self-encryption is used on a file, it ultimately breaks a file down into ¡ 1MiB chunks of data. Each of these chunks has a unique 256-Bit XOR Address which is where it will be stored in the network. To keep track of all of this, a DataMap for the data is created. 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 the given chunk
- pre_hash Vec<u8>: Pre-encryption hash of the given chunk
- source_size u64: The size of the original piece of data, before any encryption has taken place

Once this Data Map is stored on the SAFE Network, it is then possible to reconstruct and read files of any given size. A Data Map is itself a piece of Immutable Data, so it too can be put through the self-encryption process. What this means is that you can have files stored on the network that are bigger than the max size a

1MiB Data Map can handle. Self-encryption of Data Maps means they are broken into chunks and stored in a Data Map themselves, creating almost a hierarchy of Data Maps for a given file. This ability is very important to SAFE Wiki as some ZIM files are very large (>75GB). The API abstracts all the complexities of DataMaps away from the developer which means you don't really need to consider all of the complexities. You just write your file to the network and everything is handled for you.



SAFE

This application can read/write ZIM files to the SAFE Network. Please see the about section for more information.

Select a ZIM file

Please select a zim file from the SAFE Network.

Collection name:

This is the name of the zim folder.

Filename:

Upload a ZIM file

You can now upload a ZIM file to the network.

Public name: coffee.stackexchange.com_en_all_2017-10.zim

File name: coffee

Name your file carefully, this is the human readable name that will show to users.

```
Remaining bytes: 55507469          uploader.js? [sm]:72
Offset: 3000000                    uploader.js? [sm]:71
Remaining bytes: 54507469          uploader.js? [sm]:72
Offset: 4000000                    uploader.js? [sm]:71
Remaining bytes: 53507469          uploader.js? [sm]:72
Offset: 5000000                    uploader.js? [sm]:71
Remaining bytes: 52507469          uploader.js? [sm]:72
Offset: 6000000                    uploader.js? [sm]:71
Remaining bytes: 51507469          uploader.js? [sm]:72
Offset: 7000000                    uploader.js? [sm]:71
Remaining bytes: 50507469          uploader.js? [sm]:72
Offset: 8000000                    uploader.js? [sm]:71
Remaining bytes: 49507469          uploader.js? [sm]:72
Offset: 9000000                    uploader.js? [sm]:71
Remaining bytes: 48507469          uploader.js? [sm]:72
Offset: 10000000                   uploader.js? [sm]:71
Remaining bytes: 47507469          uploader.js? [sm]:72
Offset: 11000000                   uploader.js? [sm]:71
Remaining bytes: 46507469          uploader.js? [sm]:72
Offset: 12000000                   uploader.js? [sm]:71
Remaining bytes: 45507469          uploader.js? [sm]:72
Offset: 13000000                   uploader.js? [sm]:71
Remaining bytes: 44507469          uploader.js? [sm]:72
Offset: 14000000                   uploader.js? [sm]:71
Remaining bytes: 43507469          uploader.js? [sm]:72
Offset: 15000000                   uploader.js? [sm]:71
Remaining bytes: 42507469          uploader.js? [sm]:72
Offset: 16000000                   uploader.js? [sm]:71
Remaining bytes: 41507469          uploader.js? [sm]:72
Offset: 17000000                   uploader.js? [sm]:71
Remaining bytes: 40507469          uploader.js? [sm]:72
Offset: 18000000                   uploader.js? [sm]:71
Remaining bytes: 39507469          uploader.js? [sm]:72
Offset: 19000000                   uploader.js? [sm]:71
Remaining bytes: 38507469          uploader.js? [sm]:72
>
```

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

When a user uploads their ZIM file they give it a name, again 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.

4.6.3 Reading a ZIM File

To access a given ZIM file, all a user has to provide to SAFE Wiki is the name of the ZIM folder and the name of the ZIM file within that folder. The *address* for a given ZIM file hence consists of the name of the ZIM folder and its filename. This approach means it is easy to share access to ZIM files, as names can be human readable they are as easy to share as website URLs. The way I envision this being used is the name of the ZIM folder can correspond to the originator of the content then the filenames follow on logically from that. For example, 'wikimedia' could be the name of the ZIM folder then 'wikipedia' could be the name of the ZIM file. Meaning a user has two words to type in to access the latest version of Wikipedia. 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'.

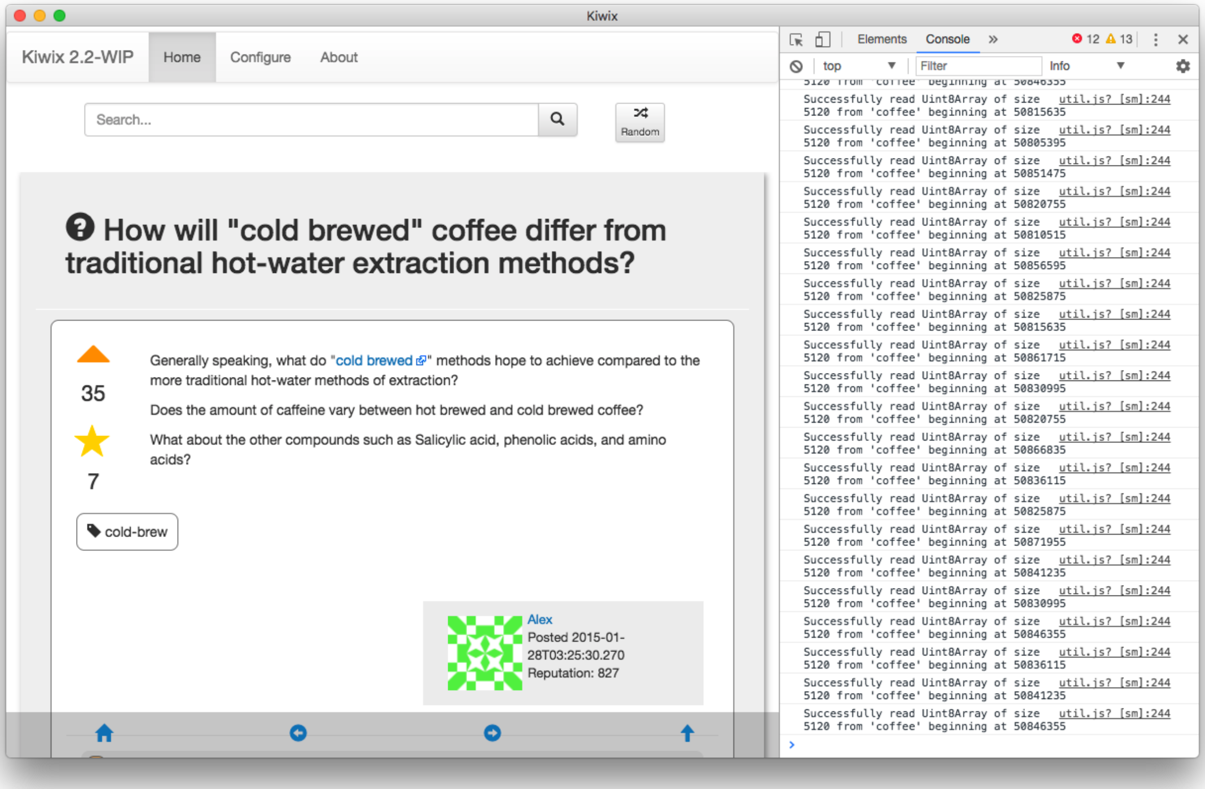
The resolution of the 256-Bit XOR address of the ZIM Folder is through hashing. As the target ZIM folder was stored at the 256-Bit XOR Hash corresponding to the name specified by the owner of the ZIM folder, the address is then derivable by anyone else that knows the name. Hence when any other user specifies that name, e.g. ‘wikimedia’, SAFE Wiki can easily find the Mutable Data Structure that houses the ZIM file the user wants to browse.

An important feature that is facilitated through the use of Data Maps is being able to randomly *seek* through files. The Data Map contains enough information about a file that you can read arbitrary bytes from the file without your client having to download and assemble the entire contents of the DataMap. For ZIM files this is important, it is illogical for SAFE Wiki to download then entire Wikipedia so that you can read through a single article.

```
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 4.6: Code to read a ZIM file from the SAFE Network

Kiwix JS by itself is setup to read ZIM files from local storage. To read files from the network all that happens is instead of reading a number of bytes (specified by begin and size) from local storage, the request is directed to the SAFE Network. Doing it this way is convenient because it doesn’t require a complete overhaul of file reading in Kiwix JS. This approach, being more modular in design, means that the original functionality of Kiwix JS is maintained. You can select whether to read files locally or to read them from the SAFE network. In the Code Listing 4.6 you can see how simple this code is. All that happens is the ‘zimFolder’ that is a Mutable Data structure is emulated using NFS. Then the ‘file’ is fetched through the ‘filename’ specified by the user (e.g. ‘wikipedia‘). To then read the required bytes is as simple as returning ‘file.read(begin, size)’. In Figure 4.7 you can see in the console on the right hand side the discrete reads from the network.



How will "cold brewed" coffee differ from traditional hot-water extraction methods?

35

7

cold-brew

Generally speaking, what do "cold brewed" methods hope to achieve compared to the more traditional hot-water methods of extraction?

Does the amount of caffeine vary between hot brewed and cold brewed coffee?

What about the other compounds such as Salicylic acid, phenolic acids, and amino acids?

Alex
Posted 2015-01-28T03:25:30.270
Reputation: 827

```
5120 from 'coffee' beginning at 50846355
Successfully read Uint8Array of size   util.js? [sm]:244
5120 from 'coffee' beginning at 50815635
Successfully read Uint8Array of size   util.js? [sm]:244
5120 from 'coffee' beginning at 50805395
Successfully read Uint8Array of size   util.js? [sm]:244
5120 from 'coffee' beginning at 50851475
Successfully read Uint8Array of size   util.js? [sm]:244
5120 from 'coffee' beginning at 50820755
Successfully read Uint8Array of size   util.js? [sm]:244
5120 from 'coffee' beginning at 50810515
Successfully read Uint8Array of size   util.js? [sm]:244
5120 from 'coffee' beginning at 50856595
Successfully read Uint8Array of size   util.js? [sm]:244
5120 from 'coffee' beginning at 50825875
Successfully read Uint8Array of size   util.js? [sm]:244
5120 from 'coffee' beginning at 50815635
Successfully read Uint8Array of size   util.js? [sm]:244
5120 from 'coffee' beginning at 50861715
Successfully read Uint8Array of size   util.js? [sm]:244
5120 from 'coffee' beginning at 50830995
Successfully read Uint8Array of size   util.js? [sm]:244
5120 from 'coffee' beginning at 50820755
Successfully read Uint8Array of size   util.js? [sm]:244
5120 from 'coffee' beginning at 50866835
Successfully read Uint8Array of size   util.js? [sm]:244
5120 from 'coffee' beginning at 50836115
Successfully read Uint8Array of size   util.js? [sm]:244
5120 from 'coffee' beginning at 50825875
Successfully read Uint8Array of size   util.js? [sm]:244
5120 from 'coffee' beginning at 50871955
Successfully read Uint8Array of size   util.js? [sm]:244
5120 from 'coffee' beginning at 50841235
Successfully read Uint8Array of size   util.js? [sm]:244
5120 from 'coffee' beginning at 50830995
Successfully read Uint8Array of size   util.js? [sm]:244
5120 from 'coffee' beginning at 50846355
Successfully read Uint8Array of size   util.js? [sm]:244
5120 from 'coffee' beginning at 50836115
Successfully read Uint8Array of size   util.js? [sm]:244
5120 from 'coffee' beginning at 50841235
Successfully read Uint8Array of size   util.js? [sm]:244
5120 from 'coffee' beginning at 50846355
>
```

Figure 4.7: Browsing a page from the StackOverflow Coffee ZIM file on the SAFE Network

Chapter 5

Evaluation

Chapter 6

Conclusion

Bibliography

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

[2] Jan. 2018. [Online]. Available: https://maidsafe.net/.

[3] 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).

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

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

[6] 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.

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

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

[9] Jan. 2018. [Online]. Available: https://www.dropbox.com/.

[10] Jan. 2018. [Online]. Available: https://onedrive.live.com/about/en-gb/.

[11] Mar. 2018. [Online]. Available: https://aws.amazon.com/.

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

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

[14] Mar. 2018. [Online]. Available: https://github.com/maidsafe/safe_examples.

[15] Jan. 2018. [Online]. Available: https://github.com/maidsafe/safe_examples/tree/master/web_hosting_manager.

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

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

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

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

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