Electronics and Computer Science
Faculty of Physical Sciences and Engineering
University of Southampton

Daniel J. A. Playle

Tuesday 28th April 2015

Building an anonymous P2P data storage network

Project supervisor: Dr. Tim Chown
Second examiner: Prof. Mahesan Niranjan

A project report submitted for the award of
MEng Computer Science

Abstract

The project addresses the lack of anonymous P2P data storage networks by designing and implementing a send-receive API for such a network. Existing networks and technologies have been examined to help aid the initial design, while the implementation of the API revolves around the protocol which has been designed by considering intended use cases and the elements of the protocol reflect this. The final protocol design focuses on the security of information, in terms of confidentiality, integrity and availability. The network attempts to allow for predictable availability of packets through expiry dates in conjunction with a proof-of-work system to prevent abuse. The design takes into account anonymity, but does not implement it, opting instead to use and benefit from the highly popular anonymity layer, Tor. As a result of the project, the implementation, named Stor, has been released under a BSD license.

Contents

Acknowledgements

I would like to thank Dr. Tim Chown for accepting my project proposal and giving continued support. I would also like to express thanks to the Electronic Frontier Foundation and the Tor Project for inspiring me to pursue a project in an area of growing relevance. As a result of my interest in this area, I am now a Tor relay operator.

1 Introduction

1.1 Background on the Issue

Over the past few years, there has been a shift towards decentralised services where protocols dictate how users interact with the system. This can be seen with Bitcoin, a decentralised currency based on cryptographic algorithms to ensure that only those following the protocol can use the system. In the world of anonymity, there is an increased interest in deanonymization attacks and they are becoming more common, through erosion of anonymisation technologies, social engineering or other means.

Prior to the project, when selecting a topic, a secure distributed anonymous social network was envisioned, however there was no existing framework that was suitable for purpose.

1.2 Project Goals

This project offers a solution to the lack of distributed anonymous communications frameworks by proposing a P2P[1] data storage network that can be run on existing anonymous infrastructure. The storage network will remove centralisation risks by allowing anyone to contribute to the network's resources. The aim of the network is to allow time and space decoupled communications anonymously. Through this mechanism of communication, services will be able to store their state, therefore allowing parties to act on this state and insert a new state into the network that other parties can further use.

In this project, security refers to the CIA triad, that is, security is achieved with confidentiality, integrity and availability. Centralised services can be considered a single point of failure, which increases the possibility of an attack on availability. Centralised services that do not provide end-to-end encryption may also present issues for confidentiality and integrity. For this reason, centralising a service should be considered to decrease the security of a system.

While some storage networks focus on store-retrieve behaviour, where packets are placed into the network and then retrieved later with some knowledge of the packets' existence, this project focuses on send-receive behaviour, where packets are sent to identities for them to be received from the network later. This also allows for store-retrieve behaviour.

The network shall be designed to satisfy the properties:

1. Users of the network cannot be identified
2. The contents of a packet can only be read by a recipient
3. The sender of a packet cannot be identified by anyone except a recipient
4. The recipient of a packet cannot be identified by anyone except the sender

The project has focused on the creation of two elements, the protocol for the network and an API to access it. The protocol is an abstract concept that describes how members of the network organise themselves, topologically speaking, and the rules

[1] Peer-to-peer, a type of decentralised topology.

behind communication. The API is a more concrete entity that implements the protocol to allow other applications to communicate using the network. While the API is more concrete, the aim is to keep it abstract and modular in places to allow any future development around the API to be as flexible as possible.

2 Research and Literature Review

2.1 Existing Systems and Technologies

Bitmessage

Bitmessage is a P2P communications protocol that enables parties to communicate with messages. Each party[2] holds at most 8 connections to other nodes in the network. Inserting a message into the network is done by encrypting the message for the recipient and forwarding it along all the connections. Nodes store messages for 2 days before they are removed. To retrieve a message, each message must be acquired and decrypted. Although Bitmessage's aim is a secure communications network, it does not address anonymity, having the undesirable affect that it is possible to tell who is using the network. Although it is possible to run Bitmessage over Tor, a node doing this is not able to accept connections [16].

Freenet

Freenet is an anonymous storage network that attempts to implement its own anonymity layer. Files are inserted and retrieved by relaying requests through successive nodes until a request is successful or until the request's TTL[3] expires. As a file is successfully relayed back to the requester, each node also caches it [5].

Because Freenet uses direct connections between nodes, the limiting factor for network speed is the slowest node in the request chain. However, if all nodes are reasonably fast, Freenet has a high potential for fast transfer speeds.

Tor

Tor is an anonymity network that implements onion routing to obscure traffic patterns. The Tor network consists of relays run by volunteers. While most Tor relays only route traffic around the Tor network, some (exit relays) allow connections to be made to the wider Internet from within the Tor. Under typical usage conditions, onion routing will select a collection of these relays from the public list, using an exit relay as the last relay if the destination is on the wider Internet. The data stream is then encrypted with successive layers of asymmetric encryption using the relays' public keys so that when a relay receives some data, it can decrypt it and forward it to the next destination [6].

A key feature of Tor is its hidden services, which are services that are hard to physically locate, meaning that it is possible to run services anonymously. Tor's hidden services provide identity verification using the 16 character onion address and as all communications stay within Tor, all traffic is encrypted. The technical specification of hidden service creation, discovery and rendezvous is described by [11].

[2] A party may or not be a participating node.

[3] Time to live, a value that is decremented every hop.

Proof-of-Work

Achieving consensus on a traditional network may be possible where there is a link between user and some available identity. This can be achieved on an anonymous network where a trusted central authority is introduced that assigns identities to users, although doing so also introduces a central point of failure. For a distributed anonymous network, consensus can also be reached through the use of proof-of-work algorithms.

A proof-of-work is a guarantee that some resource has been used to create it. In many cases, this will in the form of a CPU-bound proof-of-work where a heavy computation is required to produce a proof that is easy to verify. Originally described by Back, Hashcash uses a CPU-bound proof-of-work to compute partial hash inversions [1]. While this is common, CPU-bound proof-of-works can typically be computed quickly on Graphical Processing Units (GPUs), and even faster on Application Specific Integrated Circuits (ASICs). This has a high potential to cause a rift between those using low power devices and those able to create or afford hardware such as ASICs. This gap has been observed with the proof-of-work system used by Bitcoin [9].

Memory-bound proof-of-works are a potential solution to this issue, where memory access latency is the bottleneck for generation. As this latency difference between cheaper and more expensive device is less extreme than with processing difference for CPU-bound proof-of-works, this has significant potential to address inequality of availability for those generating proof-of-works, increasing accessibility to those that cannot afford expensive hardware. An implementation of such a system is described by Tromp [15].

2.2 Problems and Issues

Availability of Information

The lifetime of data is one metric of availability, it defines how long some data will persist on the network before becoming unavailable. The distribution of data is also an important metric of availability, which is measured by the number of active nodes that hold some information.

Networks such as Freenet approach this by increasing the availability for popular data, having nodes as part of the request chain cache the data. In this scenario, the lifetime is strongly correlated to the distribution. Bitmessage on the other hand takes an approach where distribution is universal, but data expires after 2 days, purposely decreasing availability to free network resources.

Visibility of the Network

Networks that do not address anonymity often allow the harvesting of nodes, as node identities (such as IP addresses) are publicly visible. For an anonymous system, this is unacceptable and some form of masking is required.

If some communicating parties can all be trusted and known to each other, it is possible to create a small network based just on these parties. Freenet has support for such networks, calling them darknets. While suitable for small groups where

everyone is trusted, if the requirement for the network is public availability, where parties cannot be trusted, then darknets cannot be used.

Deanonymization Attacks

In the past, anonymous services deemed to be secure have been subject to deanonymization attacks. An example of this is the take down of Silk Road, a black market hosted using Tor's hidden services. Although there is a question over how Silk Road was taken down, given that it was a centralised service, with a single party in control, it could not be considered secure, as removing just one server removed the entire service's availability.

Trust

Nodes can be malicious, they can attempt to cheat the protocol, perhaps by reporting that they are participating and sharing a resource, but in reality they could be doing nothing. Networks relying on participation have to deal with this by classifying nodes are malicious or not. This could come in the form of a simple mechanism where nodes that misbehave n times are blacklisted, or it could be a more complex machine learning algorithm.

Sybil Attacks

On any network, it may be possible for one user to control many identities, and therefore pretending to be many users. This can cause issues in any system attempting to use consensus based on identity. The typical remedy for this attack is to link a high-cost resource to identities. IPv4 addresses are commonly used for this purpose, as there is a relatively small finite pool and spoofing is considered difficult. IPv6, with it's relatively large pool of addresses may not offer such protection [4].

Due to the nature of anonymous networks, it is not possible to use identifiers, such as IP addresses, as a resource. Instead, different attack negation mechanisms must be used. A trusted central authority could assign identities which could then be used on the anonymous network, but introduced a centralised point of failure. Proof-of-works may offer a solution, where elements of a system with any form of consensus can be protected with a guarantee that a valid proof-of-work required some resource to be used. A requirement for a proof-of-work for all communications could render a large Sybil attack difficult [3], although this could decrease network availability to some users.

Ultimately, while some methods are able detect or prevent some Sybil attacks, Douceur reasons that "without a logically centralized authority, Sybil attacks are always possible" [7].

Spamming and Denial of Service Attacks

While partially related to Sybil attacks, denial of service attacks attempt to limit the availability of the network. In the context of a distributed storage network, this could quite easily come from a user constantly inserting data into the network and therefore exhausting the network's storage capacity.

To prevent this kind of attack, some form of consensus is required to determine what data is legitimate. This therefore has the same issue as Sybil attacks.

2.3 Cryptography

Encryption comes in two forms, symmetric and asymmetric. A symmetric cipher is one where the same key is used for both encryption and decryption. If used for communications between multiple parties, this requires that the key has been distributed between the parties, which can be problematic. Asymmetric ciphers on the other hand work on two keys, a public and private key. To encrypt a message, the public key is used, while decryption is only possible with the private key [12].

Padding is a cryptographic mechanism that further adds to confidentiality. If for instance a sender were to use asymmetric encryption to send an encrypted message to two other parties using their respective public keys. If one of these parties were to decrypt the ciphertext with their private key and re-encrypt it with the other recipient's public key and match this to a previously intercepted communication, it would be possible to use this known-plaintext attack to deduce the communication that has occurred between the sender and the other party. This can be prevented through the use of a padding scheme, where the message is padded with some value and then encrypted. Upon decryption, the padding is then removed. Through this mechanism, it is no longer possible to determine the intended recipient of a ciphertext just from knowing the plaintext and an intercepted communication.

Padding also offers protection against traffic analysis, where it may be possible to determine communication content from the traffic's behaviour. This is shown by Wright, where the lack of padding is shown to weaken an encrypted VoIP link [17].

Confidentiality by encryption is described by RFC-4880 through the use of a hybrid cryptosystem [8]. This is achieved through the generation of a random symmetric key which is used to encrypt the message, possibly along with a digital signature. This random key is then encrypted with the recipient's public key. Both the symmetrically encrypted message and the asymmetrically encrypted key can then be sent to the intended recipient.

3 Design

3.1 Network and API Usage

Before the design can be considered, the expected use of the network must be examined. Although the network is for the storage of data, this data is not intended to be in the form of large files, rather, it is expected that services can use the network to store their state. A concrete example of such a service could be a social network, where users can associate with each other, update their profile and make posts. Upon performing actions, the user would send an update to each user they associate with, which they would later receive and act on accordingly.

The aim of the protocol is to write the rules of internodal communication within the network. The protocol should therefore define how a node can connect to the network, discover other nodes, the process of transferring data between nodes and the logical topology of the data stored.

The goal of the network is to allow messages to be sent and received later by the intended recipient. This should be done in a way that makes the message highly available, as any services will desire guaranteed, predictable availability.

The API should provide an interface with the network, abstracting the details of the actual insertion/retrieval away from the application. Insertion of a message happens when the message will have some function applied to it and be distributed across the network to make it highly available, such that the function makes the message unreadable to anyone that is not a recipient. To retrieve a message for a particular user, all data in the network can have the inverse function applied to it to receive messages intended for them. In reality, this function is likely to be a form of asymmetric cryptography operating on the public key of the recipient and the inverse an operation of the recipient's private key.

3.2 Considered Elements for Designs

Several key elements of a storage network were identified and different design decisions based around these elements have been considered.

3.2.1 Network Connectivity

In order to join the network, a node must have some knowledge of where to find other nodes. This can be achieved through the use of some central authority that holds a list of nodes, enabling anyone to acquire the knowledge to use the network, or each node can act as a part of a decentralised directory, where as long as one node can be connected to, the remaining participants of the network can be discovered.

Neither solution is without issues: the central authority could reduce security by providing a more important class of node that could be seen as a target by an adversary, thereby reducing the security, while the acquisition of the knowledge needed to connect to the initial node would have to occur out-of-band and any information from a decentralised directory may be subject to abuse as an element of trust is required.

Although the initial connection node may seem to be much the same as the central authority, it would be possible to have several of these initial connection nodes, and as long as they are members of the network, this is acceptable. This presents any application the freedom to use any nodes as initial connection nodes, as opposed to limiting it to a single central authority.

As the introduction of a central authority presents too much risk and lowers the amount of freedom compared to a distributed directory, the API should provide a mechanism to allow users of the API to set initial connection nodes in order to discover the rest of the network.

Alternatively, if all users have access to some information stream where steganography could be used, it may be possible to use this for the purpose of advertising nodes. The Bitcoin Message Service uses a similar mechanism for the distribution of messages in the Bitcoin blockchain [2]. As all Tor users have access to Tor directory nodes, it was hoped that hidden service descriptors could be used for steganography, however, due to the nature of Tor's directory nodes, this method is very unreliable, although possible, as some malicious nodes have used this mechanism to harvest hidden service descriptors [13].

Given the nature of anonymous services, such as Tor's hidden services, the initiator of a connection is aware of the destination network identifier (or, pseudoidentity), as this is required to make the connection, although the target of the connection has no knowledge of the initiator's psuedoidentity. This has the result that connections can be considered unidirectional, although the transfer of data is bidirectional once the connection is established. In some cases, determining the pseudoidentity of the initator is desired and so this presents an interesting challenge. The initator could tell the target their pseudoidentity, which the target could either check in some way, or trust. Alternatively, as the pseudoidentity is likely to have asymmetric key backing, a digital signature could be appended to a message, proving the pseudoidentity of the initator.

$$\left(\prod_{i=0}^{n}\left(1-\frac{1}{N-1-i}\right)\right)^{N-1} \qquad \begin{array}{rl} N = & \text{network membership} \\ n = & \text{connections per node} \end{array} \tag{3.1}$$

The topology of the network describes how nodes organise themselves and connect to each other. In some networks, such as Freenet, nodes organise themselves into a small-world topology, where any node can communicate with any other node through some number of other nodes. This is done as a mechanism for implementing anonymity, but creates an attack vector where an adversary may be able to manipulate the topology to allow for attacks [10]. As an existing anonymity layer will be used, there is no need to use the topology to keep nodes anonymous. Rather, it will be considered mainly for purpose of information distribution.

Since any topology that allows node connections to be directly manipulated presents a potential attack vector, a purely logical topology, where nodes each decide who they connect to unidirectionally should be used. Since this method relies on no external influence to any node, there is a chance that some nodes may not be selected for a connection by any other node, thereby excluding them. While this is only an

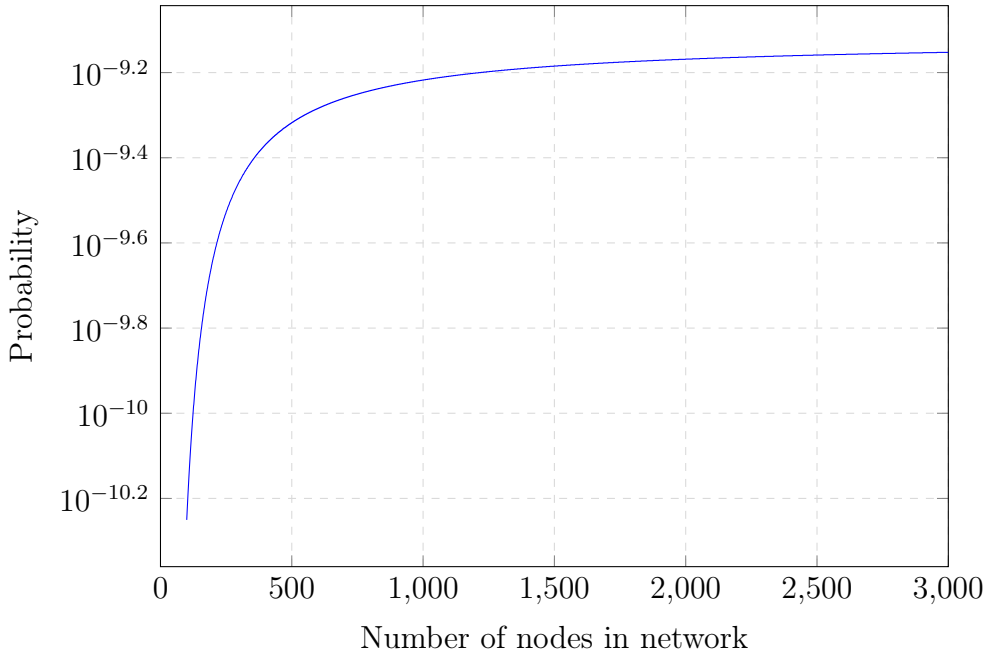
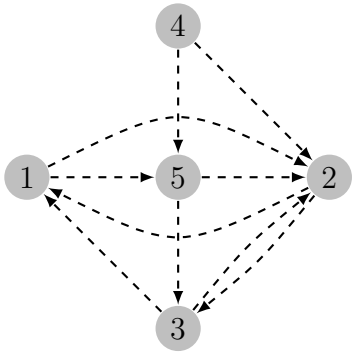


Figure 3.1: Probability of a node not being selected by any node for internodal communications (where $n = 20$)



This example network consists of 5 nodes, each with 2 connections. In this case, node 4 has not been selected for a connection, while node 2 is connected by every other node.

Figure 3.2: High-level internodal connection diagram example

issue if a large number of nodes are not selected, as this effectively reduces the potential resources in the network, this number should be kept to a minimum. Formula 3.1 shows the probability of any node not being selected by any other for a unidirectional connection. Figure 3.1 shows this probability where each node maintains 20 unidirectional connections out. As shown, this probability is very low and even when the network is large, under the condition of periodic connection renewal, so that newer nodes are included, it is likely that all nodes will always be selected for a connection. Figure 3.2 shows an example network with the internodal connections.

3.2.2 Packet Classes

Data as Notifications

A single class of data packet is used, where the intended message and notification of the existence of said message are combined. This means that if a

notification is successful, the data will always be available, resulting in lower delivery latency. However, as the size of some packets could be large, and the majority of this information is destined for most nodes, this is likely to be wasteful and could cause scalability issues.

Notifications Referencing Data

Two classes of packet are used: a data packet that is just for storing data and a notification packet that is only for notifying a node about the existence of a data packet. While all nodes still need to check all the notifications, only a subset of all nodes need to be in possession of the data packets. In addition, the notifications may also include information such as some known locations of the desired data packet (rendezvous nodes). This method has the potential to reduce the availability of data packets that are not distributed well throughout the network.

Notifications with Small Payload

Each notification packet is a union[4] of a small payload or a data packet reference. The total size of this is still comparable to that of a typical notification packet. The result of this structure would be very high availability for any small payload.

Given that notifications with small payloads have the potential to provide maximal flexibility while also reducing the load on the network, these will be used.

3.2.3 Information Distribution and Delivery

Distributing the information across the network must occur in order to keep the availability of information high. In order to receive a message, everyone must either check every message, or there must be some form of delegation to a third party, or parties, that must perform the same role.

Systems, such as Bitmessage, take the approach that everyone can attempt to determine if a message is for them, whereas other systems, such as email, take advantage that the recipient of a message can be determined and use this to place the message in a predictable location. Other networks, such as Freenet, move the obligation of notification to out-of-band communications, meaning they do not address this issue.

As the Bitmessage network grows, it splits into "streams" [16], effectively creating smaller subnetworks. Membership of a stream can be a function of some identifying element of the user, such as a public key (figure 3.3). Although this method is promising in terms of scalability, it has some disadvantages for the anonymity of the network. This type of optimisation is only possible where each stream has enough members such that if the stream was isolated, the property of anonymity would hold. Under the assumption that one user controls exactly one node, it is enough for the node membership of any stream for be sufficiently high. However, if this assumption cannot be made, then it opens the possibility to deanonymisation attacks. The worst-case for this attack is that it may be possible to determine the public key, and therefore the identity that a node is interested in retrieving

[4] A union type is one that provides an either-or contents structure.

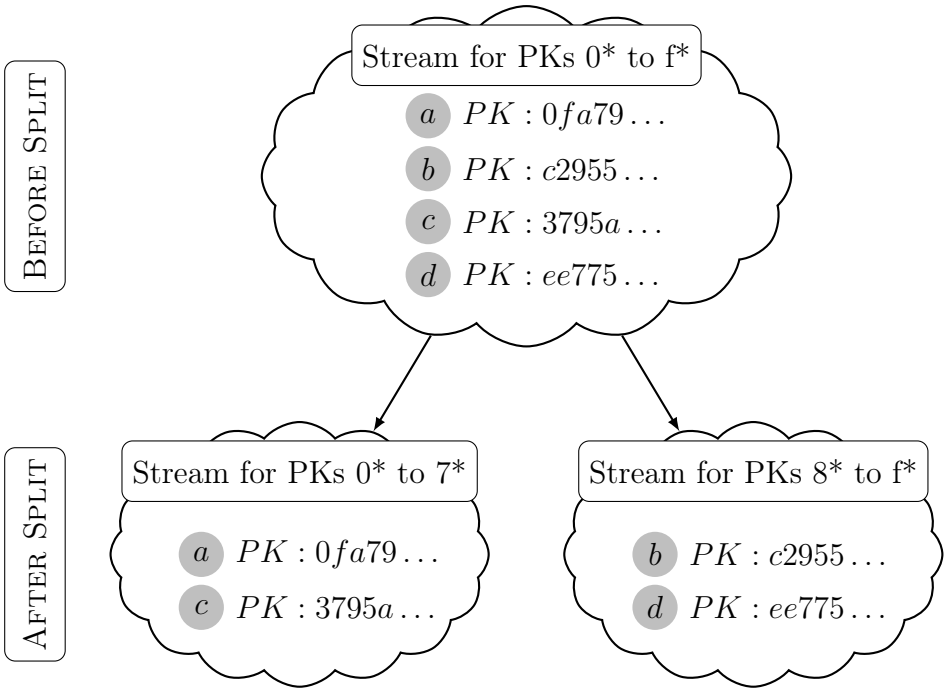


Figure 3.3: Example of intended stream usage

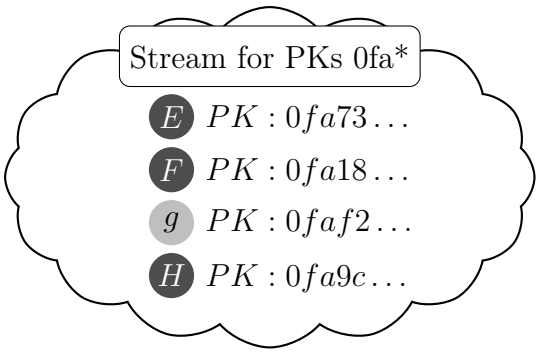


Figure 3.4: Example of malicious nodes isolating a legitimate node

messages for. This would occur where an adversary is able to isolate a node in a stream (figure 3.4) and could happen where an adversary creates a large number of nodes with similar public keys (through trial and error, or otherwise) to the target node. Although this may only reveal the recipient's identity for new packets in the stream where the isolation attack is taking place, it is still undesirable.

While all nodes need to check all messages to a degree, this mechanism is only required to notify of a message. Hence, instead of combining the notification and data functionality in a single packet, and having everyone acquire all of these, a split can be made. This would still require everyone to acquire and check all the notification packets, but these notifications could point to a data packet elsewhere on the network, along with a collection of rendezvous nodes that are known to be in possession of these data packets.

3.2.4 Broadcast Handling

Notifications will be broadcast to at least all nodes that are members of a stream, this is likely to be the bottleneck of the system. The handling of broadcasts should therefore be subject to optimisation where possible.

Universal Polling (UP)
: Nodes query other nodes for a list of their stored broadcasts, acting like a distributed hash table. The querying node can then request any broadcast it does not have stored. If the rate of broadcasts inserted into the network is greater than the rate they expire, then the size of the query response will grow unsustainably.

Active Forwarding (AF)
: Upon receiving a broadcast for the first time, the node forwards it to all of its connections. This does not require any list of broadcasts to be distributed, saving on some overhead. However, as the node initiating the forwarding is unaware if the nodes are in possession of the broadcast, the receiving node will be responsible for signalling that further transfer is not required, probably in the form of terminating the connection.

Selective Polling (SP)
: When a node receives a broadcast, it also stores the time of receipt. When a node polls another node, it stores the time of polling for that node. In a similar way to UP, polling is used to gather a list of broadcasts, but a parameter specifying the last time of polling is passed too. The polled node returns the list of all broadcasts that it received after this time. This has potential anonymity implications where it may reveal the time at which a node received a broadcast. However, this can be negated by having the receiving node add a small random time to the real time of receipt.

Simulations for each of the 3 broadcast handling methods were run for a set period of time. For the interval up to this time, nodes are stimulated, adding tasks for them to perform, such as querying another node or pushing data into the network. After the allowed simulation time, no more stimulations occur and the remaining tasks are allowed to complete. The interval from stopping stimulations to the end of the simulation is referred to as the simulation lag. More detail on the simulator can be found in appendix D. It should be noted that time, in this context, is measured in simulation ticks, not the real time the simulation takes to complete. Small simulation lag values are what is expected to result from a sustainable system where tasks do not accumulate.

Simulation results are not absolute, and real scalability and performance results will be dependant on real network usage. The simulations presented here, subject to minor variation in the random nature of simulating usage, have been conducted under the same usage conditions.

Running the simulation for UP for a set time, while varying the size of the network results in the graph shown in figure 3.5. The point where the method fails for a simulation interval of $150 \cdot 10^6$ ticks, can be seen to occur at 80 nodes. For UP, the points of failure for different simulation times were plotted, as shown in figure 3.6.

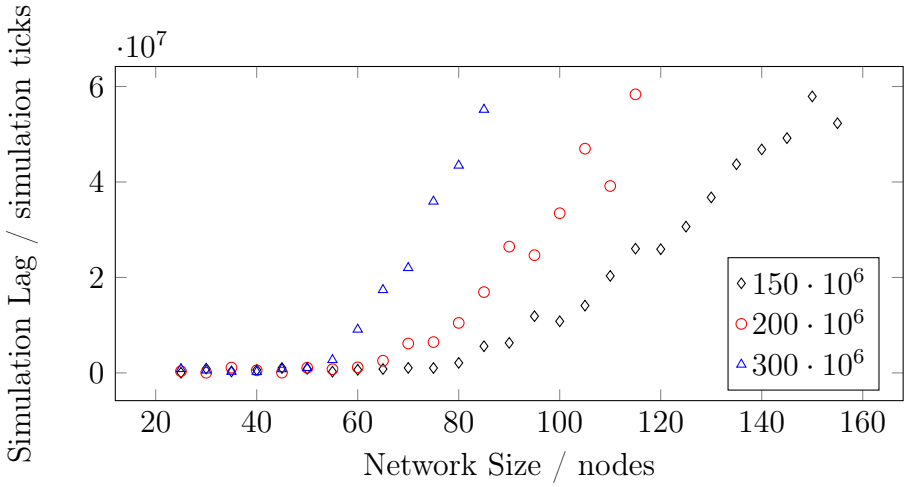


Figure 3.5: Simulation lag for network sizes for varying simulation lengths (UP)

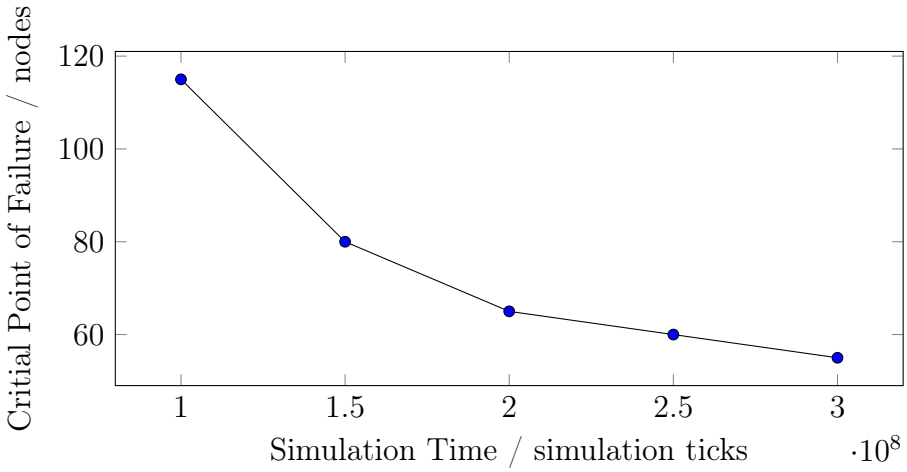


Figure 3.6: Critical point of failure for simulation lengths (UP)

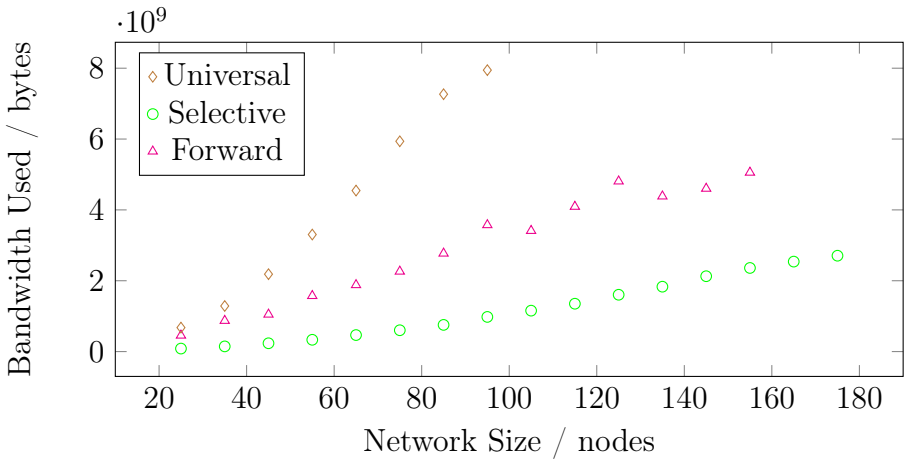
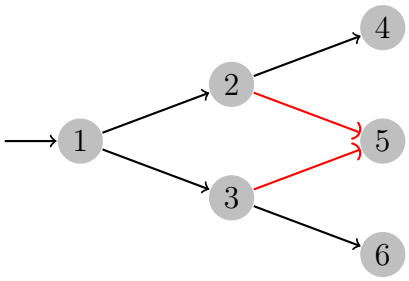


Figure 3.7: Bandwidth used for different broadcast handling methods near steady-state



Node 1 receives a packet, which it forwards to nodes 2 and 3. These both forward to node 5, creating a race condition that wastes bandwidth.

Figure 3.8: Active forwarding wasted bandwidth caused by race condition

This downward trend indicates that the number of nodes needed for network failure decreases the longer the network is active.

Simulations for SP and AF were also conducted, but points of failure could not be determined.

The simulations were run for $300 \cdot 10^6$ ticks, to achieve near steady-state behaviour in the network. The bandwidth used over each of the simulations was compared, as shown in figure 3.7. A possible reason for AF having worse performance than SP is that when multiple nodes forward the same packet to the same node, a race condition is effectively created (figure 3.8).

While optimisations for AF can prevent this race condition by informing any receiving nodes of the forwarded packet ID before the packet contents and allowing only one incoming connection for each unique packet, this has the potential to open up an attack vector for a denial of service where a malicious node could easily forward a genuine packet at a very slow speed, therefore allowing an attacker to potentially deny a node of new packets.

SP has the potential to require the least bandwidth for broadcasts. AF may also be useful to support for the purpose of initial packet distribution, to both increase availability faster and to increase anonymity by hiding the original source of packets.

3.3 API

The primary objective of the system is to provide an API for a send-receive architecture, affording the sending and receipt of messages. The node implementation and abstract protocol are therefore designed around this objective.

The API is designed to revolve around two functions: send and receive. Sending requires knowledge of the public identity of the receiver; the message and how long it should be stored in the network. Receiving requires knowledge of a private identity, which returns a message for the given identity. Both functions are blocking, meaning that sending will block until the message is in the network and receiving will block until a message that has not been received previously is returned. Appendix A.1 shows the general design for the API.

3.4 Abstract Protocol

The UML for the protocol can be found in appendix A.2.

The network is made up of both participating nodes, and non-participating nodes. Participating nodes need to provide some method of creating and handing requests (figure A.4) for:

- The active nodes they know about
- The packets they are in possession of, and the class of these packets
- The acceptable proof-of-work and cryptographic algorithms that are accepted
- The acquisition of a packet given the packet's ID

Each participating node in the network has a psuedoidentifier, which is assumed to be unique, and allows connections to be made to the node, using it as an address. To facilitate the discovery of nodes in the network, nodes are initially provided with at least one psuedoidentifier of a node in the network, allowing an initial point of contact. The acquisition of the initial nodes' psuedoidentifiers occurs out-of-band.

Each node is classified into one of 4 categories: active, where the node is online and responding to requests normally; inactive, where the node appears to be offline; unknown, where the node is yet to be tested for activity; undiscovered, where the node has not been discovered or referenced. All newly discovered nodes are considered unknown and tested to reclassify them as either active or inactive. Active nodes can then be used for connections, creating multiple unidirectional links to the network. Should an active node appear to be inactive by not responding appropriately, it shall be reclassified as inactive. Inactive nodes are periodically checked for signs of activity and reclassified if necessary.

Three packet classes are used in the protocol: meta packets, for the exchange of internal information between nodes; data packets, for the representation of large messages; and broadcast packets, to represent small messages, or to act as a notification of the existence of a data packet.

In order to achieve predictable availability in the network, the lifetime of packets is controlled through the use of time-sensitive proof-of-works. A packet is only considered valid if also accompanied by a valid proof-of-work and packets should only be processed or stored if they are valid. Distribution for broadcast packets is universal, but the maximum broadcast size is small[5] and implementations may required a more difficult proof-of-work for broadcasts. Data packets, with a larger maximum size, are not universally distributed, but rather, distributed to some subset of nodes in the network, referred to as rendezvous nodes. These rendezvous nodes can then be listed in a notification packet.

Packet distribution in the network is accomplished mainly with selective polling (figure A.6) to determining packet ownership, with requests to download new packets. Packet forwarding (figure A.7) is also permitted, and any valid packet that is forwarded should be stored. While not a requirement of the protocol, it is allowable to use selective polling with download requests to acquire arbitrary data packets. As the main distribution mechanism for data packets is forwarding from the source node, this creates somewhat of a network-hard proof-of-work requirement for the sender, where the availability (as a function of the nodes that own the packet) increases with bandwidth used.

[5]Implementation detail.

If utilising the send-receive architecture of the network, the insertion of a valid data packet into the network will also require the insertion of at least one valid broadcast packet, whose purpose will be to inform the recipient of the ID of the data packet, any associated encryption key and the identity of the rendezvous nodes on which the data packet is known to exist. Multiple recipients can be achieved by inserting a broadcast packet for each recipient.

Packet receipt is attained through the acquisition and inspection of all valid broadcast packets. Any broadcast determined to be for the node is used for the message within the packet, or to determine the location of a referenced data packet. Referenced data packets can be acquired by downloading the packet at a rendezvous node specified in the broadcast, or a node that has been observed with selective polling to own the packet.

Streams allow the network to split into smaller subnetworks, with a split occurring once the condition that all streams after a potential split could still be considered anonymous, that is, if their membership still remains high. Membership of a stream is a function of some public value associated with the node. Each stream acts independently for the purpose of broadcast distribution, with cross-stream communications permitted for the insertion of a packet into the stream where the recipient is known to reside.

4 Implementation and Testing

4.1 Node Implementation

The implementation for the protocol has been named Stor, providing the node to handle the back-end protocol communications and the front-end API which serves as an interface to other applications that abstracts details of the network. The UML for the nodes can be found in appendix A.3.

Stor nodes have several management modules that each control different elements of behaviour.

Modules

Connection Module
: A connections module is provided to handle internodal connections. Any observed node is classed as either active, inactive or unknown. Any newly observed nodes are immediately classed as unknown and will be queued for a test connection to determine their activity status. Should a successful connection be established, the node is reclassified as active, else inactive. The connections module attempts to retain a list of at least 20 active nodes at all times, which can then be used for forwarding and polling of packets. This list is periodically renewed from all nodes known to be active, allowing other nodes to be included in the network. Any active node that cannot be connected to, or fails to respond to a request is reclassified as inactive. Inactive nodes are also randomly tested to check for activity and reclassified appropriately.

Proof-of-Work Module
: The proof-of-work module determines which algorithm should be used when generating any proof-of-work. It does this by identifying the nodes any packet will be forwarded to upon insertion and comparing the difficulty required to its own acceptable difficulty. The most common algorithm where the difficulty is not more than $1.5\times$ any accepted difficulty is used for generation. An example of this behaviour is shown in figure 4.1.

Insertion Module
: To insert some message into the network after the API has applied the relevant function, the packet class is concatenated with the applied message. This structure can then be hashed to produce a checksum that is used in the generation of the proof-of-work. A packet is then made by concatenating the proof-of-work with the applied message. This is forwarded to the connected nodes, and will be further distributed through polling and requesting.

Retrieval Module
: A list of packets that should be acquired is kept and the retrieval module works through these looking up any nodes that are known to be in possession of these. Once a node for a desired packet is found, the packet is downloaded from it, checked for validity, and stored.

Polling Module
: To determine which packets are stored on other nodes, selective polling is used to enumerate packets and their classes. Each node has a table of nodes and

the last time they were polled. This timestamp is passed to the node to be polled in the URL of the HTTP request. Upon a successful poll, the table of packets and which nodes possess them is updated. The node works through the table, looking for broadcast packets that it does not have and attempts to acquire them. Data packets are only obtained through being forwarded to the node, although may also be acquired if referenced by a broadcast.

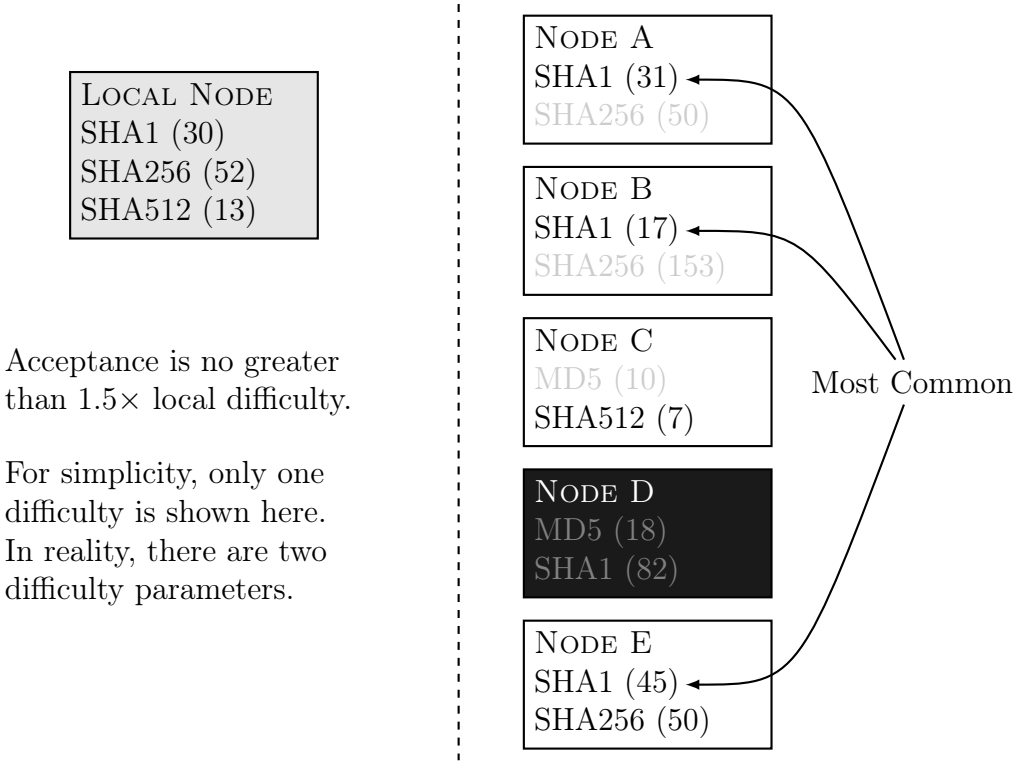


Figure 4.1: Example showing the effect of local proof-of-work difficulty on node selection

RESTful Interface

Nodal communications support a RESTful interface using HTTP for transfers. Each node will be accessible through a GET request to the following pages:

/ Information Page
: Displays supported proof-of-work algorithms and their associated difficulty.

/**poll**/***timestamp*** Packet Polling Page
: Displays the packets stored and their class.

/**nodes** Nodes Page
: Displays the nodes that are known to be active.

/**data**/***packetHash*** Packet Download
: Downloads the specified packet.

POST requests to /data are also allowable, enabling packet forwarding.

Communications require the serialisation of some data structures to pass them between nodes. As different serialisation techniques have differing advantages and disadvantages, enforcing all nodes to support just a single technique was deemed

a poor choice. Rather, any serialisation technique can be supported if later implemented. To determine the technique to use, the HTTP *Accept* header is used, enumerating the types of data that the requester is willing to accept. Any common technique between the 2 nodes is used. Should there be no common techniques, a HTTP *Unacceptable* error code is set.

Persistence

HSQLDB, an embedded database, is used to provide persistence across executions. HSQLDB allows local storage without the running a daemon and hence require no setup from the user. Nodal activity observations will be kept in an attempt to determine which nodes are likely to be active on startup, giving quick access to the network. Records of when nodes were lasted polled are also stored in order to assist in saving resources. Any packets held also persist to increase availability.

Ideally, any data held as part of the persistence would be stored encrypted to prevent an adversary with disk access from determining any prior usage. HSQLDB provides database encryption, which uses a user-supplied password on node startup, if desired. Unfortunately, the encryption method used is somewhat weak in that patterns can be determined in the database, such as fields that contain the same values. While not ideal, HSQLDB's built-in encryption has been used to provide some protection.

Hashes and Proof-of-Works

Hashing algorithms have quite a history of becoming dangerously obsolete. As such, it was felt that supporting multiple hashing algorithms was a wise decision in comparison to forcing a single algorithm upon all applications. Stor currently allows all algorithms that come with the standard Java cryptography provider, although adding more in the future is easily done.

Like hashing algorithms, nodes also support multiple proof-of-work algorithms, allowing for future flexibility. Each node has a collection of acceptable proof-of-work algorithms along with the relevant difficulty parameters. These difficulty parameters consist of a constant and coefficient term, allowing the difficulty to by computed by equation 4.1.

$$\text{difficulty} = \text{constant} + \text{size} \times \text{period} \times \text{coefficient} \tag{4.1}$$

Proof-of-work algorithms, such as that described in Hashcash, use an exponential approach in order to achieve difficulty. In Hashcash, the difficulty is equal to the number of nibbles[6] at the start of the proof-of-work checksum that are zero, and hence the amount of work required as a function of the difficulty size is exponential. This exponential nature is well-suited for systems that require a change in difficulty over time, where computing power is also expected to be exponential. For Stor, this does not follow, as the difficulty needs to be more flexible, and the amount of work to produce a proof-of-work should ideally be linear with respect to difficulty. To achieve this, the difficulty, which can take the form of any natural number, is

[6] A nibble is equivalent to half a byte.

transformed to a number in some finite interval with equation 4.2. The proof-of-work is valid if the relevant proof-of-work checksum is less than or equal to this transformed difficulty (equation 4.3).

$$\text{difficulty}' = M - \frac{\text{difficulty}}{M} \qquad M = \text{ maximum} \tag{4.2}$$

$$\text{Hash}(checksum, period, size, nonce) \leq \text{difficulty}' \tag{4.3}$$

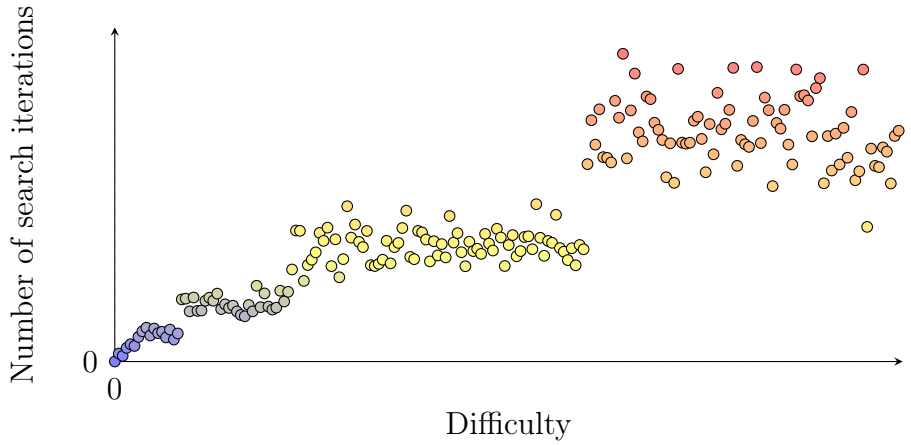


Figure 4.2: Step (exponential) proof-of-work

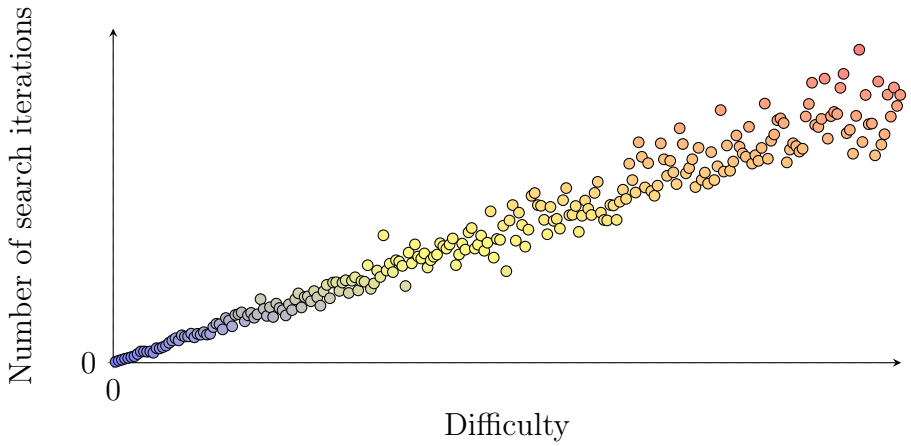


Figure 4.3: Linear proof-of-work

Packets

The packets are as described in the protocol, but no specific implementation of the meta packet is used.

Both data and broadcast packets are equipped with class flags, allowing the class of a packet to be determined from a single field. In a data packet, this has been

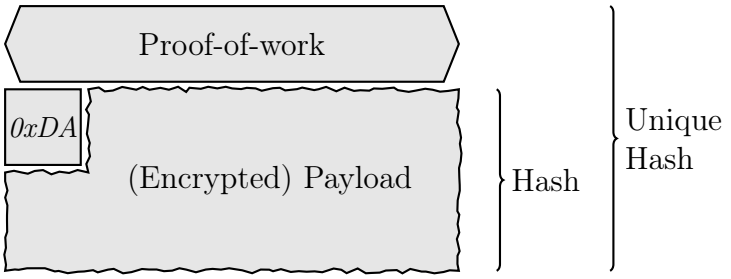


Figure 4.4: High-level data packet structure example showing *0xDA* class flag

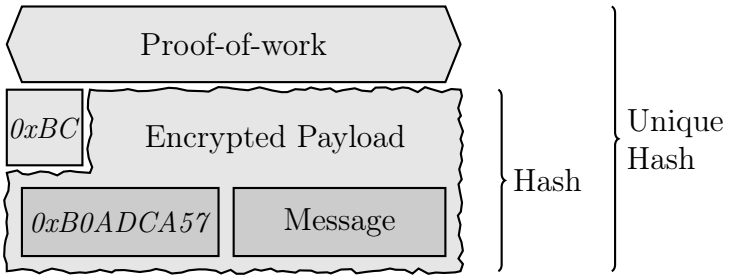


Figure 4.5: High-level broadcast packet structure example showing *0xBC* class flag and *0xB0ADCA57* decryption magic number

selected to be the byte *0xDA*, while with broadcasts, it is *0xBC*. The proof-of-work uses the hash of the class flag concatenated with the payload for validity. Each packet has a unique identifier, which serves as a hash of the entire packet, including the proof-of-work header. Figures 4.4 and 4.5 show the high-level structures of the data and broadcast packet.

Security

Care has been taken to prevent leakage of sensitive information, such as anything that could be used to help determine the geographical location of a node. For instance, HTTP headers often contain a Date field which also contains time zone information. As part of the HTTP standard, this is always GMT, so does not require special attention. However, as timestamps are used elsewhere in the application for the packet validity periods, care has been taken to ensure that these all follow the same time zone, GMT. In addition to time zone leakage, versioning leakage has been avoided by not advertising any version number.

While the node implementation offers some cryptographic method; no ciphers, padding or hashing algorithms have been invented or implemented. Rather, existing, trusted implementations have been used. Considerations were made for the legality around the export of a product containing cryptography and later determined that as long as all cryptography was linked externally from the application, no export license was required. As such, Java's standard cryptography provider has been used, although Bouncycastle was also considered as it offers a wider and freer range of algorithms.

In several instances, there has been a desire to apply serialisation to polymorphism, therefore allowing the subclasses of some supertype to be serialised and deserialsed transparently. While this could have been implemented in order to achieve

a more elegant structure to the code in some areas, such as the `ProofOfWork` class, polymorphic serialisation requires the construction of arbitrary objects, and if done incorrectly, presents a high potential for vulnerabilities. In fact, this is the exact reason as to why polymorphic serialisation is not supported by the GSON library by default.

Tor Interface

The interface to Tor was the first element to be implemented due to the importance of its role in the project.

This module communicates with a Tor instance in order to create hidden services programmatically. Bundling a Tor instance with Stor nodes was considered, but given the complications regarding keeping the instance up to date with the latest Tor releases when included in a package and potential conflicts arising from running multiple instances on a single system, it was decided that responsibility of running an instance should be delegated beyond the API for now.

Tor provides an option to enable a control port, allowing the post-startup configuration. This control port is only bound locally and can use authentication to ensure any connections are legitimate. This interface module uses the Tor controller from `net.freehaven.tor.control` to connect the control port of a Tor instance. The module allows for password authentication with the control port using the password in the *stor.cfg* configuration file.

API

The API starts an instance of a Stor node and when commanded to receive a packet, will create a hook for a private identity, which is hooked into the broadcast feed, allowing it to process all broadcast packets.

The API contains abstract classes `PublicIdentity` and `PrivateIdentity`, which require implementations of the function to apply and inverse. This abstraction makes it easy to provide alternative identities with custom representations and cryptography. It is hoped that through this abstraction, it may in fact be possible to directly hook an application's existing identity scheme straight into Stor's API. For basic usability, an implementation has been provided, using protocol buffers for data representation and RSA/ECB/PKCS1 for the cryptography. The magic number *0xB0ADCA57* is used as part of the serialisation of the plaintext to assist detection of successfully decrypted packets.

The sending of fake packets to add to anonymity has not been implemented in the node, but rather, has been left as a usage detail, where some applications may not desire this (due to added resource costs). Using this feature from the API is as simple as inserting some random payload with a valid proof-of-work header.

4.2 Licensing

As one of the core aims of the project is to release the code for Stor and its API under a BSD license, software licensing must be taken into account to not only

ensure that anyone wishing to use the code can do so, but also to ensure that any third party code used is licensed to allow its use within this project.

External Libraries

`net.freehaven.tor.control` **BSD License**
Provides a high-level interface for communicating with Tor's control port.

`org.eclipse.jetty` **Apache 2.0 and Eclipse Public License**
An embedded webserver that allows delegation of request handling.

`org.apache.commons` **Apache 2.0 License**
Provides various collections and utility classes.

`com.google.code.mimeparse` **MIT License**
A utility for parsing mime types for HTTP request acceptance.

`tinfoil`[7] **(for `TorLib`)** **MIT License**
Provides a method for performing .onion address resolution.

`com.google.gson` **Apache 2.0 License**
Provides serialisation/deserialisation for JSON.

`org.hsqldb` **BSD License**
An embedded SQL database that provides encrypted data persistence.

`com.google.protobuf` **BSD License**
A high-efficiency data structure serialiser.

4.3 Demonstration

To demonstrate the API in action, two applications have been created. Application A, the sender, knows the public identity of application B, and once the API has started the node, prompts for an input which is fed into the API and inserted into the network for B to discover, retrieve and display to the user.

Under most conditions, the demonstration startup time is around 20 seconds, with the message latency after a completed proof-of-work is around a second. In some rare cases, Tor network conditions cause a startup time of up to 20 minutes, although this is the time to get the hidden service descriptor to propagate, and is a one-time cost, not required for additional startups.

4.4 Testing

JUnit tests have been utilised to provide automated unit testing for individual components of the system, taking into account the internal workings of these elements. These have come in the form of both fine-grained and coarse-grained tests. Some of these tests are shown in appendix B.1.

Difficulties with testing encryption were met when building unit tests because all encryption performed as part of Stor uses some form of padding or initialisation vectors, creating a different ciphertext on each execution. Because of this, the assumption that the cryptography provider used is working as expected is made,

[7] `tinfoil` does not provide a fully qualified package name.

although tests where some known value is encrypted then decrypted are performed to test code logic.

As Stor is highly interactive through interfacing with the API, Tor and other Stor nodes, some elements are difficult to automatically test. Testing these elements has been achieved through manual usability tests. These tests focus on the usability and overall functionality of the system, and have been performed at distinct points through the development. These tests are shown in appendix B.2.

As part of the Stor startup process, self-tests are performed to ensure that some modules are working as expected. This is true for critical elements that cannot under normal operation fail, such as the database, webserver, Tor interface and serialiser. Perhaps the most exposed example of this is the test where a node will check that it's hidden service is publicly accessible by performing a connection to itself.

Leakage of .onion Addresses

During testing, it was discovered that metadata was leaking through the form of hidden service DNS requests. This kind of problem is described in [14], where it is shown to be quite a widespread issue. For Stor, the leakage occurred because when using `URLConnection`s, Java does not use any specified SOCKS proxy for address resolution, hence having the potential to leak information about which nodes are being connected to under some conditions.

To resolve this, hidden service address resolution is handled separately to the proxy. The responsibility for this is partially handled by `tinfoil.TorLib`, allowing Stor to pass a hidden service address to tor's resolver, which returns a local IP address that can then be used to make connections to the hidden service in the future without having to specify the hidden service address. This method ensures that no leakage occurs.

5 Review and Future Work

5.1 Management

For the first half of the project, during research, weekly meetings were complimented with reports detailing progress, problems and future targets. In the second half of the project, mainly during the implementation, meetings were generally replaced with email communication expressing progress and targets.

Prior to extensive design work, an enumeration of the project elements was drafted and each briefly assessed and ranked for criticality, identifying tasks deemed critical for project success. Where possible, tasks assigned a high criticality index were approached first to ensure any issues with these tasks could be addressed with ample time.

Throughout the project, multiple regular backups, including versioning with Git have been made to off-site locations, providing several layers of redundancy. Work undertaken has been detailed in the log book in addition to weekly reports.

Appendix C shows the Gantt charts detailing the predicted and realised time plans in addition to the risk assessment undertaken at the beginning of the project. The main differences in these time plans are due to the decision to simulate different network topologies mid-project. Even with this additional task, the project's scheduling has not been problematic.

5.2 Critical Evaluation

The project has been quite successful in achieving the original goals: a framework for sending and receiving messages anonymously and asynchronously has been designed, implemented and released under a FOSS license.

In comparison to similar systems, Stor provides similar functionality to Bitmessage, while adding flexibility and the key feature of keeping users anonymous. The main advantages over Freenet are the use of a large anonymity layer and the ability to use the network for sending and receiving messages, as opposed to storing and retrieving.

As with any product making claims about security, it is necessary for independent audit to investigate the source code for any potential issues.

Several features that have been investigated in the design were not implemented. Streams were originally intended to be supported, but due to the security considerations of having a small network initially, where there is high potential for isolation attacks, this will be left for an extension as the network grows. Zeroisation of keys was identified and investigated for the node/API implementation, but not achieved due to technical limitations both with Java and some libraries included. While some zeroisation in some circumstances may occur, Stor makes no guarantees.

Future work could address some of these unimplemented features and also further investigate issues with trust in the network. Currently, the network is trustless, meaning that a node not following the protocol, such as not storing forwarded packets, is trusted as much as any other node, which is not ideal.

Management of the project has been successful, through the use of short-term targets while taking into account longer-term goals, there has been no point where the project has fallen critically behind schedule.

In the future, some work could be done to utilise the Stor API to create an application that would benefit from anonymous asynchronous communications. Ultimately, while Stor in its current form requires some attention to fine-tune the behaviour, potential for usability in applications remains high.

References

[1] Adam Back et al. Hashcash-a denial of service counter-measure, 2002.

[2] Bitcoin message service. https://btcmsg.wordpress.com/. Accessed: 2014-11-27.

[3] Nikita Borisov. Computational puzzles as sybil defenses. In *Peer-to-Peer Computing, 2006. P2P 2006. Sixth IEEE International Conference on*, pages 171–176. IEEE, 2006.

[4] Thibault Cholez, Isabelle Chrisment, and Olivier Festor. Evaluation of sybil attacks protection schemes in kad. In *Scalability of Networks and Services*, pages 70–82. Springer, 2009.

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

[6] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.

[7] John R Douceur. The sybil attack. In *Peer-to-peer Systems*, pages 251–260. Springer, 2002.

[8] Network Working Group et al. Openpgp message format. Technical report, RFC 4880, 2007.

[9] Morgen E Peck. The bitcoin arms race is on! *Spectrum, IEEE*, 50(6):11–13, 2013.

[10] Baptiste Pretre. Attacks on peer-to-peer networks. *Dept. of Computer Science Swiss Federal Institute of Technology (ETH) Zurich Autumn*, 2005.

[11] Tor Project. Tor rendezvous specification. https://gitweb.torproject.org/torspec.git/tree/rend-spec.txt.

[12] Bruce Schneier. *Applied Cryptography*. Wiley, 1996. Chapters 3, 10.

[13] Someone is crawling torhs directories: Honeypot. https://lists.torproject.org/pipermail/tor-talk/2014-September/034751.html.

[14] Matthew Thomas and Aziz Mohaisen. Measuring the leakage of onion at the root: A measurement of tor's .onion pseudo-tld in the global domain name system. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, WPES '14, pages 173–180, New York, NY, USA, 2014. ACM.

[15] John Tromp. Cuckoo cycle: a graph-theoretic proof-of-work system, October 2014.

[16] J. Warren. Bitmessage: A peet-to-peer message authentication and delivery system.

[17] Charles V. Wright, Lucas Ballard, Scott E. Coull, Fabian Monrose, and Gerald M. Masson. Uncovering spoken phrases in encrypted voice over ip conversations. *ACM Trans. Inf. Syst. Secur.*, 13(4):35:1–35:30, December 2010.

A UML

A.1 API

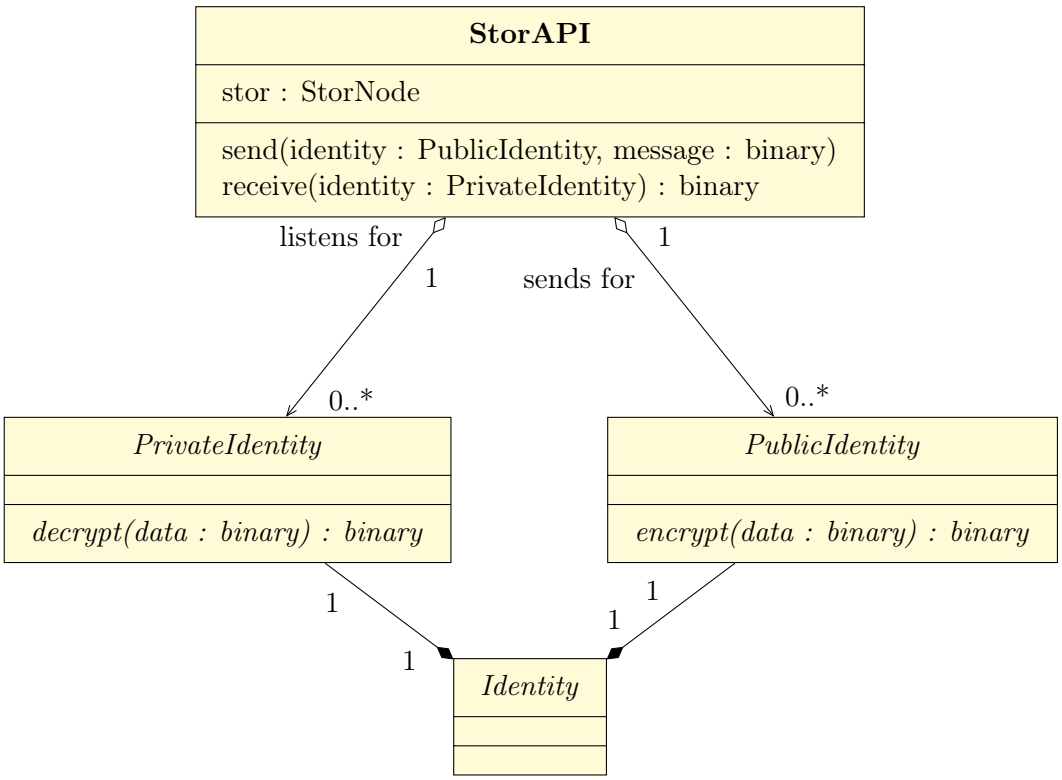


Figure A.1: API Class Diagram

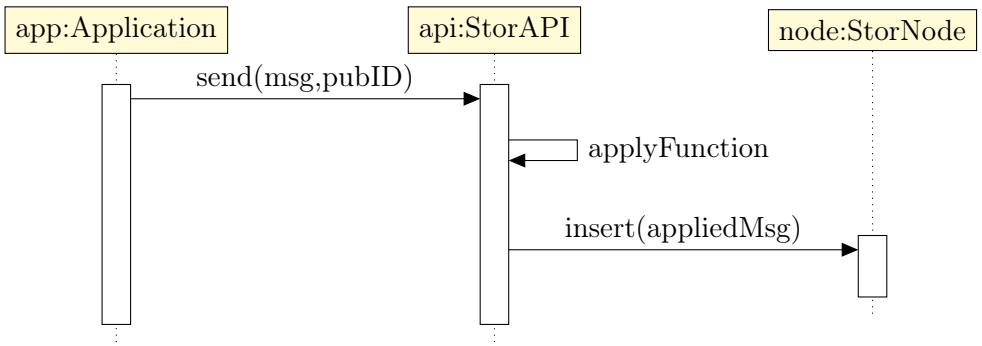


Figure A.2: API Send Sequence Diagram

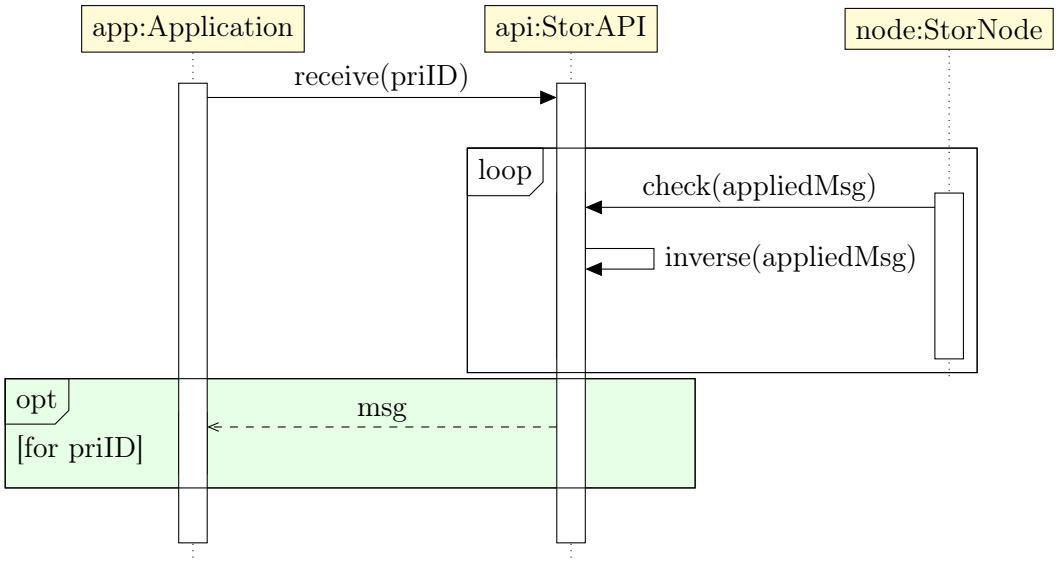


Figure A.3: API Receive Sequence Diagram

A.2 Protocol

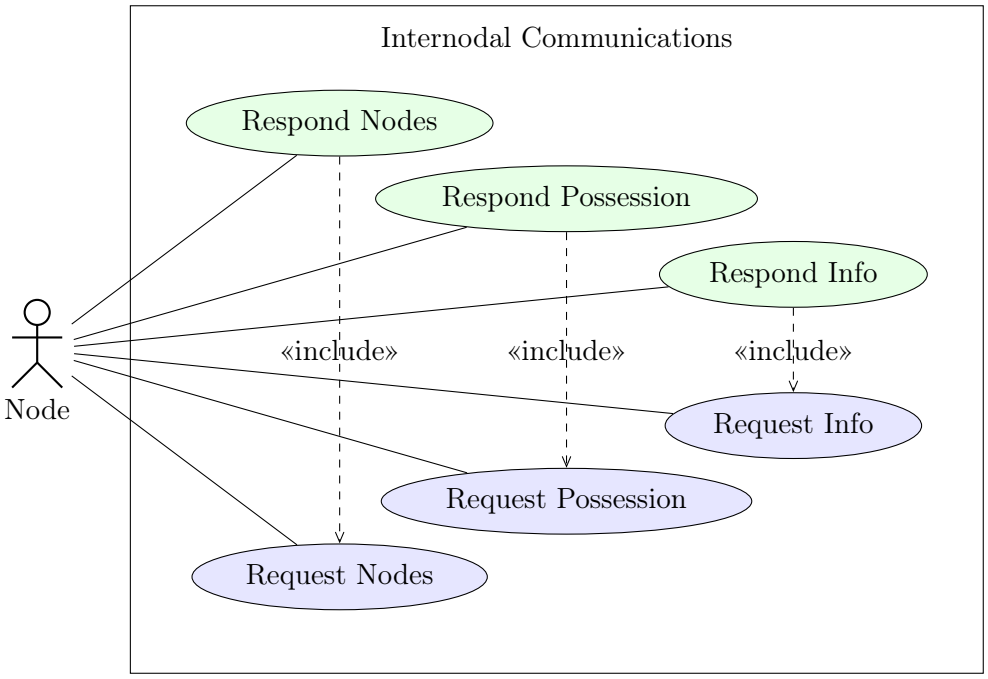


Figure A.4: Protocol's Internodal Communications Use Case Diagram

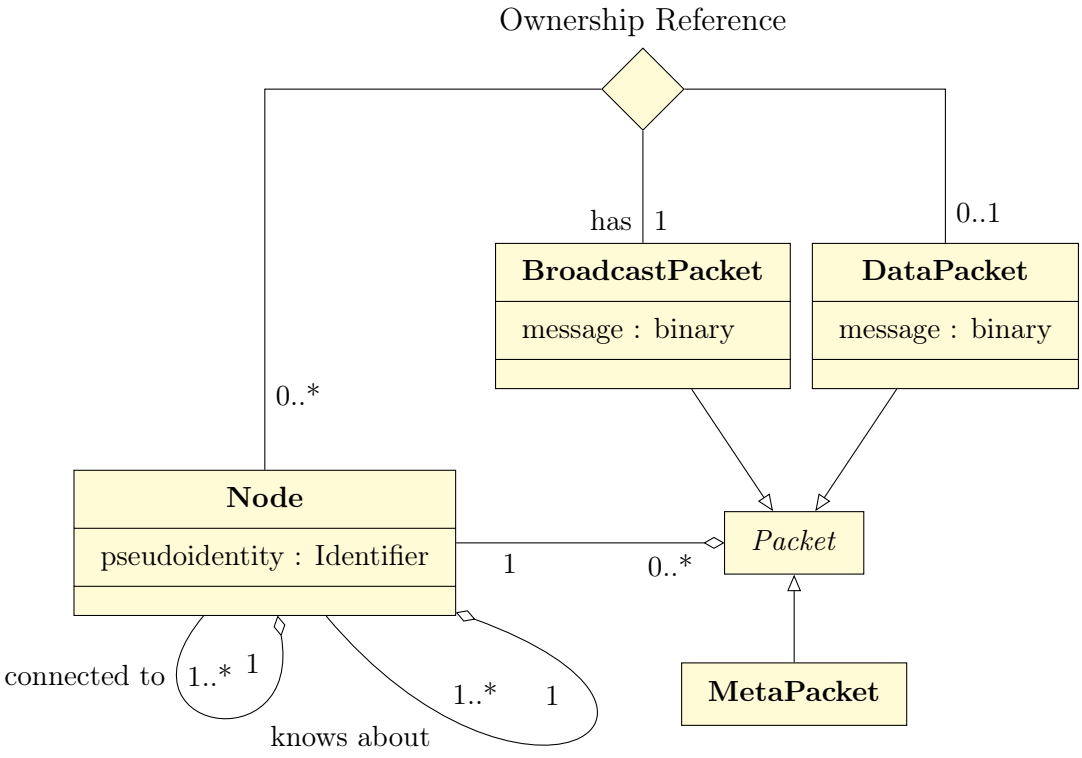


Figure A.5: Protocol Class Diagram

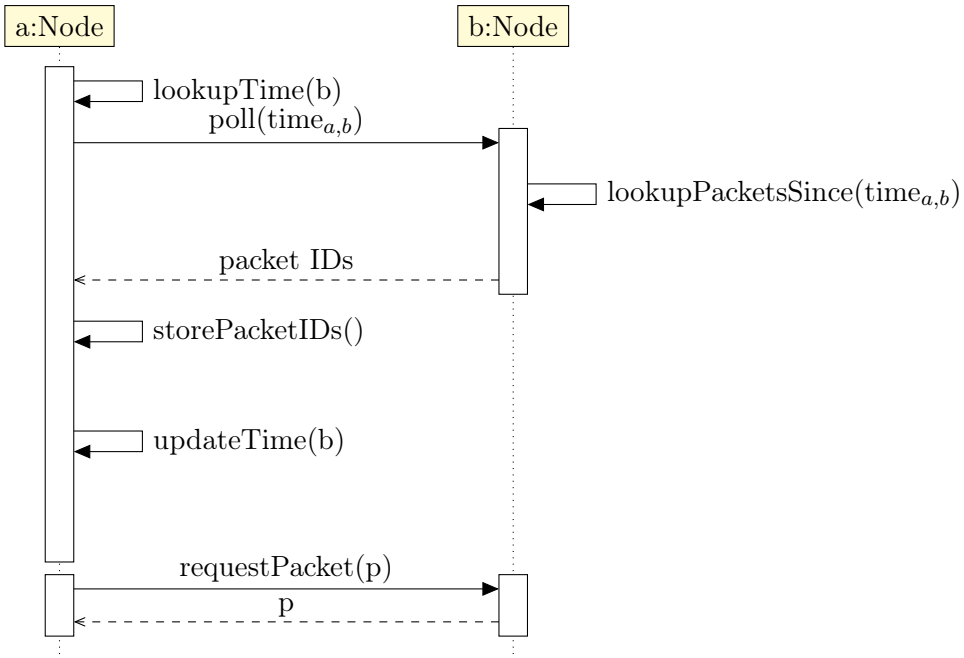


Figure A.6: Protocol Selective Polling Sequence Diagram

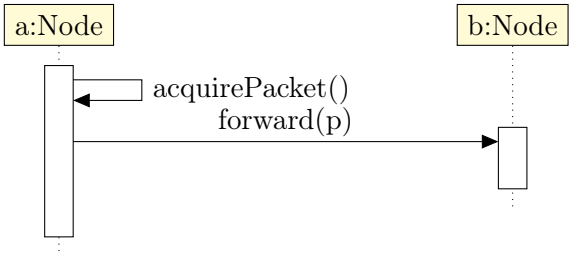


Figure A.7: Protocol Active Forwarding Sequence Diagram

A.3 Node

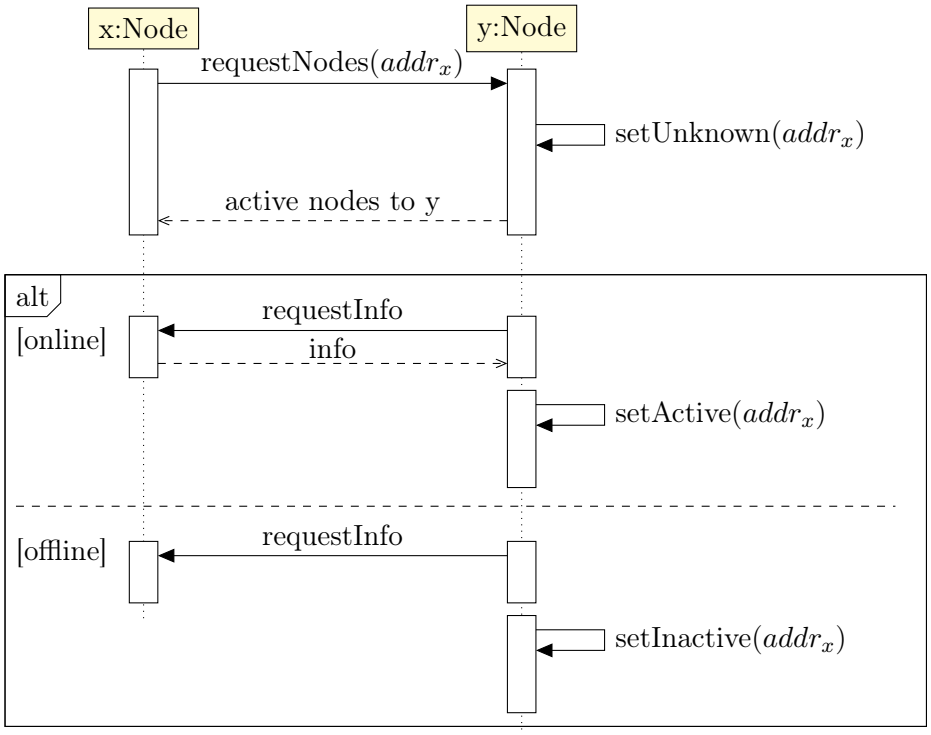


Figure A.8: Node Discovery Sequence Diagram

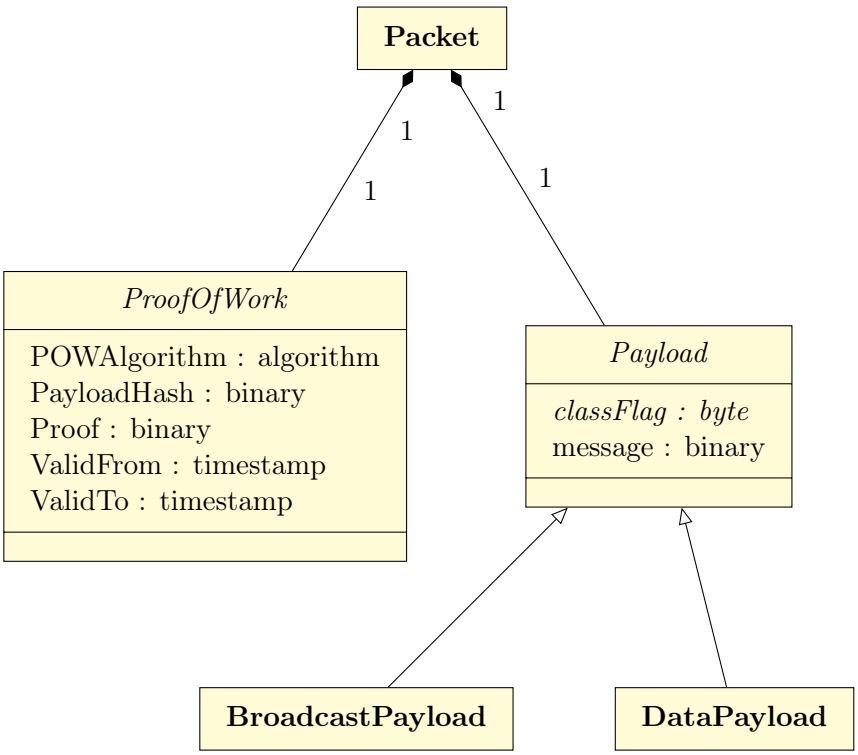


Figure A.9: Node Packet Class Structure

B Test Plan

B.1 Unit Tests

Unit Test	Description	Pass/Fail
`testGeneration`	Tests that a valid proof-of-work is generated so that it would be accepted.	Pass
`testFakeValidity`	A fake proof-of-work validity period is tested to ensure that it is not accepted.	Pass
`testFakeData`	A fake proof-of-work packet hash is tested to ensure that it is not accepted.	Pass
`testFakeProof`	Tests a fake proof-of-work nonce to ensure that it is not accepted.	Pass
`testCheckForMe`	Tests the method that checks for packet receipt on a packet that is for a held identity	Pass
`testCreationFromBase64`	Checks that identities can be created from a Base64 string	Pass
`testSuccessfulDecryption`	A packet is encrypted and decrypted, checking that the message is unchanged	Pass
`testCheckNotForMe`	Tests the method that checks for packet receipt on a packet that is not for a held identity	Pass
`testBroadcastGeneration`	Tests the protobuf for the serialisation/deserialisation of broadcasts	Pass
`testInformation`	Tests the JSON serialisation/deserialisation of nodal information	Pass
`testNodeDiscoveryInformation`	Tests the serialisation/deserialisation of the internodal communications for exchanging node discovery information	Pass
`testParams`	Tests the serialisation/deserialisation for exchanging proof-of-work parameters	Pass

B.2 Manual Usability Tests

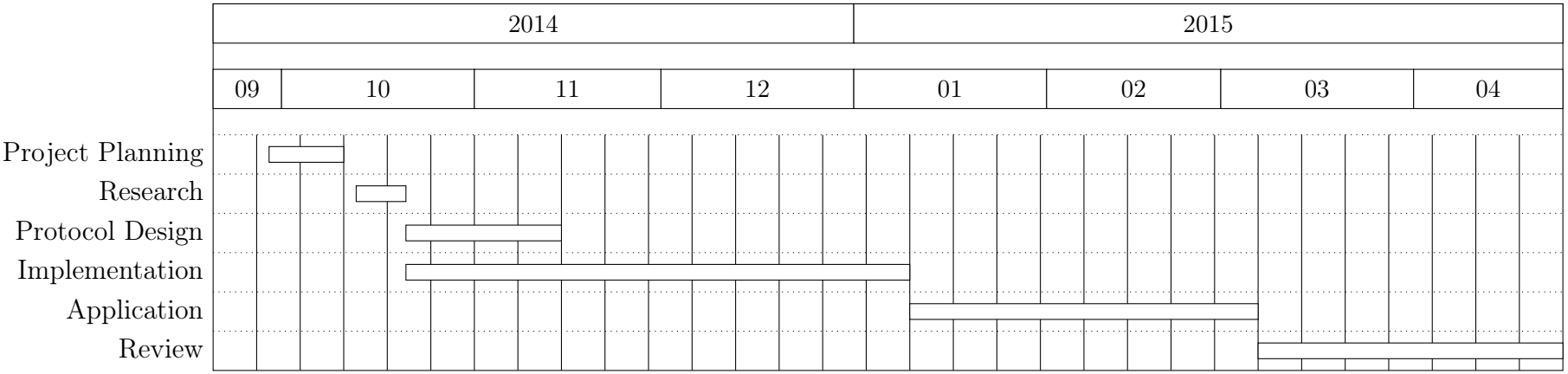
ID	Description	Expected Result	Possible Failures	Pass/Fail
0	Node reports failure if unable to connect to a local tor control connection.	Stor throws an exception, mentioning that a connection could not be made and explaining how to fix it.	Case not checked. Exception not thrown.	Pass
1	New node creates a hidden service	Stor finds no existing hidden service and creates one using the control connection. Can be tested by establishing a connection to the hidden service through Tor.	Failure to detect no existing hidden service exists. Hidden service is not created.	Pass
2	Read configuration and use parameters	Stor uses the configuration file to establish a secure control connection to the local tor instance.	Does not read config. Does not use values in config.	Pass
3	Web interface accessible	Requests are made through the locally bound interface and through the hidden service interface. Responses are expected on both.	Does not bind to an interface. Request not handled.	Pass
4	Valid selective polling response	Packet is given to the node at a known time t and polling requests made before/after this time to ensure the packet does not appear for a request before t, but appears for requests after t.	List of packets is not as expected.	Pass
5	Node accepts (POST) forwarded data	Some data is forwarded to the node to check acceptance.	Request not handled	Pass

ID	Description	Expected Result	Possible Failures	Pass/Fail
6	Node accepts forwarded packet	Node made to forward packet to self.	Packet forwarding/receipt not handled correctly.	Pass
7	Node accepts only valid packets	Invalid packets (invalid PoW, expiry, malformed) are forwarded to the node and not stored.	Node stores invalid packet	Pass
8	Node handles malformed requests gracefully	Malformed serialised forms are sent to the node and a bad request error code returned.	Node throws exception and crashes	Pass
9	Information response	Node is queried about its accepted proof-of-work algorithms and responds with a list.	Request not handled	Pass
10	Node discovery	A node with some knowledge of other active nodes is queried and the list of these nodes is given.	Request not handled Missing nodes	Pass
11	Key zeroisation	Hooks are used to clear any keys from memory upon shutdown or instance finalisation.	Hooks not fired Keys not cleared	Fail
12	Nodes blacklisted after malicious behaviour	Nodes not following the protocol are blacklisted.	Not implemented	Fail
13	API sends	The send function inserts a message into the network.	Message not inserted	Pass
14	API receives	The receive function gets a message for a given identity.	Hook into live packet feed fail Message not received	Pass

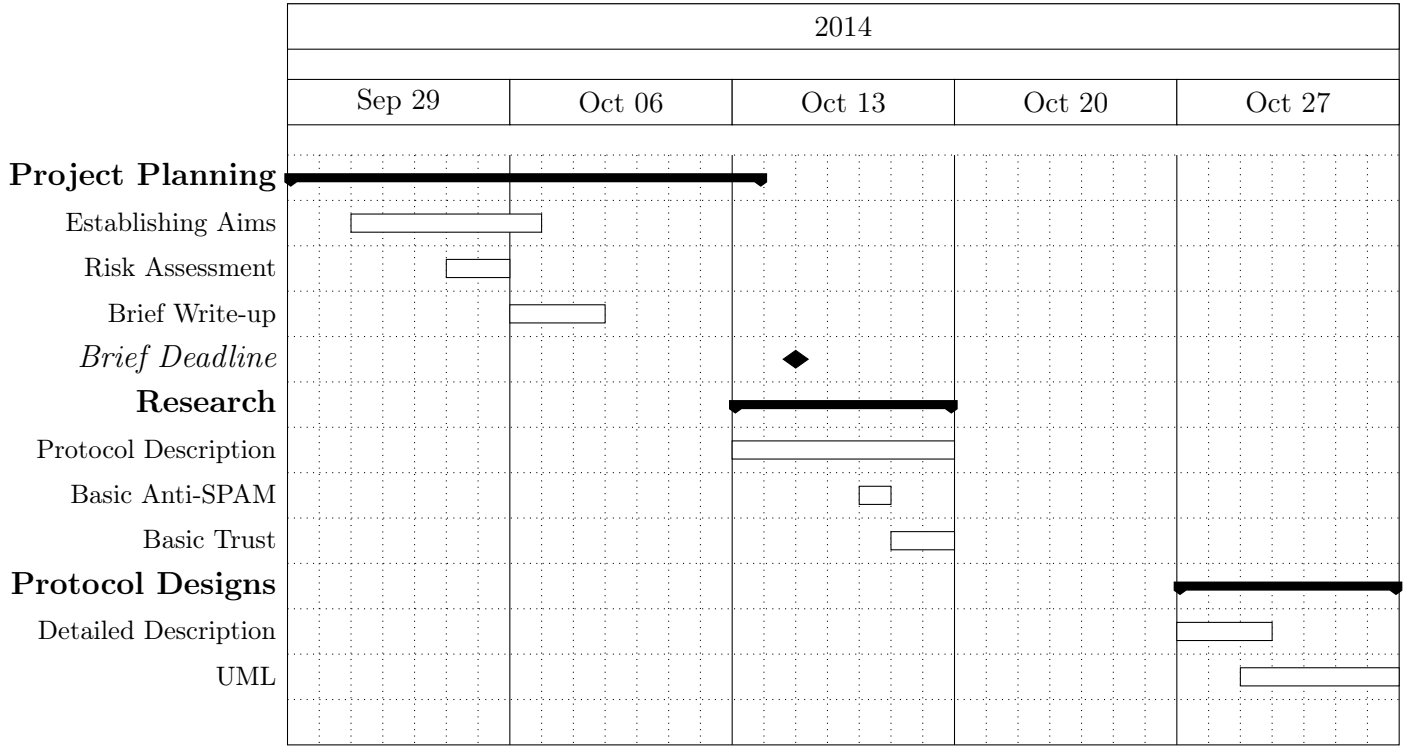
ID	Description	Expected Result	Possible Failures	Pass/Fail
15	.onion addresses resolved locally	Hidden service addresses are resolved locally only.	Resolution is passed through to normal DNS servers	Pass
16	Rejection of misclassified packets	A packet with the wrong class flag is not accepted.	Class flag is not used as a measure of validity	Pass
17	Time zones not leaked	All time zones used should be GMT only, even on foreign systems.	Non-GMT time zones are used	Pass

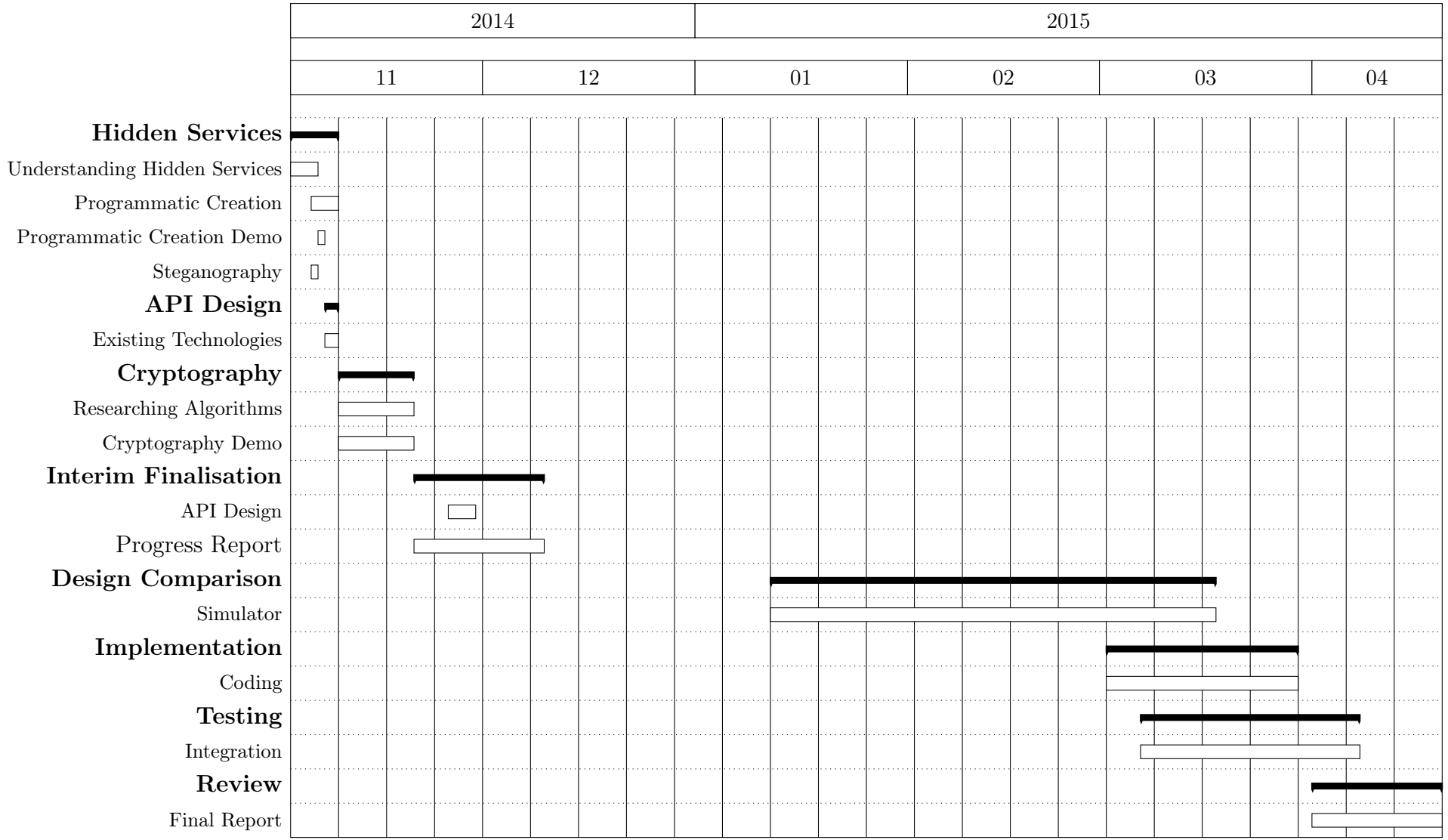
C Management

C.1 Predicted Plan



C.2 Realised Plan




2014
2015
11
12
01
02
03
04
Hidden Services
Understanding Hidden Services
Programmatic Creation
Programmatic Creation Demo
Steganography
API Design
Existing Technologies
Cryptography
Researching Algorithms
Cryptography Demo
Interim Finalisation
API Design
Progress Report
Design Comparison
Simulator
Implementation
Coding
Testing
Integration
Review
Final Report

C.3 Risk Analysis and Contingency Plan

Workflow

Risk	Impact (1-5)	Probability (1-5)	Exposure (1-5)	Mitigation
Data loss	4	1	4	Make regular backups and store in separate physical locations to offset damage from fires/floods etc.
Bug in work	2	3	6	Use source control with regular commits to ensure when something goes wrong, it can be easily rectified.
Development environment failure	2	2	4	Save changes on a regular basis, reducing the amount of damage that can occur from an IDE crash or similar situation.
Network failure	3	2	6	Use another suitable network connection. As a last resort, a mobile data connection could be used.
Temporary computer failure	3	1	3	Use a lab machine or borrow a laptop from the university.
Interfering work	3	2	6	Ensure all deadlines are known in advance and manage time accordingly so all work can be completed to an acceptable standard in plenty of time.
Underestimation of work completion time	3	3	9	Be conservative when planning time for unfamiliar work. Plan periods of time that allow for catching up.
Absent supervisor/examiner	3	1	3	Approach tutor for advice.

Project Content

Risk	Impact (1-5)	Probability (1-5)	Exposure (1-5)	Mitigation
Design changes	2	4	8	Ensure design is as modular as possible, making it easy to change any element.
Project difficulties	3	2	6	Speak to supervisor/tutor about issues
Disagreements with supervisor	2	1	2	Speak to supervisor about issues
Impossible concept	5	1	5	Have redundancy plans in the event that some concepts turn out to be impossible
Tor made unavailable	4	1	4	Plan system around a different anonymity network

Other

Risk	Impact (1-5)	Probability (1-5)	Exposure (1-5)	Mitigation
Family emergency	3	1	3	Stop work for a period of and continue after the emergency. Plan periods of catch-up time to negate any affects.
Health problems	4	1	4	If the issue is not too serious, continue work if possible. Otherwise use catch-up time to negate affects.
Health implications of working with computers	1	2	2	Follow UK HSE guidance on working with computers and VDUs.

D Simulator

In order to assess network metrics to aid design, a simulator was required. Initially, existing tools were explored to find a simulator, but none were found to give the freedom required.

It was decided to write a custom tool for modelling the logic of different networks while also measuring performance. This tool, called ResourceSimulator, revolves around agents and tasks. Each agent has some resources (such as CPU and network). Tasks are assigned to agents, and consume resources for the agents in question. Some tasks, such as in network transfers, are simultaneously assigned to 2 agents and must be performed by both agents at the same time. Tasks may depend on other tasks to complete before they can be started.

ResourceSimulator determines which tasks can be completed by performing a topological sort on the tasks by dependency. Of these tasks available for execution, a FIFO scheduling algorithm determines the largest set of tasks that can be executed given that only one task can consume a resource on an agent at a given instant.

Stimulations are performed at intervals to assign tasks to agents periodically.

E Node Implementation Examples

E.1 RESTful Interface

Information Page - /

```
{
        "address":"of464327gfl63fsh.onion",
        "proofOfWorkAlgorithms":[
        {
                "name":"SHA1-LIN-64K",
                "constant":10000,
                "coefficient":15
        }
        ],
        "maxPacketSize":1048576
}
```

Polling Page - /poll/*x*

```
{
        "validAt":1429557271746,
        "broadcastHashes":[
        "43D860FC9356323DC0BEFDCA06D2B2C3050BC6D4",
        "EE51C458D25652686991588BC678811F828C96BC"
        ],
        "dataHashes":[
        ]
}
```

Nodes Page - /nodes

```
{
        "knownActiveNodes":[
        "storcfvsito5init.onion"
        ]
}
```

Request Page - /data/*y*

```
{
        "pow":{
                "name":"SHA1-LIN-64K",
                "hash":{
                        "hashAlgorithm":"SHA1",
                        "hash":"s+0TIV2a/tOAV9YKY+btF8xm0jc\u003d"
                },
                "proof":"+mUc+ABh9GU\u003d",
                "validFrom":1429269441885,
                "validTo":1429874241885
        },
        "data":{
                "readyData":"vImdg5fsBv..."
        }
}
```

E.2 *stor.cfg* Configuration File

```
# The port to connect to Tor's controller
Port 9100

# The password to auth with Tor's controller
Password myPassword
```

F Project Brief

Problem

Currently, there are few networks that satisfy the property of being both anonymous and asynchronous. These networks typically have weaknesses such as a single point of failure, falling to correlation attacks that can reveal what files users are accessing, or revealing the IP address of users through node harvesting.

Goals

The proposal of this project is to design and implement an anonymous P2P (peer-to-peer) data storage network addressing weaknesses of existing networks. Well-established anonymity networks such as Tor will be used to provide the backbone on which the network will run. The project can be broken down into 3 distinct stages.

Protocol Design

A protocol shall be designed that allows users to communicate anonymously and asynchronously. This protocol should satisfy the set of properties:

1. Users of the network cannot be identified
2. The contents of a packet can only be read by a recipient
3. The send of a packet of data cannot be identified by anoyone except a recipient
4. A recipient of a packet of data cannot be identified by anyone except the sender

API Implementation

The API will allow a high-level interface for sending and receiving data using the network. This API will be designed in such a way that it minimises restrictions on how users can use it.

API Application

An application will be made to demonstrate the API and network in action. As the scope for the application could be quite large, this leaves room for extensions should surplus time be available.