University of BRISTOL

DEPARTMENT OF COMPUTER SCIENCE

Shadow Peer-to-Peer Networks

Luke Murray

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Engineering in the Faculty of Engineering.

Sunday 12th May, 2013, CS-MEng-2013

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of MEng in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Luke Murray, Sunday 12th May, 2013

List of Figures

List of Tables

List of Algorithms

Executive Summary

Anonymity is a widely sought after commodity on the Internet. Many reasons exist for seeking it; from protection from persecution to secret governmental work. To this end, many anonymous networks have been created to provide this. This project aims to explore the limits of anonymity and to design a network that provides more anonymity for its users than any existing solution.

The project introduces the following concepts:

- The 'Shout': A method for sending packet based messages between peers without those peers knowing the true identity of the other peer.
- The 'Shout Group': A method for protecting the anonymity provided by the shout method.
- Public Key Hiding: A cryptographic method for scrambling a public key such that it becomes unrecognisable but retains its usefulness as a public key.
- A Uni-directional Toroidal Network Structure: An arrangement method for positioning nodes within the network giving the network a regular structure.

From these concepts, Shadow P2P is created and finds the following:

- The shout group method provides a high level of protection for the anonymity provided by the shout method which is used for all network communication.
- The public key hiding method is cryptographically secure; the original public key cannot be distinguished from the scrambled version.
- The uni-direction toroidal network provides the sender and receiver of packets with additional anonymity, if the provided source routing is used intelligently.

Contents

Supporting Technologies

- The SciPy library (`http://www.scipy.org/`) was used for statistical analysis.
- The PyPlot library (`http://matplotlib.org/api/pyplot_api.html`) was used to produce the graphs in this thesis.

Notation and Acronyms

IP	:	Internet Protocol
TCP	:	Transmission Control Protocol
UDP	:	User Datagram Protocol
NAT	:	Network Address Translation
TTL	:	Time To Live
SMC	:	Secure Multiparty Computation
Node	:	A distinct computer system
Peer	:	A node that is part of the network
Shout	:	A method of anonymous P2P communication achieved through IP spoofing and broadcasting
Shout list	:	A list of IP addresses broadcasted to by a shout
Virtual Position	:	Coordinates within an abstract view of the network structure
Local Root	:	The virtual position from which a peer inherits all its other virtual positions

Acknowledgements

I would like to thank my supervisor, Dr. Simon Hollis, for his input on the topic and advice on the project's direction. Gratitudes are extened to my flatmates for their understanding that the many hours I spent alone were not signs of me becoming a recluse. Finally, to my friends and family for agreeing that only I could have come up with something so... unique.

Chapter 1

Contextual Background

The Internet is generally seen as a medium through which free speech can propagate faster and spread wider than was previously possible with physical means (e.g. mail, newspaper). Now, more than ever, Internet users can communicate their views, beliefs and opinions unhindered. However, speaking one's mind is not without danger. In some cases, what one says may attract undesirable attention and negative consequences. For instance, making criticisms about a political leader may lead that politician to abuse their power and bring untoward harm to the critic. A trivial solution would be to simply not make such a remark. However, the Internet has made speaking one's mind very easy and usually without consequence through the use of anonymity.

The ability to de-couple the identity of an individual from their views is not unique to the Internet but it is much easier to achieve. If our aforementioned critic were to send a letter by mail with their remarks written on it then the critic can retain anonymity by not stating who the letter is from. On the Internet, the same effect can be achieved in a few key strokes. This abundance of anonymity has given rise to a number of websites, all dedicated to allowing one to express one's views anonymously. In some cases, this has given rise to antisocial behaviour, however, there are other cases where anonymity can be a literal life saver.

When forming connections between parties, the Internet itself does not provide any information about the person who is using that connection; such information must be provided by the user. Thus, by default, the connection is anonymous. What is provided, however, are the Internet Protocol (IP) addresses of the parties forming the connection. This address could be traced back to an institution, household or individual computer and from this it may be possible to deduce the identity of the user. For this reason, this project defines the identity of a user to be their IP address.

To prevent revealing a user's IP address, numerous solutions have been created that keep users anonymous online. Each of these has its own merits and flaws but they do all have something in common. This is that, in every solution, the parties involved always know the identities of the parties that they directly communicate with. In these solutions, it is generally the case that any party that know the identities of the other parties does not have any knowledge of what information that party is sending or receiving (due to confidentiality methods used, mainly encryption). Therefore a party that knows the identity of another party is unable to link a view, belief or opinion back to them. This is generally considered to be a sufficient level of anonymity.

I argue that there is a use for a higher level of anonymity. On this basis, I have created an anonymity solution that keeps the user anonymous by ensuring that no other party knows their identity. This way, an extra level of anonymity can be provided; in my network design, it is only with great difficulty that a party can discern whether or not an internet user is participating in the network. This feature is not provided by any other network.

1.1 A need for anonymity

There is a need for anonymity solutions in Internet society. In some places, the rise of the Internet has allowed its users to circumvent censorship measures present in their country. Where before only the public media needed controlling to prevent the dissemination of information, now any user is able to spread news far and wide. A pro-censorship government would struggle to exert control over all user publishing. Those users responsible for disseminating information that would otherwise be censored are usually subject to litigation. Obviously, those individuals would be keen to avoid the law.

A good example of this is China which has a long history of media censorship and has a well known internet filter dubbed "The Great Firewall of China". The only parties allowed to publish news online are licensed print publishers who already self-regulate their content to comply with the laws of the state. If a user were to publish material or make statements which contravened the censorship policy, they are usually arrested. It takes as little as signing an online petition or communicating with a foreign political group to be imprisoned. Anonymity would then obviously be desirable when engaging in politics online, a person can then express themselves fully without fear of retribution from censors.

Censorship is just one of the reasons for using anonymity software. Take, for instance, apostasy; this is the act of giving up one's religion. There are 14 countries where apostasy is illegal and in 9 of these it is punishable by death. Thus, if a person were to become apostate in one of these countries and wanted to discuss it, they would have to do so anonymously or risk losing their life. It would be wise in such cases to take additional measures against having one's identity discovered, above and beyond that already provided.

There are a whole host of other situations where anonymity is desirable.

- Corporate and government whistle-blowers who don't want to lose their job or face litigation.
- A bank may wish to scout out new investments without their competitors knowing what they're interested in.
- Some online market places have been known to engage in price discrimination. By disguising their identity, a customer can secure the same product at a lower price.
- Law enforcers visiting illicit websites and services in order to collect evidence on them. Without anonymity software, the law enforcer may be shown a different version of the site without illegal content or they may risk alerting the website owners to the surveillance.
- Secret agents obviously do not wish to be discovered in the field. Directly connecting to a military computer in their home country is a dead give-away, even if the content is encrypted.
- Anonymous tips cannot be delivered through a simple website as the web server will record the IP address of anyone visiting. Tippers will need anonymity software to cover up their IP address.
- Individuals may wish to prevent a search engine correlating all their searches. Anonymised records of a user's search terms can lead back to individuals[2].

1.2 Work by others

My project is heavily influenced by existing solutions to the problem. The systems I will describe here are already available and widely used. These systems are The Tor Project[24], I2P[14], Freenet[10], GNUnet[11] and the Gnutella network[12]. The Phantom Protocol[8] also provides some important contributions to the project but its implementation(s) have not been keenly adopted. Virtual Private Networks (VPNs) will be discussed as an existing solution but these networks ultimately lend nothing to the new network design.

There are some non-network related applications that provide some useful ideas to the project. One such application is Bitcoin, the P2P virtual currency, which will help in hindering and preventing attacks on the network.

1.2.1 Virtual Private Networks

VPNs are not peer to peer constructs in most cases. They are collections of computers with many different connections to the Internet, all of which are owned by a single company. The company provides a service to its customers where they may connect through the company's computers to the Internet for some subscription fee. When connecting through a VPN, the IP address of the customer is replaced by that of the VPN company which anonymises the user amongst the client base of the company. The connection between the VPN and the customer is securely encrypted which prevents any activity performed by the VPNs being traced back to an individual customer. The level of anonymity that VPNs grant is dependent entirely on the records that the company keeps of the activity of their customers. If the company kept a full record of all activity and some entity, that is hostile to the network, acquired the information then the anonymity of the customers is rendered void. If the company keeps no such record of activity, then the customers will remain anonymous. Sensible customers must first trust that the company doesn not keep logs before using their services. This requirement of trust in a third party is the common fatal flaw of all VPNs; the Peer-to-Peer (P2P) networks described next have no such requirement.

Despite this downside for anonymity, VPNs generally give customers better performance over P2P networks. It is advantageous for VPN companies to set up their servers with high bandwidth and low latency connections in order to compete with other VPN companies. P2P network peers can appear anywhere on the Internet, giving rise to varying connection quality. Anonymity concerns can be addressed by forming connections through multiple VPNs. If the VPNs are hosted in different countries and the countries are politically hostile with one another then law enforcement in one country is unlikely to be able to forcibly obtain logs from both VPNs.

1.2.2 Tor

The Tor Project[24] is a P2P network originally designed by the U.S. Navy to protect government communications. The software has since been released as open source software and is maintained by numerous volunteers. Users connect to the network by contacting a directory server and retrieving a list of peers. A client can then establish a path through the Tor network (called a 'circuit') by establishing encrypted tunnels through multiple peers. The client may then use this circuit to connect to the Internet itself or to one of Tor's 'hidden services' (a service only available if the user connects to it with the Tor client).

The network operates such that no intermediate peer that the client tunnels through knows what the client is sending / receiving or which two parties the communication is occurring between. This is guaranteed through cryptographic methods and this eliminates the need for the user to trust any party in the network; unlike with VPNs, an entity cannot obtain a log and discern who communicated with whom. Even if the peers did keep logs, the entity would not learn which users performed which actions. Only if all intermediate peers in a circuit are compromised can an adversary learn what actions a given user has performed. As it is the client software that picks the peers in the circuits it uses, the client can avoid picking known hostile peers. As the Tor client picks peers at random and the vast majority of peers are good, the chances of a client making a circuit entirely from a single adversary's peers is infinitesimally small.

Tor's contribution is that of 'onion routing' where layers of encryption are sequentially added or removed by intermediate peers between the sender and the exit node. Whilst it is not the first application to use this technique it is a clear example of good usage.

1.2.3 I2P

I2P[14] or the Invisible Internet Project is a low-level network that constructs secure tunnels between peers. The low-level that the network operates at allows all current internet using applications to communicate over the network as if it were the Internet itself making it very versatile. The tunnels that it constructs are one-way in the sense that a tunnel either transfers packets away from the tunnel's owner (an outbound tunnel) or towards them (an inbound tunnel). In order to send messages between peers, a message is sent from a peer down one of its outbound tunnels. The tunnel exit sends the message to the entrance of one the other peer's inbound tunnels which then forwards the message to the receiving peer. The sending peer must first look up the 'lease' for the peer they wish to connect to in the network database in order to retrieve the address and key material for an inbound tunnel to that peer.

Of interest to this project is the way I2P splits the information about a peer's anonymous identity and its real identity with the use of two separate Distributed Hash Tables (DHTs). Additions to the anonymous database are done through an established outbound tunnel which preserves the anonymity of the peer as it does this. Information about the services that the peer hosts or uses are then unconnected to the identity of that peer. In addition, the outbound traffic and inbound traffic are disassociated from one another; this will make attacks that use traffic analysis much harder to execute therefore this project uses a variant on this idea.

1.2.4 Freenet

Freenet[10] provides distributed data storage and the ability to publish 'Freesites' (websites only visible to the network peers). It is censorship resistant by design; one feature of Freenet is that once some data has been published to the Freenet, the uploader can disconnect and the data is still available to the peers. Another interesting feature is that searching for a piece of data causes it to be spread more widely throughout the network. When a data store replies to a request for some data, all the peers along the data response path store the data as well. Direct deletion of data is not possible, data can only be 'forgotten' over large periods of time if no peers request it. It is interesting to note that checking whether some data has been forgotten results in that data being 'remembered' in the case it is still present in the network. The wait for deletion will then start all over again.

Important to this project is the way information spreads to all peers it passes through rather than just the peer that requested it. This will be useful for this project's network management.

1.2.5 GNUnet

GNUnet[11] is designed for censorship resistant file sharing. It may be used over a number of different methods of transportation between peers including (but not limited to) IP, HTTP and wireless LAN. Essentially, the network abstracts exactly how the data is transmitted away from the applications that use it. The network employs an economic model that aims to prevent attacks on the network and free-loading by having peers rate other peers. Over time the peers build up a view of how much they trust a given peer. When traffic load is high in the network, requests from less trusted peers are dropped, preventing them from doing further harm if they are hostile.

From this network we take the concept of 'cover traffic' upon which GNUnet seems to rely for the majority of its anonymity. A peer may set some threshold and then will only send packets if it is routing enough other traffic through itself in order to disguise its own traffic among the crowd. In my network the function of cover traffic is the same with the added provision of dummy packets so that cover traffic may be generated as required.

1.2.6 Gnutella

Gnutella[12] was originally designed to have all peers on an equal standing. When a request was sent to the network, it would be flood-broadcasted to all other peers in the network. Peers would then respond back along the route the request took with query results (if that peer had any). This method of query and response is clearly unsustainable for larger networks and, due to Gnutella's popularity, it suffered greatly from this flaw. Nowadays, the Gnutella network is composed of leaf nodes and ultra-peers; Leaf nodes connect only to ultra-peers and ultra-peers (which have a much greater available bandwidth) connect to large numbers of other ultra-peers. This way the network doesn't suffer from query packet saturation.

1.2.7 Phantom

The Phantom Protocol[8] describes a tunnel based network design with theoretically secure anonymity properties. It claims to be superior to both Tor and I2P in that it isn't vulnerable to some of the same pitfalls. Unlike Tor and I2P, it aims to be completely segregated from the normal Internet and to be compatible with all existing software using an Internet connections. Whereas Tor forbids large data transfers and dedicates itself mainly to anonymous web browsing, Phantom aims to allow all forms of internet traffic. The tunnel design is obviously similar to that of I2P but it is clear that much more thought has been given to ensuring anonymity throughout tunnel creation and tunnel use.

1.2.8 Bitcoin

Bitcoin[7] is a P2P virtual currency where the units of currency (Bitcoins) are produced at an almost fixed rate. To do this, the network employs a system of proof-of-work; a proof-of-work is a solution to a problem which is hard to solve but easy to verify the solution. In order to produce Bitcoins a peer must perform some proof-of-work before obtaining them. Since it takes a lot of computational effort to produce the proof-of-work, the rate of supply of Bitcoins can be limited. This rarity gives the Bitcoins value and this makes it feasible to use Bitcoins as a currency. The proof-of-work mechanism is further augmented in the network by dynamically changing the threshold at which a proof-of-work will be accepted. As more computational power gets added to the network, the threshold is raised to dynamically in order to return the production of Bitcoins to the predetermined fixed rate.

In this project, the proof-of-work concept is used to make attacks on the network much harder to execute.

1.2.9 IP Spoofing

IP spoofing[23] is a well known technique that allows the sender of a particular IP packet to conceal their identity. This is easily achieved by the sender because the sender is responsible for putting their correct IP address on each and every packet they send. Whilst it is trivial for a packet with a spoofed IP address to be created, having this packet traverse the Internet between sender and receiver is much harder.

Network Address Translation (NAT) is the first hurdle, this is a feature of the sender's router that provides a large number of Local Area Network (LAN) IP addresses so that all machines on the LAN can access the internet via a single IP address provided by the user's ISP. The NAT will replace the source IP address of any packet exiting the LAN with the IP address provided by the ISP. Therefore, a router with NAT enabled will undo the effort of the IP spoofing and will add information about the user's identity to each and every packet. To make things harder, most commercially available routers do not allow the NAT feature to be disabled because there are very few reasons that a user possessing a single IP address would want to do this. However, it is just a matter of acquiring the correct hardware.

Secondly, ISPs perform source address filtering on packets that enter their networks (ingress filtering). The aim of such filtering is to block malicious IP addresses from performing attacks within, or across the ISP's network. These filters are also in place to block spoofed IP addresses. Research into how effective these filters are showed that 31% of the clients performing the testing were able to perform some form of IP address spoofing[6]. Furthermore, some of the same tests were used in a similar research effort 4 years prior[5] where it was estimated that 20% - 30% of the clients could perform IP spoofing leading to the conclusion that uptake of such filtering is very slow. Therefore, whilst this does indicate that 69% of all senders would not be able to perform IP spoofing, the proportion that can will be able to so for some time.

IP spoofing is used in this project to prevent the receiver of an IP packet from learning the identity of the sender.

1.2.10 Metrics of Anonymity

In order to be able to reason about how anonymous something is, a method of measuring anonymity is required. In this project, I use two such metrics.

In [19], the concept of an "Anonymity Set" is introduced. It is stated that "Anonymity is the stronger, the larger the respective anonymity set is and the more evenly distributed the sending or receiving, respectively, of the subjects within that set is". This is a rather intuitive definition of anonymity. By analogy, the larger the crowd a person is in, the more anonymous they are. Also, the more a person acts like one of the crowd, the more anonymous they become.

A more complex metric is introduced in [21]. This represents the anonymity of a subject in terms of entropy. The more entropic a subject's anonymity measure is, the more anonymous they are deemed to be. The entropy values that the metric gives not only indicate how a subject can be anonymous to another subject, but also how anonymous they are compared to the absolute maximum.

1.3 Reaching the limit of anonymity

This project aims to try and make a network design that provides more anonymity to its users than any other network currently provides. Even upon reaching this level, the project aims yet higher. It aims to reach for the boundary of what it is possible to provide in terms of anonymity on the Internet. In creating this network, I look at both networking and cryptographic methods of providing anonymity. Details of their theoretical design and practical implementation will be considered.

This project attempts to create a network where not only is it infeasible to eavesdrop on traffic or determine who is talking to whom but it will also be infeasible (or at least incredibly costly) to determine whether or not a node is participating in the network. This is a feature not provided by any other network design. Whilst it may be of limited use, I have shown that there are scenarios where it is necessary to have such a level of protection. It is therefore important to try to fill this gap in network designs so that there will at least be something of use in these situations.

This idea of 'hard to detect' participation gives rise to a much more intriguing feature of my network design. In all other P2P networks, the peers hold the details required to connect to their neighbours and engage in communication; namely, the peers hold the IP address of their neighbours. Recall that I defined the IP address of a peer to be their identity as this information can easily be traced back to a person or other entity responsible for that IP address. This means that the neighbours of a peer know that peer's identity. This is far from ideal if we wish to protect against entities discovering that a peer is participating in the network. To rectify this flaw, I have designed the network such that *a peer's neighbours do not know that peer's IP address*. To clarify, neighbours do hold the peer's IP address but only amongst many others, thus the peer is anonymised by virtue of being 'lost in the crowd'. As may be obvious, the challenge here is to show exactly how communication can occur between peers who are unsure of their peer's IP address (without which, messages cannot be sent to that peer).

On top of the basic communication links between peers, designing a network around anonymity is another major undertaking. I will use the ideas and proven concepts from the existing networks to construct a network that meets the needs of the project's security and anonymity requirements.

The other P2P networks mentioned previously all employ some variant on the mesh network structure. In such networks, connections and disconnections are generally made between peers as is needed to improve network efficiency, or to perform self-healing. This is inconsistent with the requirement of anonymity; if the network is structured such that peers that exchange more data have a higher chance of being directly linked, then it can be determined who is likely to be talking to whom. In essence, for the greatest possible level of anonymity, the network structure should not bear any relation to the traffic the network handles.

Some networks have mechanisms in place for preventing network structure giving away information. In the case of Tor, packets incur random delays as they pass through peers. This makes it nearly impossible for an adversary being able to tell which packets entering a peer correspond to packets exiting that peer. This is more of a get-around for a single problem case than a full solution. If there is a large throughput of traffic among a linear sequence of peers, it is reasonable to assume that there is a bulk transfer between the peers at the ends of the sequence. Thus there is still information leakage from the structure of the network.

1.4 Project Objectives

In the course of reaching the limit of anonymity, the project works towards the following goals.

1. Conceive methods that allow peers to communicate anonymously.
2. Invent defensive measures against adversaries seeking to identify the anonymous parties.
3. Design a feasible anonymous network design based on the novel techniques.
4. Implement (at least partially) this network design.

Chapter 2

Technical Background

2.1 Shouts

A 'shout' is the method this project uses for anonymous peer to peer communication. By analogy, it is as if the sender were shouting to a street full of people from behind a wall. The sender cannot be seen by those on the street and those on the street all appear to ignore what was shouted. As the receiver acts identically to the rest of the people on the street, the true receiver cannot be determined. To achieve this, the sender is anonymised through the use of IP spoofing (removing the sender's address from packets they send) and the receiver is anonymised through a broadcast to a list of multiple addresses (a 'shout list'). This method can be used regardless of the choice of IP version (i.e. IPv4 or IPv6).

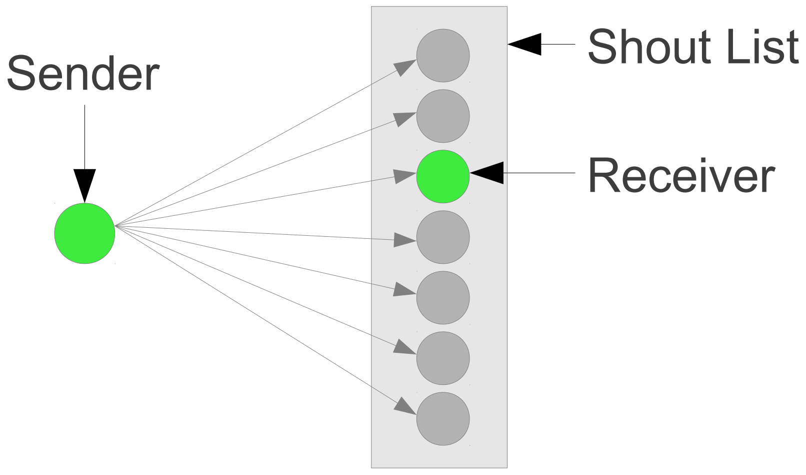

Figure 2.1: To shout, the sender sends a packet with its source IP address spoofed to every IP address in the shout list.

The act of 'shouting' is defined to be sending packets with a spoofed source IP address to every IP address in the shout list (see Figure 2.1). A 'shout' is all of the packets sent when shouting a single time. In order to communicate directly with a peer, it is necessary and sufficient to obtain that peer's shout list.

The shout is designed to work against an adversary that aims to identify network peers from the nodes of an Internet. The adversary aims to do this by monitoring the traffic passing through the links connecting the nodes. It is limited in the number of links it can monitor simultaneously; if the adversary could monitor every link at the same time then it can easily spot nodes that are network peers.

2.1.1 Shout Lists

A shout list is a list of IP addresses, one or more of which corresponds to network peers. All the other addresses are randomly selected from the set of all IP addresses. This list is created by the receiver and given to the sender. This allows for uni-directional communication from sender to receiver. If the sender wishes to deliver a packet, they send identical packets to all the IP addresses in the shout list. The receiver should receive one of these, however it is not guaranteed that they do as the method of packet delivery is unreliable. Specifically, packets are sent at a lower level using the User Datagram Protocol (UDP). Obviously, this method of message delivery is very inefficient, however, recall that anonymity is our main concern here and that any attempts at efficiency are secondary.

The receiver that creates the shout list needs to select a large enough set of IP addresses such that monitoring all of the links leading to every destination IP address is infeasible for the adversary. If the adversary obtains the shout list, they know that at least one of the destination IP addresses belongs to a peer and they can then focus solely on traffic to those addresses. The adversary may see packets traversing a link that are going to one of the shout list addresses; however, as any peer that sends a responding shout will spoof its source IP address, they will not be able to tell if it is the node with that IP address that is responding or not. The adversary will need to monitor the links closest to the nodes with the receiving IP addresses in order to correctly determine whether or not the node is a peer. As there are likely to be as many links that need monitoring as there are IP addresses in the shout list, larger shout lists will mean the adversary must spend more resources on determining the true IP address of a peer. Peers should make the number of IP addresses in their shout lists large enough such that an adversary is dissuaded from performing the search.

Above, it has been assumed that the adversary has gained a copy of the shout list. In the network, shout lists are not freely distributed but rather delivered to those with a need to know (i.e. a peer's neighbours). It is possible for an adversary to become a peer's neighbour but this adds an additional hurdle to performing the search.

2.1.2 IP spoofing

IP spoofing is easily achieved, the protocol itself requires the sender to set the source IP address field to their assigned IP address. There is no reason the sender cannot set this to some other arbitrary value. That said, there are other hurdles that prevent IP spoofing from being performed. These are discussed in more detail in Chapter 3.

Ideally, the sender should set the source IP address field to a random address for every packet sent. If they were to use a single address, then an adversary might be able to block all packets coming from that address and thus prevent the communication occurring. The use of random addresses would also increase the difficulty of traffic analysis on the traffic that the sender sends as packets from a single sender are much harder to identify. In reality, the Internet's routers block packets with obviously spoofed IP addresses. Therefore, before engaging in communication with the network, peers needs to ascertain which IP addresses they can actually spoof. It is suggested that the peer sends test packets with spoofed source IP addresses to some machine they have control over and then get that machine to report which IP addresses were successful as spoofed source addresses. These addresses should then make up the pool from which spoofed source addresses are randomly chosen.

2.1.3 User Datagram Protocol (UDP)

UDP is a very basic protocol on top of IP. It simply adds the use of a virtual 'port'. It is desirable for use in the network because it is stateless and uni-directional. Because the shout removes the sender's true IP from each packet, the receiver has no means of replying to any packet received so a bi-directional protocol such as TCP is inappropriate.

Whilst we could give the receiver the shout for the sender, forming bi-directional communication should be avoided. The reasoning behind this is that packets travelling between two endpoints will take the shortest route. It follows that if the communication is bi-directional, messages travelling in both directions will take the same route. If an adversary is eavesdropping at some point along the path, they will pickup both directions of traffic. To avoid this, the network will allow messages to take multiple routes between sender and receiver which will help prevent an adversary from associating both sides of the communication.

2.2 Shout Groups

To maintain anonymity we require that sender and receiver peers do not know the other's IP address. The sender is protected through IP spoofing and never gives its own IP address to the receiver. The receiver, however has given their IP to the sender by placing it somewhere in a shout list. Here, we assume that there are two peers, A and B, who have exchanged their shouts. As stated previously, we don't do this in the actual network as bi-directional communication is to be avoided, we do it here only for simplicity. Assume A can request some data from B and get a response; A first shouts its request to B and B then shouts the response to A. If A wants to learn the IP address of B it could do so simply by searching the shout list. A chooses half of B's shout list and sends a request, if A receives a response from B then B's IP address is in that half of the shout list, otherwise B's IP address is in the other half. A can then split that half again and repeat the process. This continues until A finds a single address which will result in a response when a request is sent to it. Logically, this is B's IP address and the anonymity granted by the shout list has been broken.

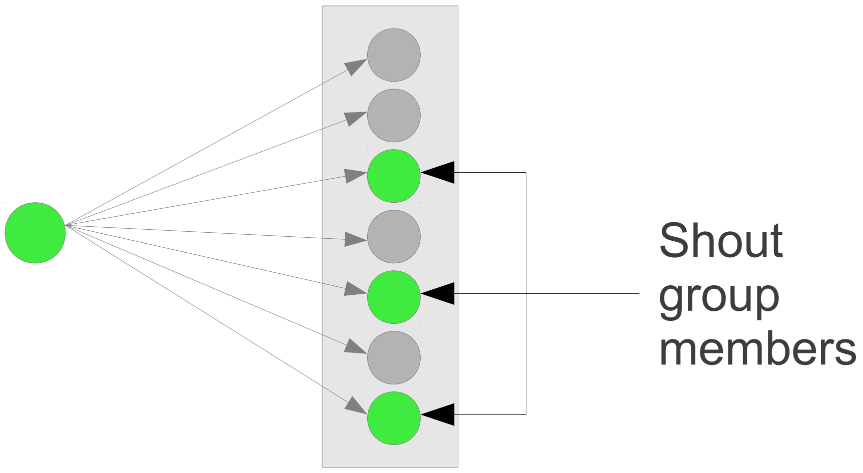

Figure 2.2: Each receiver in the shout group has their IP address in the shout list

Shout Groups are a method of preventing such searches. Multiple receivers share the same shout so that shouting to it will cause all the receivers to receive a message. In essence, each receiver puts their IP address into the list (see Figure 2.2). We can now prevent against searches by requiring that the packet must be received by multiple receivers in the shout list before a response is sent by the recipient. Note that a receiver is defined as any node receiving a particular packet whereas the recipient is the packet's intended destination. There are issues with this method that need addressing. How do the receivers tell each other that they've received the message? What if the adversary controls some of the receivers? What if we used a linear search instead of a binary one? The following sections address these issues.

2.2.1 Preventing Search

The search the adversary performs works on the principle that there are subsets of IP addresses in the shout list that will cause a response to be given by the recipient if a request is sent to just that subset. If shouting to the subset yields a response then the subset must contain at least one IP address of a peer. Consider if the recipient operates such that only if all of the receivers receive a message will the recipient send a response. With this in mind, if we were to conduct a binary search, two randomly chosen halves of the full shout list are unlikely to have all the receivers in one half therefore neither subset will result in a response being sent when shouted to. Even if the adversary manages to get lucky with the split, they still have to repeat the process several more times with more lucky splits required. This thwarts this method of search.

Consider instead a linear search where the subset the adversary sends to starts out as the entire shout list. Every time shouting to the subset yields a response, remove an IP address from the set. If the subset does not yield a response, then the last IP removed from the subset must be a peer. Replace this IP address in the subset and remove another one. Repeat until all the IP addresses have been removed at least once; the remaining IP addresses are all peers. Clearly this breaks the anonymity of the shout so a better method of defence is needed against shout list search.

To do this we need to use a probabilistic defence. Consider again the linear search; we can detect an adversary because the number of receivers their request hits sometimes decreases by one. If the receivers keep giving responses to the adversary, even when it doesn't send to all the receivers, then the adversary will not be able to tell if the IP address they removed is a peer or not. However, at this point the shout group can detect that the sender is performing a search. Consider the case where the recipient stops giving responses to any further request by the adversary at some random point after the search was detected. This cut-off has to come before the adversary removes all of the peers from the subset it sends to or the recipient will be unable to respond to the request (at which point the adversary may assume that the last IP address they removed was a peer's IP address). Another consideration is that if the shout group performs a cut-off after a random number of shouts have been received from the adversary, the adversary could negate the effect of the random cut-off by continually sending to the same subset of addresses to see if the responses eventually stop which indicates that they have triggered the random cut-off. Using this method would allow the adversary to determine whether or not the last IP address they removed belonged to a peer.

A formal description of the problem is as follows. We have an adversary, A, the set of IP addresses in the shout list, L, and the set of IP addresses belonging to receivers in the shout group, $V, V \subset L$. The goal of A is to find $x \in V$ with only knowledge of L; to do this, A interacts with an oracle, B. A gives input to B via an intermediary C which when given a list of IP addresses, S, by A will pass $S \cap V$ on to B. B returns $\{0, 1, \top\}$ for each input passed to it; 0 represents no response from the shout group, 1 represents a response and $\top$ represents that B has performed a cut-off. We aim to construct B such that A will output $x \in V$ with probability equal to if A had randomly picked a member of L. A and B know L, A knows S but not V and B knows V but not S. A may reset B after its cut-off and perform another set of interactions with it. B's goal is to make the number of resets needed for A to give a correct value of x as large as possible. In other words, the goal of B is to make:

$$Pr[x \in V | x \in L] = \frac{|V|}{|L|}$$

Consider the case where we cut-off the adversary at some random point after the adversary stops sending to half of the addresses in V. The adversary will then have two sets, S and $(L - S)$, with roughly half of the addresses in L in each. These are two sets that could have been created by A randomly splitting the addresses in L and would have required almost no knowledge from B. I say 'almost' because both subsets will contain exactly $\frac{|V|}{2}$ IP addresses that are in V whereas if the set S were picked randomly then we would expect the number of addresses from V that S contains to be binomially distributed:

$$Pr(|V \cap S| = \frac{|V|}{2}) = \binom{n}{k} p^k (1-p)^k$$

$$n = |V|, p = \frac{1}{2}, k = \frac{|V|}{2}$$

Therefore, in order to make the sets appear more indistinguishable from the binomial, B should perform the cut-off at some random point after it appears that A stops sending to some threshold, t, of addresses in V and before A stops sending to $|V| - t$ addresses in V for some t where $1 \leq t \leq \frac{|V|}{2} \leq |V| - t \leq |V|$. This gives the adversary less information than before; now S may contain between t and $|V| - t$ members of V instead of it being guaranteed to contain $\frac{|V|}{2}$ of them.

Whilst we cannot produce perfectly indistinguishable sets, it has been shown that we can drastically restrict the information that the adversary can learn from a single set of interactions with the shout group (this set of interactions being from first contact with the shout group to when it is cut-off). The adversary will have to use multiple sets of interactions with the shout group in order to better build a pattern and distinguish members of V from members of L. As will be discussed, we implement a proof-of-work scheme in order to greatly slow the collection of data points.

One further thing is necessary for this method to work. An adversary could send the same S to B multiple times. If it reaches the point where it removes the t-th valid IP address from S and repeatedly sends this to B, B will eventually perform a cut-off and the valid IP address will be revealed as S has not changed since the last valid IP was removed. This does require more time to perform but does guarantee the A will return a valid x. To prevent this, the rate at which S changes must be monitored by B through the information it has about $S \cap V$ and the number of interactions that A has performed. The rate at which A removes addresses from S can then be estimated and the cut-off can be adjusted so that it occurs between the removal of t and $|V| - t$ valid addresses in a random fashion.

Proof of Work

When a cut-off is performed, what actually happens is that the shout group will no longer respond to messages that it receives from a party with a given identity. If this party is an adversary, they could resume the attack simply by choosing a different identity to make requests to the shout group with. This is where the proof-of-work scheme comes in.

A proof-of-work is intended to prove that a party has performed a certain amount of computational effort. What the computation is calculating is unimportant, what matters is that the result should require the party to expend a non-trivial amount of computational resources. The result obtained proves that the party has expended these resources and this result is easy to verify. Additionally, a proof-of-work computation takes an input and the resulting proof-of-work proves that the computation was performed on that input. This means that a party can be prevented from pre-computing a proof-of-work by providing new data that the party must perform the proof-of-work computation on.

The Bitcoin network uses a modified version of the Hashcash [4] proof-of-work scheme. In order for a party to produce a proof-of-work, they must perform enough cryptographic hash computations such that the resulting numeric value is lower than a predetermined difficulty value. The lower the difficulty value is, the harder a proof-of-work is to produce. By modifying the level of difficulty, the rate at which these proofs-of-work are produced can be regulated.

It is recommended that neither the Hashcash, nor the Bitcoin variant is used. This is because dedicated hardware[3] exists specifically to perform these proofs-of-work. An attacker with access to this hardware will be able to produce proofs-of-work much faster than an average user. This will render the requirement of a proof-of-work, at a fixed difficulty level, useless. Even if the difficulty level is dynamically modified, because the computational power drastically outweighs that of legitimate users, the attacker can prevent new users from joining the network by causing the difficulty level to be too hard for an average user to surpass. It is therefore wise to choose a different computational problem for this project such as a memory bound function[1].

With regards to preventing search of a shout group's shout list, a proof-of-work system is used to make searching the list much harder. When a node wants to join the network, it must first produce a proof-of-work linked to its identity. If the node turns out to be an adversary (and is cut-off as seen previously) then that identity is banned from the network; a note is made by every peer to never accept that proof-of-work again. In order to rejoin the network after being banned, an adversary must first create an entirely new identity and produce a proof-of-work for that identity. This will severely hinder an attacker's progress in determining the identity of peers in the network.

By dynamically adjusting the difficulty of creating a proof-of-work, the protection the proof-of-work scheme offers can be future-proofed against increases in computing power. The rate of joining the network can be set to be a fixed value (e.g. 3 new peers every minute) and the threshold is then adjusted as needed to maintain the low rate. In contrast to Bitcoin where proofs-of-work are constantly being sought after, there may not be peers trying to join the network all the time and so proofs-of-work may be far and few between. Therefore, the difficulty of creating a proof-of-work is set to never decrease beyond a certain threshold. Otherwise an adversary could simply wait for the difficulty to fall to a low enough point before performing a search on a shout list. The adversary could then repeatedly join the network with ease as producing proofs-of-work has become trivial.

2.2.2 Message Receipt Verification

The shout group has previously been assumed to be a group of nodes who all explicitly trust each other and none of whom are adversarial to the purpose of the shout group. However, unless users of the network create a shout group amongst a group of themselves who they all mutually trust, there need to be mechanisms in place to prevent an adversary breaking the anonymity granted by the shout group. If an adversary exists amongst the shout group then they can see which nodes have received which messages. With this information it is trivial to simply send a packet to each IP address in the list and see if a member of the shout group states that they have received the message; if the message has been received by a member then the IP address that that message was sent to is the true IP address of a shout group member.

This problem can be generally expressed as a Secure Multiparty Computing (SMC) problem[13]. Each node in the shout group knows which messages it has received and which anonymous identity send it; identifiers for these messages and their senders become the private inputs. The output should indicate which messages should be accepted by the network and which senders are to be banned (if any). This problem is made exceedingly challenging by the following necessary requirements.

The sender must not be able to determine exactly how many shout group members have received any given message; if they can, a hostile sender could remove addresses one by one from the shout list until the number of receivers drops by one which would reveal the true IP address of a shout group member. Each message must be received by at least some threshold of shout group members (call this t) if it is to be accepted; otherwise, a hostile sender could send messages to single IP addresses in the shout list until the corresponding message identifier appears in the output. Furthermore, receivers should not be able to pretend to have received a message when it has not, nor should they be able affect the computation by pretending that more receivers have received a message than have actually received the message. The computation must keep track of the state of senders who have been detected to be performing search on the shout group and this must be done without revealing to the hostile sender whether it has been detected or what its random cut-off is. For the purposes of preventing search explained in the previous section, the random cut-off would be set once the number of receivers the message is sent to decreases to $V - t$. If the hostile sender sees that its random cut-off has been set, it may assume that the last IP address it removed from the subset of addresses is a true IP address of a member of the shout group.

Partial solutions to this problem exist. Assume that the shout group is setup where only one member is the recipient of all messages received by the group. Furthermore, assume that if the recipient is hostile then it is not colluding with any other shout group member. This reduced problem can be solved using secret sharing. Each secret sharing scheme has some secret that is to be shared and a system from which those shares are generated. If each non-recipient node produces some share in the system and then sends them to the recipient then the recipient will be able to determine the secret if t or more valid shares are received. The secret will tell the recipient which message has been received by the parties. The barriers to implementing such a scheme are that the non-recipient shout group members creating the shares have to agree upon a system in which to create the shares without letting the recipient know the details of the system.

For instance, if we were to use Shamir's scheme[22] we would set a_0 to be some cryptographic hash of the message and set the coefficients of the polynomial to be random $a_i \in \mathbb{R}$ for $i \in 1 \ldots t-1$. The nodes must agree upon the same random values or the secret cannot be recovered. Additionally, revealing the coefficients to the recipient that has not received the message will allow it to produce more shares in the system. Thus, if the recipient is an adversary, they can simply produce $t-1$ shares to combine with any given share from another node. If the share from a node combines with the $t-1$ shares to reveal the secret then the adversary knows that the node sending this share has received the message. As it can be determined if less than t nodes have received the message, this breaks the conditions for our system's security. The same random coefficients can be generated by all non-recipient shout group members by sharing a seed to a Cryptographically Secure Pseudo Random Number Generator (CSPRNG). This has to be kept secret from the recipient or they can generate more shares in the system which breaks the system's security as explained previously.

No complete solution to this SMC problem is currently known. A solution to the problem may be found in the future and this can be readily applied to the network. For the time being, shout groups will have to be constructed from mutually trusted parties.

2.2.3 Shout Group Creation

If a solution for message receipt verification is found, shout groups can be constructed from nodes that already exist within the Shadow P2P network. Again, this can be constructed as a SMC problem. Each node that is to become a member of the shout group provides the private input of a shout list, constructed as in section 2.2. The output is a random permutation of the input lists which becomes the shout group's shout list. No node should be able to determine which IP address belonged to which shout list.

The solution to this is quite simple. The node that is constructing a shout list for itself is assumed to not be in the network. The constructing node chooses some of the nodes in the network at random and determines a circular arrangement of them which includes the constructing node. The constructing node then encrypts each IP address in their shout list individually under an asymmetric encryption algorithm using a public key belonging to that node. This list of encryptions is then sent to the first node in the circle of nodes. Each node in the circle will receive a partial list from the node before it and add encryptions to the list created from its own shout list. This larger list is then shuffled randomly and sent to the next node in the chain. When the

```
**Data**: The group parameters, $(G, g, q)$, the public key, $h$ and the message $m$.
**Result**: The ciphertext pair $(c_1, c_2)$.
select a random $y = \{1 \dots q\}$
$c_1 = g^y$
$m\prime = m$ converted to a member of $G$
$s = h^y$
$c_2 = m\prime.s$
$return(c_1, c_2)$
```

Algorithm 2.1: ElGamal encryption.

```
**Data**: The group parameters, $(G, g, q)$, the private key, $x$ and the ciphertext pair $(c_1, c_2)$
**Result**: The message $m$
$s = c_1^x$
$m\prime = c_2.s^{-1}$
$m = m\prime$ deconverted from a member of $G$
return $m$
```

Algorithm 2.2: ElGamal decryption.

constructing node gets the list back from the last node in the circle, it can decrypt each IP address in turn to obtain the finished shout list for the shout group.

This shout list can then be distributed back to the participants (using the shout list itself to perform the shout). Each participant should inspect the list to check if each IP address it placed in the list is present and if so, it should compute a signature on the list and send it to each member of the shout group, again using the shout list. If every member of the shout group receives valid signatures from all other members then the shout group has been created successfully, otherwise the shout group creation is abandoned and the process is restarted.

2.3 Hiding Public Keys

The peers in the network maintain confidentiality in their communications through the use of symmetric and asymmetric key cryptography. When sending packets to a given receiver, generally a form of hybrid encryption is used where the bulk of the data is encrypted symmetrically and the key for this is then encrypted under the receiver's public key. This is a widely used methodology however in Shadow P2P, there is a drawback. The onion routing method I intend to use will involve intermediate peers adding layers of encryption onto packets in order to alter packets going through a node so that they become unrecognisable once processed. This is done through symmetrically encrypting the packet under a secret key and then encrypting this key asymmetrically. The problems here are that:

1. The public key will be part of the packet and encrypting it as it passes through the peer would render the public key unusable. Therefore the public key has to be left unchanged. This would allow an adversary to recognise a packet that exits the peer as being a specific packet that entered it because the public key that the packets bear will be the same.

2. If the adversary knows which peer the packet's public key belongs to then it can tell which node the packet is destined for.

Both of these are a threat to anonymity, the adversary gains knowledge of where the packet is heading and possibly information on where it came from.

To avoid this, I have invented a method of manipulating public keys such that they cannot be recognised as above but so that they can still be used for encryption. This method does not work with every public key system however it is known to work with the ElGamal cryptosystem[9]. It will therefore be described as applied to ElGamal in this section.

```
Data: The group parameter, $g$, and the public key, $h$
Result: The hidden generator, $g\prime$, and the hidden public key, $h\prime$
select a random $r = \{1 \dots q\}$
$g\prime = g^r$
$h\prime = h^r$
return $g\prime, h\prime$
```

Algorithm 2.3: ElGamal public key hiding.

2.3.1 Applied with ElGamal

In ElGamal, encryption and decryption are performed as in Algorithms 2.1 and 2.2. To hide the public key, h, we take both h and g and perform the hiding as described in Algorithm 2.3. The hidden public key is then the pair $(g\prime, h\prime)$. The value of r in algorithm 2.3 bears no relation to the value of y that used in algorithm 2.1 and, in fact, when they are selected by the same party, they must be different. Otherwise, if y and r are the same, the value of $h\prime$ can be used to decrypt ciphertexts encrypted with the ephemeral key y as $h\prime = h^r = h^y = s$.

Encryption under a hidden public key is as in algorithm 2.1 but changing the parameters such that $(G, g, q) \rightarrow (G, g\prime, q)$ and $h \rightarrow h\prime$. Decryption is performed identically to algorithm 2.2 with no change in parameters. The decryption algorithm still works because:

$$c_1 = g\prime^y = g^{ry}$$

$$c_2 = m\prime.h\prime^y = m\prime.h^{ry} = m\prime.g^{xry}$$

$$m\prime = c_2.c_1^{-x} = m\prime.g^{xry}.g^{-xry}$$

Hopefully it is also apparent that the hiding algorithm can be applied as many times as is desired whilst still maintaining the key's usability. Also note that no matter the number of times the key has been scrambled, the decrypting party still only needs to perform one exponentiation which is a useful side effect in terms of effeciency.

In terms of security, the anonymity holds as long as it is hard to compute discrete logarithms in G. If this is not the case then $r = \log g g\prime$ can be found easily and then an adversary need only compare $h\prime$ with $k^r \forall k \in K$ (where K is the set of public keys the adversary holds) in order to find who the public key belongs to. Additionally, the group parameters should be shared between all peers in order to anonymise the recipient among them; otherwise G and q need to be provided for the encryption and the recipient could be identified from these.

2.3.2 Applied with Elliptic Curves

Specifically, this method of hiding can also be applied to the Elliptic Curve Integrated Encryption Scheme (ECIES)[20]. As in ElGamal, the generator and public key are transformed with the same random number to get a hidden public key pair. In this case, the group generator, G, is hidden by computing $G\prime = r.G$ and the public key, K, is hidden by computing $K\prime = r.K$. For encryption, the value $R = y.G\prime$ is generated with some random y and the shared secret $S = y.K\prime$ is computed. This S is then used to derive keys for a symmetric encryption scheme and a Message Authentication Code (MAC) scheme. When decryption occurs, the private key, k, is used to compute the shared secret, $S = k.R = k.(y.G\prime)$. The shared secret is then used to derive the keys as done by the encyrpting party, as these keys are the same, decryption and authentication of the data can occur.

2.4 Uni-directional Toroidal Network

The connections between peers in the network are structured together in a regular fashion; specifically, they are arranged in an k-dimensional grid with the width in each dimension equalling w. Links are then created between the peers following the dimensional axes, however, links only allow transmission in one direction (and thus uni-directional). This would prevent responses reaching peers that sent the original requests; to correct for this, the network is connected such that when a packet reaches the edge of the network in a given dimension,

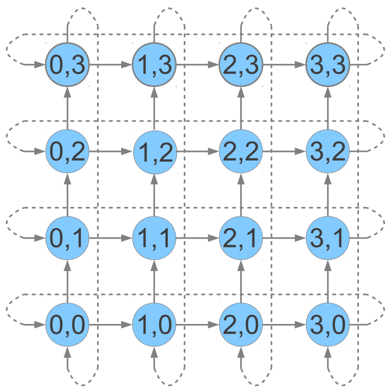

Figure 2.3: An example of a uni-directional toroidal network structure

the packet is sent back to the first peer in that dimension (therefore the network is toroidal). This structure is shown for a 2D network with width 4 in Figure 2.3.

This project chooses this network layout over other designs as it arguably provides more anonymity than those used in other distributed networks. A popular design amongst these networks is the mesh network (meshnet); peers form a small number of links with other peers in the network without considering the overall network layout. Peers may later reorganise their links to increase network efficiency or robustness based on information they have learned about their local network structure. The flaw here is that if a network is optimised for speed, peers will form links with peers that they have lower latency with; in reality this usually means that a peer will connect to the closest peers because these will have lower latencies to the peer in question (which is in turn caused by constraints due to the laws of physics). This reveals some location based information about the peers and so this is clearly an undesirable system to use.

Randomly reorganising the peers from a list of all known peers is a much more anonymous strategy as it attempts to disregard all information pertaining to the physical world. However, this may still leak information when a peer is reorganising its connections when it doesn't have a complete list. The peer might not have a complete list because information about peers further afield will necessarily take longer to propagate through the network.

In reality, networks are rarely assembled as toroids. However, the toroidal network structure here is designed to be a virtual overlay onto any network structure. This is achieved simply by assigning nodes in the underlying network to positions in the toroid. As long as a path exists between every pair of nodes in the underlying network (much like on the Internet), all of the links in the toroidal structure can be formed and so the toroidal structure can be used.

Physical connections are not the only concern, routing methods are also of interest. The two main schools of routing are destination based and source routing based. With destination based routing, the packet's destination address is included in the packet so that intermediate peers can route it correctly. However, that same address can be used to collect and analyse all the traffic going to a specific destination peer. With source routing, the source specifies a complete (or partial) route through the network to the destination. This method doesn't explicitly specify the destination so unless the intermediate peer recognises the path, the destination becomes anonymised. In Shadow P2P, complete source routing is used in a way such that it is computationally infeasible to determine the destination. The network also takes advantage of the regular structure to allow senders to utilise a wide range of routes to the destination as will be discussed.

2.4.1 Source Routing

We assume here that each peer has complete knowledge of the network, they know where each peer is positioned and all required cryptographic material. A route can be specified through the network by providing a direcion in which the packet should be sent at each peer in the path. In the example given in Figure 2.4 a packet will be sent down the blue path if it uses the routing information "up, right, right, right, up". The red path specifies an alternative route, perhaps for a second packet. As shown, the packets only have a single intermediate peer in common and therefore only this peer can attempt to analyse the packets as a pair. It should be clear that there are many different routes between sender and destination and that the sender can choose any one of them to send traffic down. It can be shown that, in 2D networks, there are $\binom{dx+dy}{dx}$ unique routes through the network between two peers separated by dx peers and dy peers on the x and y axes respectively.

This distribution of traffic amongst many different routes will make traffic analysis exceedingly difficult. In particular, it can disguise a large data transfer as individual packets are dispersed among a large number of intermediate peers. This would not disguise the fact that a particular peer is engaging in such a transfer but it will disguise who the other participant is, assuming such transfers are common on the network. This is because, if an adversary monitored the amount of traffic going into and exiting each peer, they would see significant rises in the traffic of the two peers performing the transfer but only small rises in the peers between them. If the small rises are within the normal operational bounds of those intermediate peers, the adversary will not be able to determine both the sender and receiver. One way to achieve such an even distribution of traffic would be to use Valiant's algorithm[25].

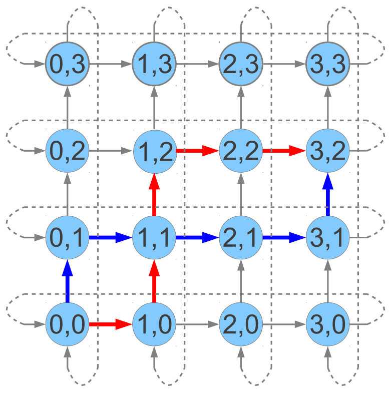

Figure 2.4: An example of different paths through the network. The sender is at (0,0) and the receiver at (3,2)

One particular concern with routing is that of packets being routed forever and never reaching a destination. Such packets would build up in the network and eventually cause the network to grind to a halt. A packet going from a sender to the destination must, at most, traverse links in a given direction equal to the network size in that direction less one. Therefore we can limit the size of the routing field so that packets cannot traverse the network forever.

2.4.2 Dynamic resizing

With such a regular structure, an obvious concern is that of network expansion and contraction. How can a regular grid of peers be maintained when adding and removing peers. To this end, I have conceived a method for allowing peers to join and leave the network freely without compromising the structure of the toroidal grid.

To do this, I introduce the concept of *virtual positions*. Up until now, it has been assumed that each peer exists at exactly one position on the grid. Now, peers may take several positions on the grid simultaneously, whilst they exist only once in reality. We call the positions that the peer takes, "virtual positions". This is very useful as we can now freely extend the grid without needing extra peers to fill the gaps.

Expansion

This is best explained in reference to a diagram, see Figure 2.5. First, we require that the network is square, i.e. the network sizes in each dimension are always equal. Then, to expand the network, we simply duplicate the grid of virtual positions in each dimension. Now each peer has 2^k times more virtual positions than they did before. The most important point to note here is that the network has been resized but no new physical connections needed to be created. For instance, in the diagram A still has B as its northwards neighbour and C as its eastward neighbour in all cases. This is achieved because we are simply exploiting the looping nature of the toroid to provide more space by 'unrolling' it. Also important to note is that this expansion is implicit; the peers all know which virtual positions they are going to take ahead of the expansion.

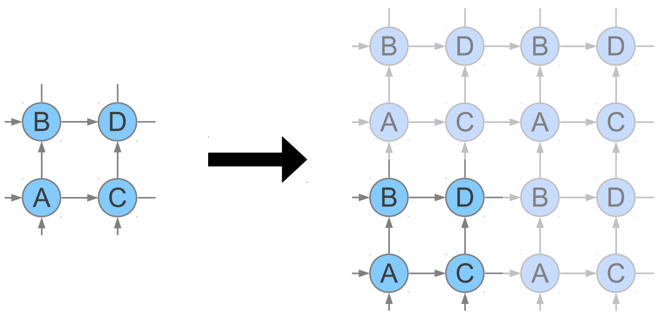

Figure 2.5: A demonstration of network expansion. Wrap-around links not shown.

When a node wants to join the network, it will must contact an existing peer which will help the node join the network. This peer will then replace one of its virtual positions with the new node. The peer will inform both the joining node of how to contact its downstream neighbours and will inform the joining node's upstream neighbors of the change in their downstream neighbour. In Figure 2.6, the node E replaces A at coordinates (2,2). A will tell E how to contact B and C and it will tell B and C how to contact E. Figure 2.7 shows how this looks "in reality". Note that D has not been given any information regarding E's join; it simply has no need to know.

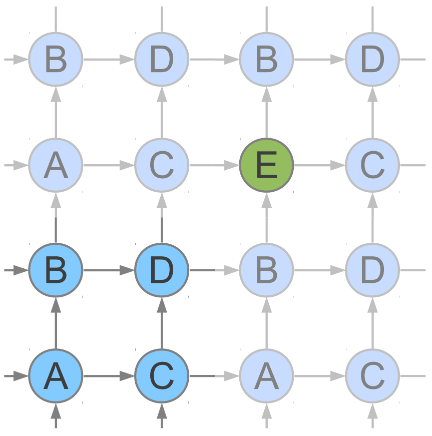

Figure 2.6: Peer virtual positions and virtual links.

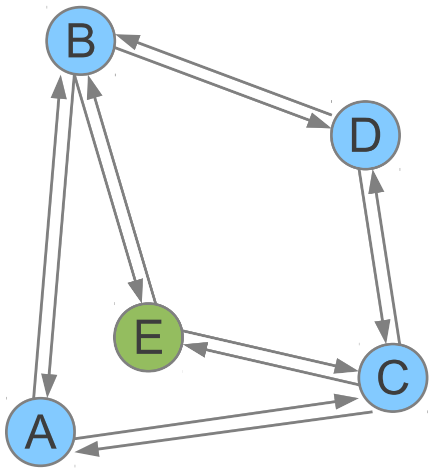

Figure 2.7: Peer physical links.

As the network expands, peers gain new virtual positions in predetermined places; these are the peer's 'inherited positions'. In the 2D example, a peer that owns the virtual position (x, y) inherits the virtual positions $(x+w, y)$, $(x+w, y+w)$ and $(x, y+w)$ where w is the original width of the network in any given dimension (recall that previously the network was defined to be the same width in all dimensions). If we were to expand again, the peer with virtual position $(x+w, y)$ would inherit positions $(x+3w, y)$, $(x+3w, y+3w)$ and $(x+w, y+3w)$. As the netwok begins with a single peer, the network will always be a power of 2 in width in any dimension because the network expansions causes all widths to double. In addition, every virtual position that the peer inherits also has virtual positions that they inherit. The virtual positions that a peer will inherit if it owns virtual position (x, y) is then:

$$(x+ai, y+aj)\ a = \lfloor \log_2 (max(x,y)) \rfloor + 1; \forall i, j \in \mathbb{N}$$

This provides every peer that owns at least one virtual position with a potentially unlimited number of descendant positions. This will allow a peer to continue in helping nodes join for as long as that peer desires. When a node joins, the helper peer should give the joining node a virtual position as close as is practical to the helper peer's 'local root' to prevent expanding the network unnecessarily. The local root of a peer is the virtual position from which it inherits all of its other virtual positions. It should be noted that when a peer is given a virtual position, it also gains all of the virtual positions that the virtual position inherits as described by the equation above. The virtual position inheritance scheme is clearly arranged in a tree with virtual postition (0,0) being the root (see Figure 2.8).

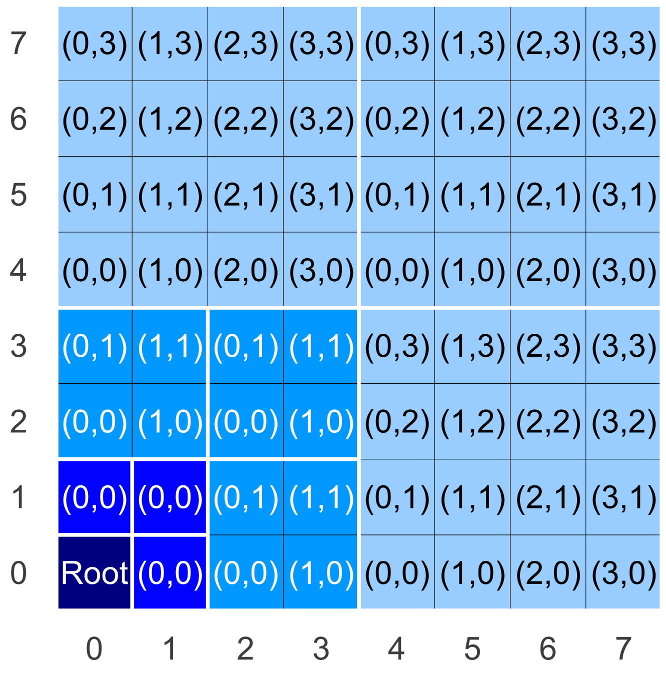

Figure 2.8: Each virtual position indicates the coordinates of its immediate parent in (x,y) form.

Balance

Peer have to process traffic coming from (and going to) two virtual positions for every virtual position that they own. Therefore, peers should be inclined to give away their virtual positions. In Figure 2.9, A has given out all of its virtual positions to new peers that have joined the network. Unfortunately, the links are poorly balanced between peers; B and C have ten unidirectional connections apiece wheras all other peers have just four. To rectify this situation, we allow peers to be repositioned through the trade of virtual addresses. In Figure 2.10, B gives position $(3,0)$ to E and E gives $(2,0)$ back to A. G moves in a similar fashion. The links between peers are now far more balanced.

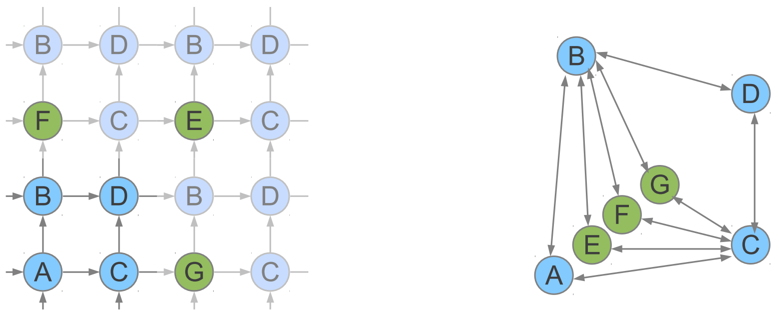

Figure 2.9: An unbalanced arrangement of virtual positions.

Another consideration is how willing peers are to perform a balancing trade. For example, B asks E if it will take the virtual position it has at $(3,0)$, B is willing to do this trade as it will lower B's throughput. If E is willing, it informs A that it is leaving the position (2,0) so A can take control of the position again; it then places itself at (3,0). If E is not willing, B can ask another peer to trade positions with. A is the party most likely to be unwilling to perform the trade as it increases its load, however, as the other peers know of the virtual positions A owns, A can be forcibly removed from the network for 'not pulling its weight'.

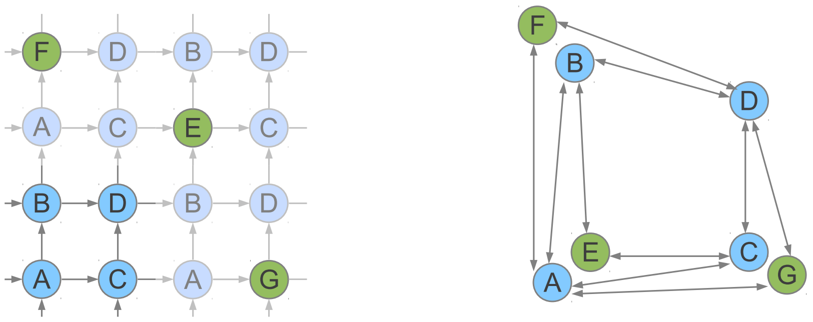

Figure 2.10: A more balanced arrangement of virtual positions.

Contraction

Contracting the network is basically a reverse of the expansion process in that the network width is halved in each dimension. In order to contract the network, the peers have to be in a configuration identical to if the network had just undergone an expansion, i.e. the peers' virtual positions are repeated once in each dimension (see Figure 2.11).

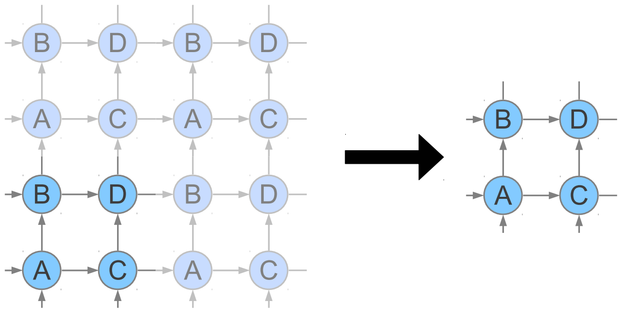

Figure 2.11: Once the peers enter the correct configuration, the network can be downsized.

This clearly requires the peers within the network to perform a feat of organisation. This is accomplished by controlling how peers leave the network. As peers leave the network, they return their virtual positions to other peers. Consider a single virtual position belonging to the leaving peer. If one or more of the corresponding inherited virtual positions have been given to other peers then one of those peers is asked to take possession of the current virtual position. If it has not given the inherited virtual positions to other peers then it gives the virtual position to the peer in possession of the virtual position's successor in the tree structure. This ensures that all of the peers end up owning positions closer to the root which eventually achieves the configuration required to shrink the network.

2.5 Summary

In this chapter, I have introduced three techniques that contribute to my objective of making communication methods between network peers that are more anonymous than current techniques. The shout removes all information about the sender and most information about the receiver from the messages that are sent which allows neighbouring peers to communicate without revealing their IP addresses; this is a feature not offered by any network previously discussed. The public key hiding technique allows peers to encrypt under a public key without being able to determine which peer the key belongs to. Finally, the uni-directional toroidal structure removes information that might allow an adversary to identify a peer from a peer's position in the network. It also allows network traffic to take a multitude of different paths to its destination which makes collecting a complete communication between two peers exceedingly difficult and so traffic analysis aimed at identifying peers can be greatly hindered.

In addition to the anonymity techniques, I introduced shout groups whose express purpose is to protect the level of anonymity granted by the shout technique and contributes to my objective for inventing defensive measures against adversaries. The technique allows peers to detect and then severely hinder an adversary's attack without revealing their identities whilst still allowing communication between peers to occur.

Chapter 3

Project Execution

The overarching goal of this project is to design a network, in full, that meets the requirements for anonymity previously stated. To achieve this, I have designed a new anonymous network named "Shadow P2P". This chapter describes the network's aspects and the reasoning behind them.

This chapter takes the techniques introduced in the previous chapter and incorporates them into the design as follows. The shout technique is used by the peers in the network for communication with their neighbours. This is coupled with the shout group technique which each peer will use to detect and prevent foul play by adversaries. Each peer therefore requires a shout group before it can join the network. The peers are arranged according to the uni-directional toroid structure that I described; peers can then communicate with every other peer. Shouts will be used by the peers to send messages via their downstream neighbours and the responses will be returned by their upstream peers. The majority of packets that are sent within the network are encrypted and through the use of the public key hiding technique, the packets can be made untraceable.

What remains to be done is defining what each node requires in order to become a peer and how the peers interact in order to maintain this anonymous network. On top of this, methods for discouraging and / or preventing attacks on the network must be designed. These methods are described in the following sections.

3.1 Node Requirements

3.1.1 IP Spoofing

This project makes some assumptions about the abilities of the network's peers, namely, that it can perform IP spoofing. It is further assumed that the peers have a single IP address assigned to them by their ISP (which doubles as the peer's identity in the design) and that the connection between the computer running Shadow P2P software and their ISP is mediated by a router in possession of the user. As discussed previously, NAT and ISP source address filtering make IP spoofing a non-trivial task. It is suggested that the user have their node send packets with a spoofed source IP address to a physically distant machine that is also under the user's control. If the packets arrive at the distant machine, then the node can perform IP spoofing.

Once IP spoofing has been achieved, the user will need to acquire software for interaction with the Shadow P2P network. How this is achieved anonymously is beyond the scope of this project but simple methods do exist. For example, the software may be passed between users on some removable media; as this takes place without use of the Internet, there is no chance of the user revealing their IP address. On top of this, the user will have to acquire a shout list for a peer in the network. P2P software is often distributed with IP addresses of known peers so, in Shadow P2P, there may also be shout lists of known peers included. Whilst it is true that a peer is more vulnerable the longer it maintains the same shout list (in the case of ongoing attacks), the peer can stop responding to requests using a given shout list at any time which renders it useless to attackers. The software distributors would then later remove the unresponsive shout list and replace it with another.

3.1.2 Creating a Shout Group

Before a node is ready to join the network as a peer, it will need to generate their own shout list and join a shout group. Shout list generation can easily be performed as described previously. Creation of a shout group, however, is harder as it requires nodes to create a shared shout list from their own shout lists without revealing to one another what their shout lists are. Shout groups can be formed in one of two ways.

If the user can find a small group of potential users offline, then this group of users can have their nodes form a shout group. In this case, the users will each make their own peer in the network, each of which has the same shout list being used to contact it. This does involve meeting those users in the shout group face to face so it would be preferable to form this group from people that the user knows and trusts personally.

The user can also create a shout group from existing peers in the network. To do this, the user gets their software to engage in the shout group creation protocol. This is explained further in Section 3.3.2.

3.2 Network Structure

Here, it is detailed how the peers in the network are organised. Peer arrangement, necessary information and peer joining / leaving are explained. The network takes on the form of a uni-directional toroid as explained in Section 2.4. This structure is necessary in order to perform the proposed method of routing and network resizing. Communication between peers is achieved using the shout and shout group methods.

3.2.1 Authenticated Virtual Position Transfer

In order to change which peers are connected to which other peers, virtual positions are transferred between peers. If a peer is informed that the virtual position of one of its downstream neighbours has been given to another peer, it will have to change the shout used for that neighbour.

It was shown earlier that the peers of the grid formed a tree for the purposes of inheriting virtual positions. When a virtual position is given to another peer it also gains all virtual positions that that virtual position inherits. A potential problem that needs to be addressed is that a peer belonging to an adversary could claim to joining nodes that they own a given virtual position on the network when in reality they do not. A joining node needs to know who to believe when collecting information about the network.

To validate the transfer of ownership of a virtual location from one peer to another, a signed certificate detailing the transfer is created by the peer doing the giving (the giver). It is required that the giver signs the message or other peers could simply claim they own the positions that rightfully belong to the giver. These certificates will form chains reminiscent of the X509 certification model. The signed certificate is then broadcast to the network so that peers can update their information regarding who owns which virtual position. The other peers in the network are then the regulators for virtual position ownership. They can ensure that no virtual position is given twice and that no peer gives a virtual position that it doesn't currently own. This resembles the way the Bitcoin network prevents double spending of its currency.

To keep things elegant, we restrict which virtual positions can be given and which peers they can be given to.

- A peer may only ever receive one virtual location (note that when given a virtual location, all the virtual positions that that position inherits are also transferred). This virtual position becomes their "local root". This also prevents a peer simply offloading virtual positions that it is responsible for onto arbitrary peers.
- Peers may only give virtual positions that are immediate descendants of their local root.

The network also requires a mechanism for returning virtual positions for when a peer leaves the network. There are two cases that need to be covered:

- The peer has not given any of its local root's descendant virtual positions. The virtual position is simply returned to the peer that owns the local root's parent virtual position. This takes the form of a signed message. This message can then be broadcast to the network so that the other peers are aware that the original owner has regained ownership of the virtual position.

- The peer has given one or more of its descendant virtual positons to other peers. In this case the peer performs a breadth first search on the peers owning the children of its local root. When it finds a peer that has not given any of its virtual positions, it sends a signed message to the parent of its local root virtual position indicating its intention to leave and the virtual position of the replacement peer that it has just found. The parent owning peer will then broadcast the signed message (causing the virtual position to change ownership) and will give the virtual position to the replacement peer. The replacement peer will then return its current virtual position to the peer owning its local root's parent. The peer's local root will then be that of the peer that has just left the network.

These operations should result in a tidy tree of certificates. Each peer that has been given a virtual position will have a certificate from the peer owning the immediate parent of that virtual position, proving that it does indeed own that virtual position.

The exception to this is the peer that owns the root virtual position of the network. It quite obviously does not have a parent virtual position and so if the peer leaves, there will be no peer that can take its virtual position or reassign it. In this special case, when the peer finds its replacement, it signs the message itself that gives the root virtual position to the replacement. This is similar to X509 where root certificate authorities sign each other's certificates because they have no parent authority.

Should a certificate be produced that conflicts with the requirements of the network, or with current ownership, then the network will simply not accept the change.

The network also has to handle peers disconnecting without "saying goodbye". This presents a problem because the methods outlined for transferring virtual positions thus far assume that all peers disconnect from the network in a controlled manner. To handle this, we further require that peers assert their claim to the virtual position they have been given at regular intervals. This assertion will take the form of a message broadcast to the network telling it that it is still there. If a peer doesn't receive assertions from the peers that it has given its local root's immediate descendants to then it will remove them from the network. It does this as if the disconnected peer informed it of its intention to leave and finds a replacement peer as before. When the peer signs the message giving the replacement peer that virtual position, it will be accepted by the other peers because they will also see that the peer has failed to assert its ownership.

3.2.2 Necessary Information for Complete Network Knowledge

Before the routing is possible, the peers need to have complete network knowledge, i.e. the peer must know the virtual positions and public keys of every peer in the network. This information is necessary for anonymity because this enables the peers to send their communications down many different routes to their destination. The more paths there are available, the more packets in the same communication can be spread out; thus it becomes harder for an adversary to piece the traffic back together.

The information that needs to be known for each peer is as follows.

1. The public key for data encryption.
2. The public key for data signatures.
3. The public keys for routing in each dimension.
4. The proof-of-work token associated with these keys.
5. A signature of the above items using the signature key.

The data encryption and routing keys are required for routing, these keys need to be of a type which supports the public key hiding technique described in the previous chapter. The signature key is required for verifying message authenticity and virtual position database entries. The proof-of-work token verifies that a peer has gone through the required effort to join the network. The signature is used to bind all of the information together and prevent tampering. Together this forms a peer's keyset.

In addition to this, all certificates concerning which virtual positions have been given to whom must be acquired. These will confirm which peers are in which positions.

3.3 Packets

Shadow P2P uses a number of different packets for different purposes. For simplicity, the number of packet types is kept to a minimum. The packet structures, their uses and usages are described in this section. Due to the anonymity requirements of the network, all packets represent unreliable transmissions. A TCP-like 'ACK' response to each packet can aid adversaries in detecting a receiver or sender of a communication. In general, the fewer packets that are sent, the better the anonymity.

Some packets require that they are broadcast. Such broadcasts need to occur a controlled manner to ensure that unnecessary transmission of the packet does not occur. The diagonal method described in [15] modified for a uni-directional toroidal network is a good way to achieve this. For this end, the virtual position of the source of the broadcast should be included with the packet; this will allow the peers receiving the packet to determine if they need to resend it and, if so, in which direction. As these broadcasts are used solely for information that is public to the network, there is no consequence to adding information about the broadcast's origin.

3.3.1 Generic Data

This packet should make up the bulk of the network's traffic. It is used for transferring any form of data between peers. This packet can be used providing that a path is known between from the sender to the receiver (if the sender has complete network knowledge then this requirement is obviously fulfilled). This packet provides confidential and authenticated data transfer. Furthermore, it ensures that messages cannot be traced as they pass through peers and that no intermediate peer can determine the destination of the message.

To prevent messages being traced, these packets use 'ephemeral onion routing' to scramble the packet's data in its entirety. Onion routing is performed as follows: a packet leaves the sender with a single layer of symmetric encryption. Whenever the packet passes through an intermediate peer, another layer of encryption is added on top of the first (and so layers of encryption are built up like layers of an onion). When the packet reaches the destination peer, each layer of encryption is removed in turn to recover the data. This is 'ephemeral' as each encryption key is chosen at random when the packet passes through the peer; in other words, none of the symmetric encryption keys are pre-agreed. The following sections explain how this process works in more detail.

The packet is divided up into 4 sections. The first section is for routing keys, the second is for the data encryption public key of the receiver, the third is for so-called 'hybrid headers' and the fourth is the data payload field (see Figure 3.1).

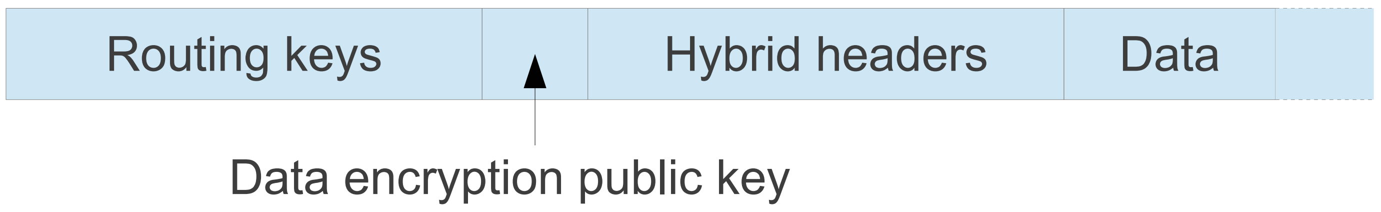

Figure 3.1: The structure of a generic data packet

Packet Construction

Once a random route has been selected to send the packet along, the routing keys field is filled out as follows. For each peer in the path, take the key corresponding to the direction the packet will take from this peer to the next peer. This public key is then hidden using the public key hiding technique and the resulting pair $(g\prime, h\prime)$ is appended to the field. After these keys have been added, pairs of random group members are appended to the field so that the final number of pairs in the field is equivalent to $k \times (w - 1)$. This is the maximum number of pairs that the field may take; it allows packets to be routed to any other peer on the network and prevents packets traversing the network forever. It should be clear that this field doubles as the packet's Time To Live (TTL) as the packet can only be routed through a finite number of peers.

The data encryption public key field is set to be the data encryption public key of the receiving peer with the hiding technique applied.

The hybrid headers field contains encryptions of symmetric encryption keys under the data encryption public key. A random symmetric encryption key is chosen and encrypted under the data encryption public key. If the symmetric encryption method to be used requires the use of an IV, it should also be included in this encryption. The field is then set to this encryption and then random encryptions are appended to this field until the total number of encryptions is equal to $k \times (w - 1)$.

The data field is set to be the plaintext data and is then encrypted symmetrically under the same symmetric key generated for the hybrid headers field. This data field must be the same size for every generic data packet that passes through the network or the packet's length will allow it to be recognised. The length must be pre-agreed by the network peers. The data field can easily be made of the required length by padding the plaintext before encrypting it in addition to splitting the data into multiple packets if required.

The finished packet (see Figure 3.2) is sent to the first peer in the random route that was generated.

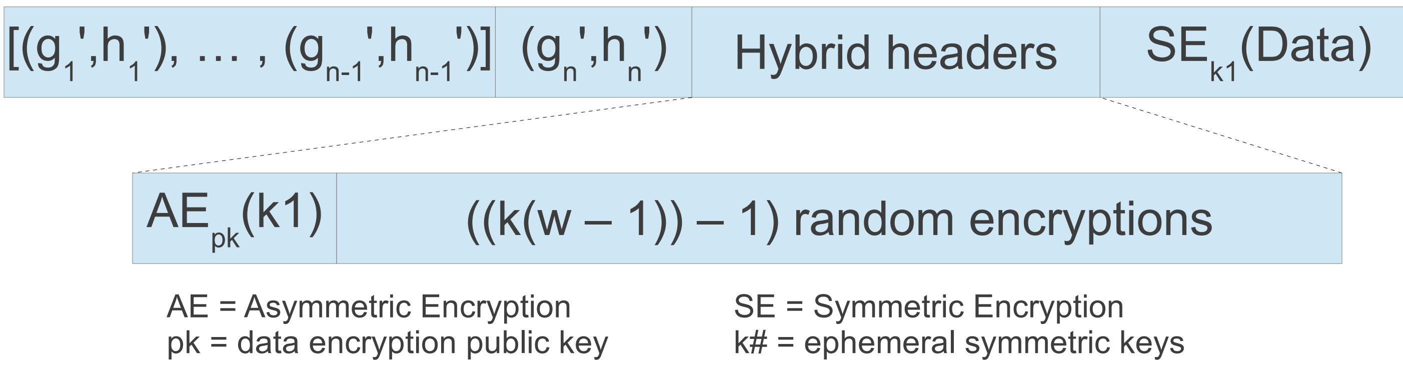
[(g1',h1'), … , (gn-1',hn-1')]
(gn',hn')
Hybrid headers
SEk1(Data)
AEpk(k1)
((k(w – 1)) – 1) random encryptions
AE = Asymmetric Encryption
pk = data encryption public key
SE = Symmetric Encryption
k# = ephemeral symmetric keys

Figure 3.2: The construction of a generic data packet.

Packet Processing En-route

Every peer in the path that a packet takes performs operations on the packet as it passes through. The aim is to transform the packet so that none of its elements can be recognised as being related to their previous values. This will prevent packets being tracked along their paths. Figure 3.3 shows the changes to each field.

The peer checks the first routing key within the routing keys field and determines which direction to send the packet in once processing is complete. The routing key is a pair of $(g\prime, h\prime)$ as described for public key hiding. The peer uses the private part of each of their routing keys to see if $g\prime^{x} = h\prime$. If it does then the packet is to be routed in the direction corresponding to that key. The peer removes this pair from the front of the field and adds a random pair of group members to the back to maintain the field's size. Each pair in the field then has the public key hiding technique applied, this should result in the field being unrecognisable from it's previous value. If the key doesn't correspond to one of its routing keys then the data encryption public key (the second field) is checked to see if the data can be decrypted. If it can, decryption of the packet is attempted, otherwise, the packet is dropped.

The peer will apply the public key hiding technique to the data encryption public key field. The field will then be unrecognisable from it's previous value.

The peer generates a symmetric key for data encryption. The last encryption in the hybrid headers field is removed and the remaining encryptions have a symmetric encryption applied to them using the generated symmetric key (and IV). An encryption of the symmetric key (and IV) under the data encryption public key is added to the front of the field. The field's length is maintained and all of the field's elements are unrecognisable from their previous values.

The data field then is then encrypted under the generated symmetric key. This will make the field unrecognisable as its previous value.

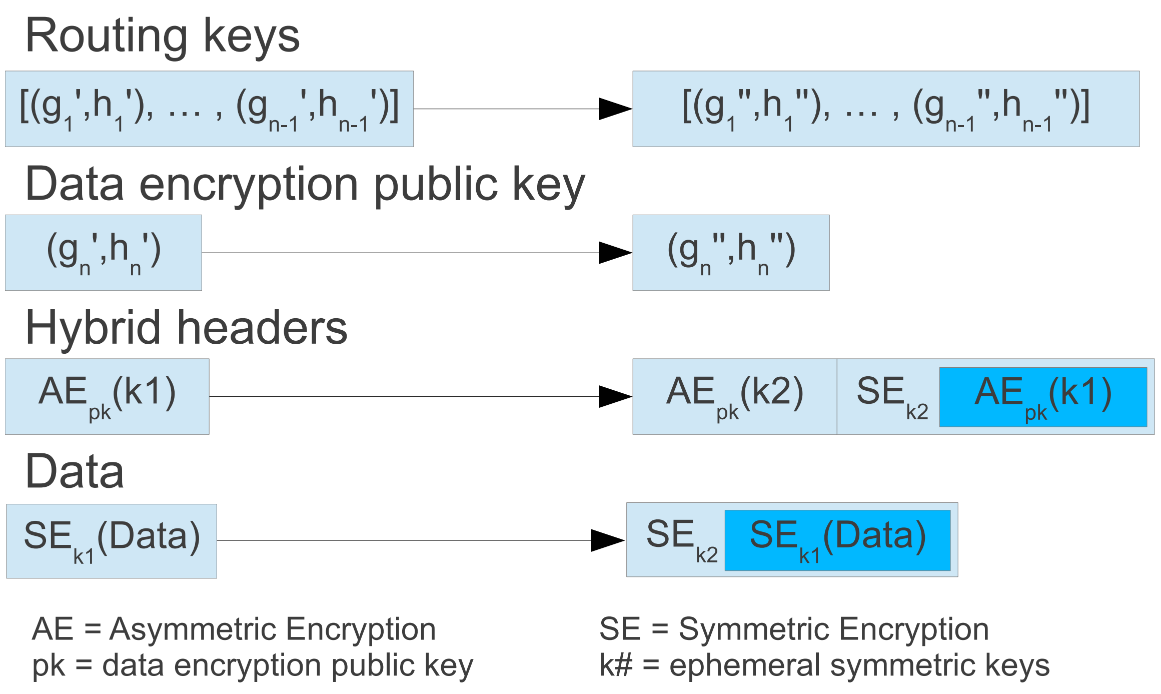
Routing keys
[(g1',h1'), … , (gn-1',hn-1')]
[(g1",h1"), … , (gn-1",hn-1")]
Data encryption public key
(gn',hn')
(gn",hn")
Hybrid headers
AEpk(k1)
AEpk(k2)
SEk2
AEpk(k1)
Data
SEk1(Data)
SEk2
SEk1(Data)
AE = Asymmetric Encryption
pk = data encryption public key
SE = Symmetric Encryption
k# = ephemeral symmetric keys

Figure 3.3: The processing of a generic data packet at an intermediate node.

Packet Decryption

If a peer receives a packet and the first element in the routing key field doesn't correspond to one of their routing public keys then the data encryption public key is inspected. Using the private part of their data encryption key, x, if $g\prime^{x} = h\prime$ then this peer is the packet's destination. The packet's data field may be decrypted as follows. To begin, the first element of the hybrid header's field is decrypted with the private part of the peer's data encryption key; this obtains a symmetric key (and possibly an IV). The remainder of the hybrid headers field and the entire data field is decrypted using this key (and IV). The process may then be repeated on the first element of the remainder of the hybrid headers field; a new symmetric key (and IV) will be obtained with each iteration (see Figure 3.4). The decryption ends when the data field becomes plaintext.

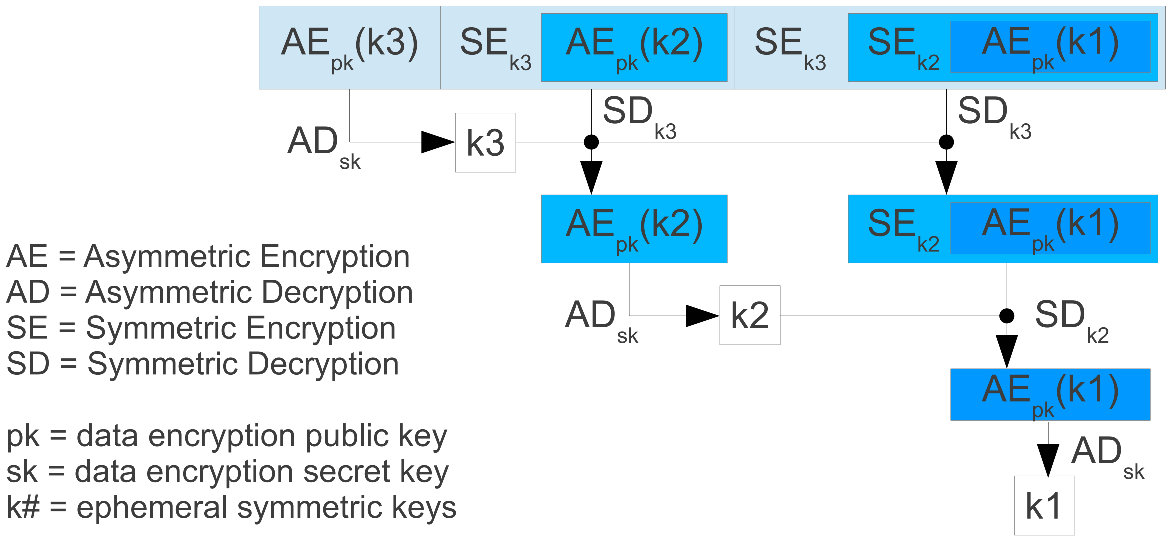
AEpk(k3)
SEk3
AEpk(k2)
SEk3
SEk2
AEpk(k1)
ADsk
k3
SDk3
SDk3
AEpk(k2)
SEk2
AEpk(k1)
ADsk
k2
SDk2
AEpk(k1)
ADsk
k1
AE = Asymmetric Encryption
AD = Asymmetric Decryption
SE = Symmetric Encryption
SD = Symmetric Decryption
pk = data encryption public key
sk = data encryption secret key
k# = ephemeral symmetric keys

Figure 3.4: The process of decrypting the hybrid headers to obtain the sequence of keys used to encrypt the data field.

Choice of Authenticity or Anonymity

The sender may choose to compute a signature on the data using the private part of their signature key. If they do, they will have to identify themselves to the receiver so that the receiver can verify the signature. Quite clearly, the sender cannot remain anonymous and authenticate data they send as authentication is a function that requires the identity of the sender. Here, "anonymity" is referring to how peers are anonymised amongst all of the peers in the network rather than being anonymised amongst a set of IP addresses; if this form of anonymity is broken then only the sender's identity within in the network is revealed instead of their real IP address.

A peer can send these packets to another peer *and receive replies* anonymously as follows. The data that the sender sends is left unsigned, the peer's identifier is left out. As long as the data otherwise contains no identification of the peer, the packet is anonymous. No information about the sender is required to deliver this packet type to the receiver. If the sender requires a reply, they need not provide information on their location, instead they perform source routing from the receiver back to them. This is possible as they have complete network knowledge. The sender will then have a sequence of direction keys that would guide a packet from the receiver back to the sender if the receiver used this sequence as the routing keys field. This sequence is provided as part of the data field in the packet. The receiver cannot determine the sender's location in the network from the routing keys, in order to send a packet back to the sender, they simply use the provided sequence as the packet's routing keys field. Additionally, the sender needs to provide their data encryption public key. To do this, the sender hides the public key and adds it to the data field.

Traffic Analysis Considerations

If an intermediate peer in the generic data packet's transmission were to send every packet onwards (after they have been processed) in the order they had been received then the peer would provide no anonymity. Even though the packet is unrecognisable, it is known which packet that entered the peer it corresponds to. Obviously then, the order the packets leave the peers in should be randomised. A method for doing this is making the peer have a "mix" element. The method shadow P2P uses is known as "Stop and Go", this is described in [16]. It works as follows, the peer takes in a number of messages, processes them and assigns each one a random delay. Once the delay expires, the packet is sent onwards. The effectiveness of this mixing strategy is discussed in the next chapter.

Even with this mix, there may not be very much traffic on the network at a given point in time. This can result in the mix not providing any anonymity as a single packet being received, processed, delayed and sent onwards by a peer will allow an observer to determine that the only output packet corresponds to the only input packet. To combat this, the creation of dummy packets is allowed. These are made to be indistinguishable from generic data packets but contain no data. Furthermore, they should only be sent to the immediate neighbours of the peer creating them to prevent dummy packets clogging up the network with useless traffic.

3.3.2 Packets for making shout groups

Here, the procedure and packets used for creating a shout group from existing peers in the network is described. It is assumed that the node making the shout group has already created its keyset and has found information to contact a peer within the network.

Shout Group Request

The joining node asks a peer in the network (via the shout obtained previously) to broadcast this packet which requests the creation of a shout group (see Figure 3.5). For convenience, we call this peer the "joiner peer". It contains the peer's identifier, the joining nodes's keyset and a timestamp. In order for the joiner peer to be able to contact the joining node, the joining node must provide its shout list.

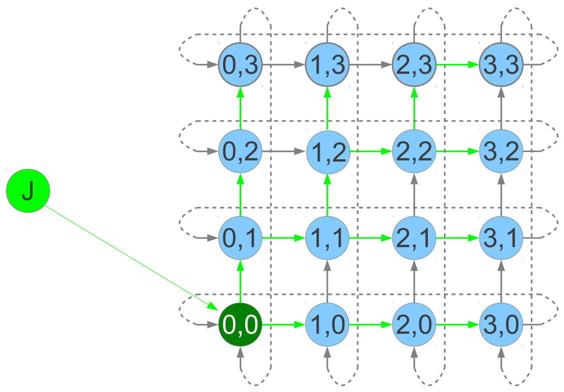

Figure 3.5: The joining node (light green) sends a shout group request packet to the joiner peer (dark green) which then broadcasts it.

Shout Group Volunteer

Peers then offer to take part in the shout group by sending this packet back to the joining node by sending it via the joiner peer which then sends the packet on to the joining node using its shout list (see Figure 3.6). In this packet the peer sends their keyset.

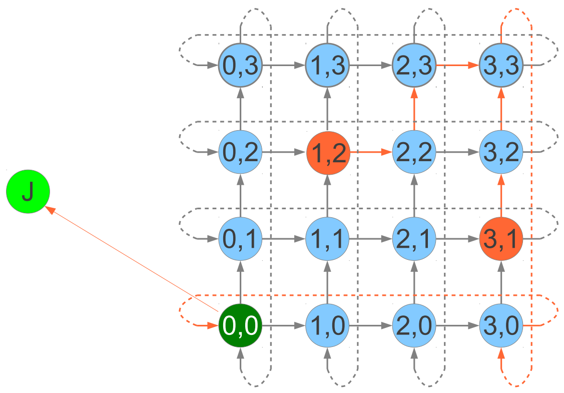

Figure 3.6: The volunteering peers send a shout group volunteer packet to the joiner peer which then passes them on to the joining node.

Shout Group Setup

The joining node will then choose a random subset of the peers that it received volunteer packets from to be members of the shout group. The user will choose a random permutation of these peers and append the joiner peer to the list.

The packet is then created with a list of peer identifiers. This list is created by taking each element from the permutation created previously and appending its identifier to this list. The list should end with the joiner peer's identifier. This packet is then sent to the joiner peer which will then remove the first peer's identifier from the front of the list and send it to that peer (see Figure 3.7).

When each peer receives the packet, they make encryptions of each of the IP addresses in their shout list under the joining node's public key. These are added to the packet's list of encrypted IP addresses and then the entire list is randomly shuffled. The next peer's identifier is then removed from the front of the list and then the packet is sent to that peer.

When the peer reaches the joiner peer again, it is sent to the joining node. The joining node decrypts each IP address in turn in order to obtain the final shout list.

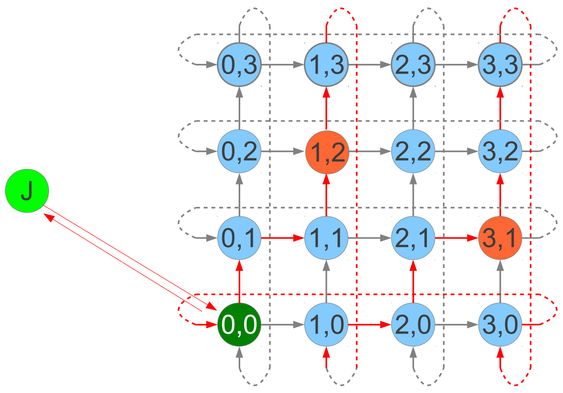

Figure 3.7: The joining node sends a shout group setup packet round in a path that ends back at itself.

Shout Group Verify

The last step in this process is to distribute the list and confirm that no unauthorised changes to the list have occurred. The joining node produces a signature on the list and then constructs a packet with this signature and the decrypted shout list. This packet is then duplicated for each other peer in the shout group. The data field of each of these packets is then encrypted under a different ephemeral key and this ephemeral key is encrypted with the recipient's public key. Each packet is now only readable by the intended recipient. The packets are then shouted using the shout list that has just been constructed.

Upon receiving this packet, the peer tries to decrypt it. If it can, it verifies the signature. Then it checks to see if the list contains all of the IP addresses in its own shout list. If it does, it signs the list with the appropriate private key and sends the signature to every member in the shout group using the shout list it has just received (see Figure 3.8). If all the peers in the shout group receive signatures from every other peer then the shout group has been created successfully. If the signatures are not all received after a period of time, the shout group creation fails and is re-attempted by the joining node, if desired.

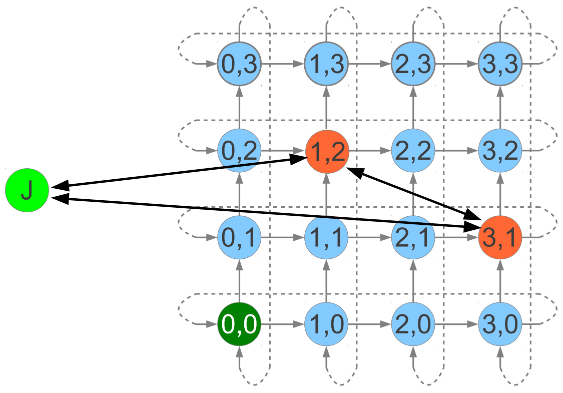

Figure 3.8: The shout group nodes send signatures to each other.

3.3.3 Issues with shout group creation

A risk of using this method to create a shout group is that the shout list the joining node provides to the joiner peer does make it more vulnerable to a search attack on their shout list by the joiner peer. However, the risk is minimal. For the shout group creation, the only response that the joining node gives to the joiner peer is the shout group setup packet. To prevent any search occurring here, the joining node should only wait for a short period of time for the volunteer packets after which it should abandon the creation process and choose a new joiner node. Additionally, it should wait for a substantial period of time between receiving the volunteer packets and sending the setup packet. This severely limits the information that the joiner peer can gain about the joining node's identity. To prevent any search once the joining node becomes a peer, it is sufficient to make sure that the joiner peer is not a member of the shout group that is created.

There is also a risk to the volunteers. As the network requires that a peer use the same set of IP addresses for their individual shout list, the shout groups that the volunteer is part of will have shout lists which have non-negligibly sized intersections. If an adversary managed to obtain the shout lists for two shout group's with a common volunteer, then the intersection would give the adversary the individual shout list of the volunteer. This list is useless on its own as a node will not respond to a message that is not receipted by their shout group, however, it does give an adversary additional information when attacking the shout group. If the attacker always uses this intersection in the subset they send messages to, then the attacker can attach the shout group as if it had parameters of $t = t - 1$, $|V| = |V| - 1$ and $|L| = |L| - |I|$ (where I is the intersection of the lists). With more intersections, the more the adversary can manipulate the shout group into revealing the identity of a peer.

3.3.4 Packets for joining

Join Request

This packet is sent from the joining node to a peer in the network and contains details of the joining node's keyset and shout list. The shout list is that of the shout group which the joining node has already formed, either through the method described for creating one from existing peers or from mutually trusted nodes. The peer can either give the joining node one of its virtual positions or it can select a peer in the network that should give the joining node one of its positions. In the first case, the joiner peer broadcasts a packet transferring a virtual position to the joining node. In the second case, the joining packet is encapsulated within a generic data packet and sent to the peer selected to be the joiner peer. This 'passing-the-buck' behaviour is used to improve network balance as much as possible; as a peer is better off giving away its virtual positions then the selected joiner peer has no reason to refuse this request.

In addition, this packet also contains a signature by the joining node of its shout list. This will be used to prove to the joining node's upstream neighbours that the shout list they will use to contact it is authentic.

Join Response

Once the selected joiner peer receives the joining packet, they give a virtual position to the joining node by creating a certificate of this transfer. The keyset info from the joining node and the virtual position transfer certificate are then broadcast to the network via the associated packets.

The joiner peer will then inform the joining node of its new position in the network (which will become its local root). As the joiner peer has been in the virtual position being given, the peer knows the shout lists needed to contact the downstream neighbours of that position. These shout lists are also sent to the joining node so that it can send packets downstream.

The upstream peers of the joining node's virtual position will hear of the change in ownership from the certificate broadcast and will need the joining node's shout list in order to send packets to them. To facilitate this, the joiner peer, which knows the joining node's shout list, will send the list and its signature to the upstream peers via the appropriate packet. The joining node will then need to get all keysets and all virtual position transfer certificates in order to construct its own complete network knowledge. Again, using the appropriate packet, the joiner peer can provide all the necessary information.

3.3.5 Shout List Change

This packet contains the shout list of a peer or joining node. It is sent to the upstream peers encapsulated in a generic data packet; the data field is authenticated by the owner of the shout list. The upstream peers will verify the signature of the data field and will begin using the shout list as their new downstream connection. The connection replaces the previous downstream connection that these peers had to the joiner peer. No data is sent down these old connections once the upstream peers receives the certificate informing them of the change in their downstream neighbour.

3.3.6 Packets for Keyset Info

These packets are used to distribute information on the keysets of peers. There is a packet for broadcasting keysets, a packet for requesting keysets and a packet for delivering keysets to specific peers.

Broadcast

Upon receiving this packet type, the proof-of-work and signature on the keyset are checked for correctness. If the signature or proof-of-work is not valid then the packet is discarded immediately. Otherwise, the information is added to the peers's knowledge of the network and the packet is resent as the broadcast requires. The checks on the data prevent flooding attacks from packets with unseen data as an attacker would have to generate many different proofs-of-work making it very difficult to do. To prevent flooding attacks with existing peer data, the network simply will not continue the broadcast of anything that has been seen before.

3.3.7 Request

This packet is used for requesting a specific keyset for a given virtual position in the network. The packet isn't broadcast as every peer is expected to have the information requested (and so would result in a flood of responses), however, nor is it directed at any given single peer as that peer has a small chance of not having the information. The packet will instead take a random walk through the network's virtual position grid; a random direction to travel is chosen at each position and the packet is resent. This occurs until it is received by a peer that has the correct keyset.

3.3.8 Direct

This packet is used to give a specified peer a specific keyset. It can be used as either a response to a request or it can be used by a joiner peer to help build a joining node's complete network knowledge. As with the broadcast packet, the proof-of-work and signature are checked for correctness.

3.3.9 Packets for Virtual Position Certificates

These packets contain certificates of one peer giving a virtual position to another. It has the same three packet types as in section 3.3.6 and these all bahave in an identical manner to the previous descriptions. However, where correctness on proofs-of-work and signatures occurs, the certificates are instead checked for correctness. The certificate is deemed incorrect if its signature is invalid or the signing peer is not permitted to give that virtual position.

Chapter 4

Critical Evaluation

Having described the network in full, its components will be evaluated individually. A complete implementation of the network has not been created. Such an implementation is unnecessary since the components are designed to be combined with one another without interference. Additionally, performing IP spoofing on the Internet is against the terms of service agreement of most ISPs and, on the scale that the IP spoofing would have been performed, it would also be of questionable legality. A simulator was initially sought after for the purpose of creating and testing an implementation, however, despite the abundance of network simulators, none were appropriate. NS3[18] and LANSim[17] were briefly trialled as they were the closest simulators to what was desired but neither provided the desired level of functionality. Therefore, the individual components will be analysed in their individual capacities, using appropriate theoretical and / or practical methods.

Here the metrics of anonymity come into play. For the majority of this analysis, the intuitive metric described in [19] will suffice. In this metric, peers are more anonymous when amongst greater numbers of peers and when they behave more like every other peer. As with the age old saying, "When in Rome, do as the Romans do". Later, the entropic metric described in [21] is used to evaluate anonymity in the case of mix networks, which is what the metric was designed to apply to.

4.1 Attacks on Shout Groups

At first glance, it may appear that a measure of anonymity is appropriate to analyse the effectiveness of shout groups. However, this is not the case because the goal of a shout group is not to provide anonymity but to *protect* it. The anonymity is provided by the shout mechanism. The goal of the shout group is to prevent this anonymity being broken by an adversary, so here we analyse the difficulty of breaking this protection.

Recall that a shout group protects anonymity through the prevention of searching the associated shout list. Therefore it is judged how well these methods prevent this search by examining the cost to an adversary performing an attack on a shout group.

4.1.1 Search cut-off

The adversary, A, sends a message to a set of addresses, S, testing whether or not they get a response from the shout group B. They aim to give an $x \in L$ such that $x \in V$ where L is the complete shout list for the shout group and V is the set of true IP addresses belonging to members of the shout group. A gets cut-off by B at some point in its search, at this point it records S; A must collect many such subsets of L and determine from them a probable member of V.

First, it is claimed that if A were to split a random ordering of L in half, they can expect the number of members of V contained in this split to be binomially distributed over many trials. It is quite clear that these trials fit the description required for a binomial distribution. The probability of a success in each Bernoulli trial, p, is $\frac{1}{2}$ as each address has even chance of being in either half of the random split. Simulating 100,000 of these random splits on a shout list of length 10,000 with 100 members of V obtains the result in Figure 4.1.1; this is quite clearly a binomial distribution.

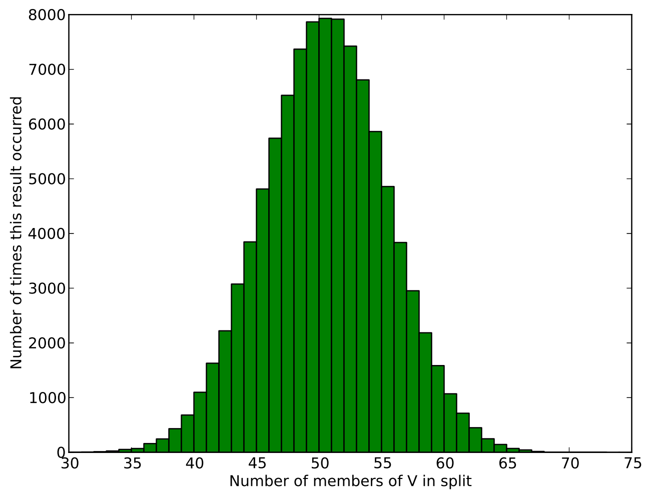

Figure 4.1: The distribution of members of V in the random splits

In order for B to protect its members identities, it should perform the cut-off event such that A gains as little information as possible. Ideally, the distribution of the S results should be indistinguishable from the distribution of random splits of random permutations of L. Here we simulate the attacks A performs on B and analyse the resulting distribution. B has two strategies it can use: cut-off at some random point after exactly half of V have been removed from S and cut-off at some random point between the removal of t and $|V| - t$ members of V.

No Threshold

This first strategy was simulated over 1,000 attacks on the same shout group. In each attack, S was set to a random permutation of L and B was asked for a response, if it responded, S had one of its members removed. The attack was repeated with this continually shrinking S unti B did not respond, i.e. B had performed a cut-off. The final set S was then recorded. B was set up to collect information on the rate at which addresses that were members of V were being removed from S. Once half of the members of V had been removed from S, B has $\frac{|V|}{2} - 1$ data points representing the number of shouts received between each address being removed. A mean, m, of these results was then calculated and B chose a random number, r, in the range $\{0...m\}$. Once r more shouts had been heard, B stopped responding to shouts.

Once all 1000 S results had been collected, the total number of times any given address appeared in S was tallied. These tallies were then sorted and plotted to obtain Figures 4.2 and 4.3; in each of these, $|V| = 20$. The lower part of each figure shows where in the distribution the members of V lie; a point with a y-coordinate of 1 indicates that the bar in the same x-coordinate on the histogram represents a member of the shout group.

Clearly, the addresses belonging to the members of the shout group have received the lower tallies and some of the shout group members can be identified simply by choosing the addresses with the lowest tallies. **As identification can easily be achieved by an adversary, the shout group should not use this method of selecting a cut-off point.**

Threshold Methods

A threshold, t is chosen to represent the number of members of V that may be removed from S before a random cut-off is chosen. The random cut-off is selected so that it should occur before $|V| - t$ members of V have been removed from S. This is done by measuring the number of shouts received between the removal of each member of V. Assuming that one address is removed from S between each shout, the average number of shouts received is computed; this is the estimate of how many shouts occur between the removal of addresses from V. Using this estimate, a value is generated randomly from the binomial with $\mu = \frac{|V|}{2}$ and $\sigma = \frac{|V|}{6}$ and to this is added a uniformly selected random number between zero and the estimate block size. The value of

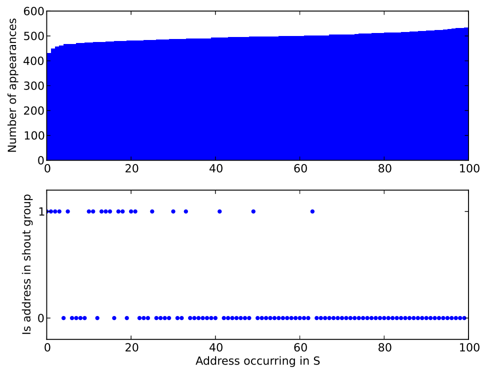

Figure 4.2: $|L| = 100$

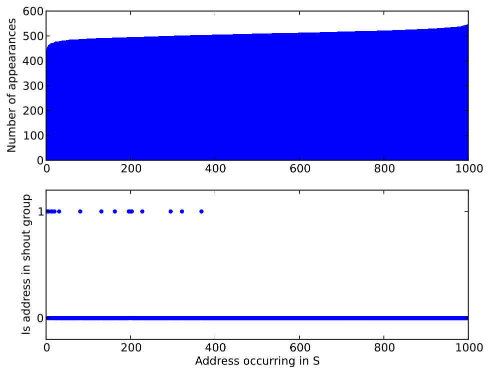

Figure 4.3: $|L| = 1000$

σ is chosen such that nearly all points on the curve fall in the range $[0, |V|]$. If the value generated from the binomial is below t or above $|V| - t$ then the process is repeated to ensure the value falls within the threshold boundaries. This reuslting value is the number of shouts that B will respond to before performing the cut-off. The simulations below are run with $|L| = 200$, $|V| = 20$ and $t = 3$ over 1000 iterations. The results are illustrated in Figure 4.4.

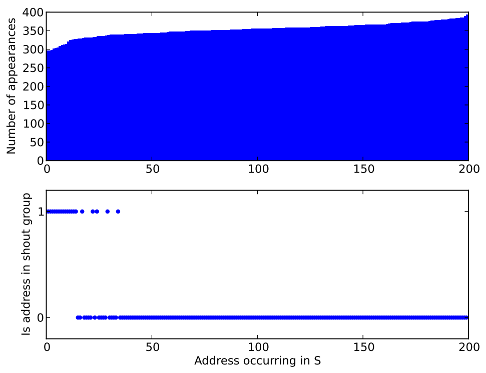

Figure 4.4: Members of V occur in S less frequently than other addresses

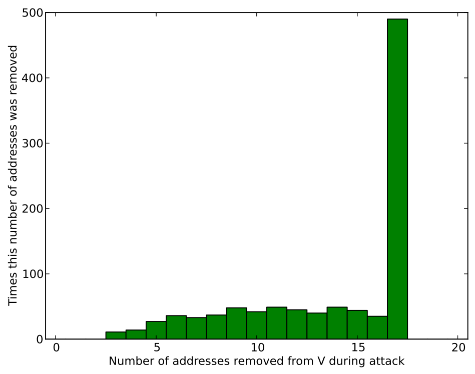

Figure 4.5: 50% of cases trigger the hard cut-off

There is little improvement on anonymity protection with the introduction of thresholds; members of V are still grouped towards the end of the graph with fewer occurrences in S. A probable reason for this is illustrated in Figure 4.5. Here it is shown how many of the members of V have been removed from S by A when the cut-off occurs. In a half of cases, the cut-off does not occur inside the desired range but at the 'hard' cut-off that is set to occur if $|V| - t$ members of V are removed from S. This hard cut-off leaks information about V to A as the last address removed will be a member of V. This case is quite a way from the desirable binomial distribution where the cut-offs will produce different S that are indistinguishable from a randomly chosen cuts of random permutations of L.

In order to find a method to overcome the issue, it is worth looking at the desired case. To illustrate this, we perform the same simulation but allow B to use information about S to pick a cut-off. The cut-offs chosen are generated from binomial and uniform distributions as before except that the knowledge of S is used to guarantee that the cut-off occurs after the removal of exactly the randomly selected number of members of V given by the binomial. Figures 4.6 and 4.7 show the results obtained. These figures show the desirable situation that B can achieve using information that it would never have in reality. In Figure 4.6, the members of V now appear to be distributed uniformly; the number of times an address appears in the variants of S is independent of whether that address is in V.

It should now be quite clear that the rate of search estimation method used initially is not sufficient to protect anonymity. A new method must be conceived if shout groups are to be at all useful. The aim here is to model

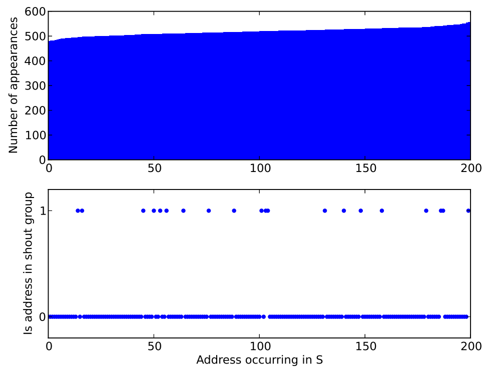

Figure 4.6: Addresses occurrence in S is independent of its membership in V

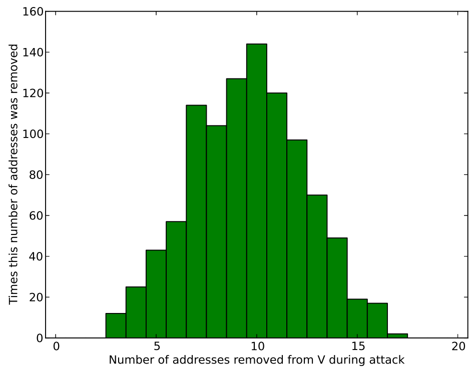

Figure 4.7: The cut-offs that were chosen randomly from the binomial distribution

a random variable, P, that uses only information that B has such that it is indistiguishable from another random variable, Q, which has knowledge of everything that A and B know. P and Q both give the number of shouts that are responded to before a cut-off occurs. As in the desired case, Q is modeled as follows. S is initially set to a random permutation of L, x_i is the position of the i-th member of V in S, q is the cut-off position in S such that $0 <= x_{i-1} <= q < x_i <= |L|$.

$$Pr(Q = q) = \begin{cases} \frac{x_i - x_{i-1}}{|L|} \times \binom{|V|}{i} 0.5^{|V|} & \text{if } t < i < |V| - t \\ 0 & \text{otherwise} \end{cases}$$

P will have to model this, however, it may not use the information concerning S. In Q this information is in the x elements and so a replacement for the expression $x_i - x_{i-1}$ must be found. Previously this was set to be the estimate of the number of shouts received between addresses removed; this was calculated by taking the average number of shouts between address removals. However, this doesn't hold up well in circumstances where the gaps between members of V is larger than average. Multiplying this estimate by the number of unremoved members of V and adding the number of shouts already received should ideally yield $|L|$, however, large values of the average will often cause it to exceed $|L|$. Instead, B can use its knowledge of $|L|$ to fix this behaviour. By subtracting the number of shouts received from $|L|$ and dividing by the number of unremoved members of V, a better estimate is obtained. If the number of shouts before the threshold is reached is larger than average, the estimate becomes smaller and vice-versa. Here, g represents the number of shouts received.

$$Pr(P = p) = \begin{cases} \frac{|L| - g}{|L|} \times \binom{|V|}{i} 0.5^{|V|} & \text{if } t < i < |V| - t \\ 0 & \text{otherwise} \end{cases}$$

We use this better estimate in the simulation. The results in Figure 4.9 shows that there is now a negligible proportion of cases which trigger the hard cut-off, as desired. Figure 4.8 shows a similar result to that in Figure 4.6, the number of cases where an address appears in S seems to be independent of its membership in V.

The method of cut-off selection can be improved yet further, there is still some information that hasn't been utilised by B. So far, a cut-off has always been selected just after the threshold number of members of V havbe been removed from S. Instead, we can update our estimate of the gap between address removals as more addresses are removed. Towards the end of the attack where nearly $|V| - t$ addresses have been removed, the estimate will be more accurate and so the hard cut-off is less likely to be triggered. Simulations with this method of estimation obtain the results shown in Figures 4.10 and 4.11.

Suprisingly, this gives a worse result than before. The members of V are again skewed towards one end of the diagram. The members of V are being removed from S more than the other addresses. Therefore the previous method is currently the best known method to use for selecting cut-offs. In conclusion, although a better threshold method may exist, **the best threshold method used here suffices to prevent attacks** as it already takes 1000 attacks on the shout group to break the anonymity.

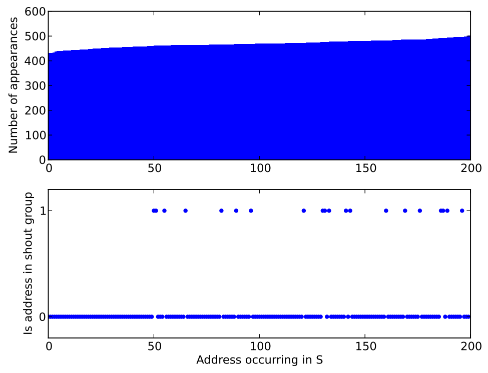

Figure 4.8: Under the better estimate, the variables appear independent

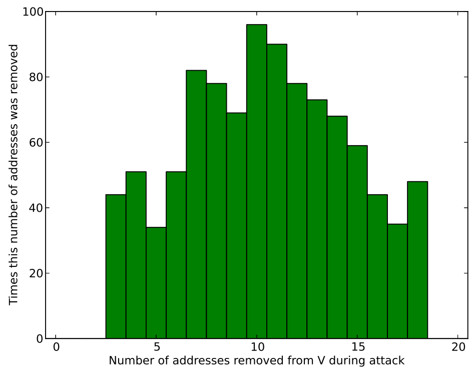

Figure 4.9: Very few cut-offs occur at the hard cut-off.

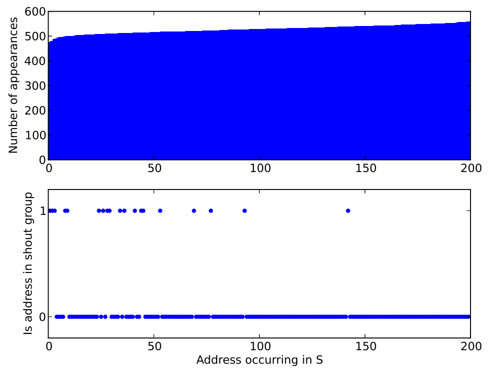

Figure 4.10: The members of V generally have fewer occurrence in S

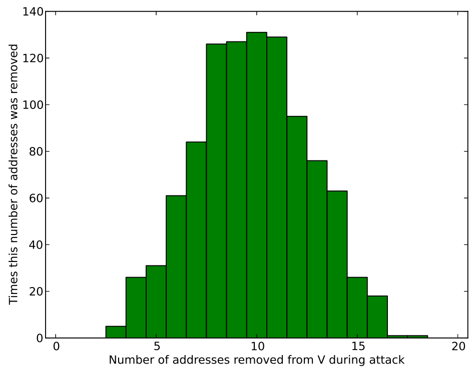

Figure 4.11: A negligible number of cases trigger the hard cut-off

4.1.2 Number of Attacks Required

Although the shout group does provide a high level of protection to the members of V, it will always leak some information about V to A. It is therefore worth seeing how many attacks are likely to be required before A has some level of confidence about which addresses belong to V. For this, the best method from the previous section is used as an increasing number of attacks are compiled against it. A shout group with $|L| = 200$, $|V| = 20$ and $t = 3$ is used for these simulations.

The expectation that any given member of V will appear in S is $\frac{1}{2}$. As an address may either appear or not appear in S, it should be clear that the number of times an address appears in all variants of S that are collected is binomially distributed. In this analysis, the number of times each member of V appears in different S is recorded against how many attacks have been performed against the shout group. From this data, the p-value is calculated from each member of V using the binomial distribution; this obtains a set of p-values for a given number of attacks. For each set, the maximum and mean p-value are computed. These are plotted in Figure 4.1.2.

The graph shows that as the number of attacks increases, the average and maximum p-values decrease. The attacker need only perform the number of attacks that gives them a desired level of confidence. The traditional confidence levels of $p = 0.05$ and $p = 0.01$ have been highlighted. An attacker that seeks to identify all of the shout group members would be interested in where the maximum p-values drop below a given level of confidence. The attacker can then has that level of confidence in the identity of every single shout group member. For a $p = 0.05$ level of confidence, the attacker needs to perform 8,000 attacks; at $p = 0.01$, 10,000 attacks are required.

If the attacker is interested in identifying just a single member of the shout group, they would be interested in the minimum p-value. Figure 4.1.2 plots these against the number of attacks performed. This gives

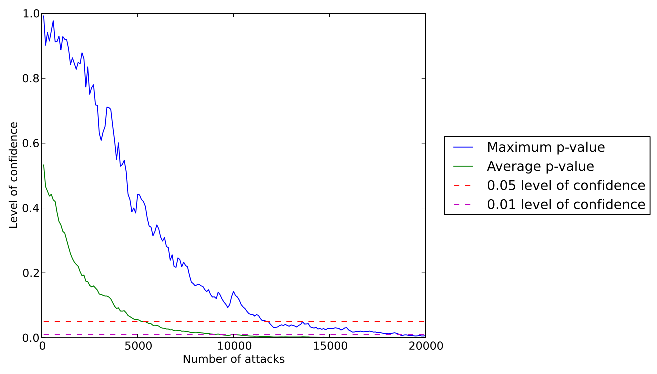

Figure 4.12: The change in mean and maximum p-value as more attacks are performed

discouraging results; just 200 attacks are needed to drop the p-value below the $p = 0.05$ level of confidence. For the $p = 0.01$ level of confidence, roughly 750 attacks are needed. Clearly the shout group does not provide a satisfactory level of protection under these conditions. However, **the values for $|L|$, $|V|$ and t can be chosen so that the shout group provides protection on any level desired.**

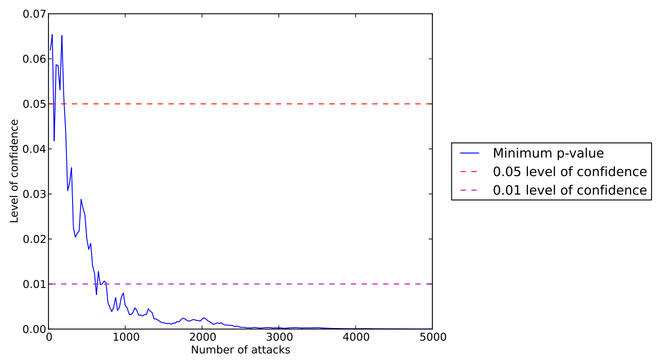

Figure 4.13: The change in minimum p-value as more attacks are performed

4.1.3 Fraction of Shout List and Threshold Parameters

In the previous section it was shown that only a small number of attacks was necessary to identify a single shout group member. However, these were under a fixed set of shout group parameters. In this section, how the level of protection changes with these parameters is investigated. The simulations that follow obtain sets of p-values for each member of V. The mean of these p-values is then calculated from all the simulations performed and this is the p-value plotted. All of the following simulations perform 100 attacks on the shout group.

First, $|L|$ is modified with a fixed $|V|$ and fixed t. $|V| = 20$ and $t = 3$ in these simulations. The result in Figure 4.14 shows that increasing the number of addresses in the shout list increases the p-value but only up until $|L|$ is ten times the size of V. This plateau means that increasing the number of addresses in the shout list further will not increase the level of protection afforded. It is not known why this occurs.

Secondly, $|V|$ is modified with fixed $|L|$ and fixed t. The constants set here are $|L| = 200$ and $t = 3$. Figure 4.15 shows a steep decline in p-value as more addresses become members of V. Where $|V|$ is greater than one half of $|L|$, there will a 50/50 chance of the next address removed from S being a member of V. The cut-off selection process demands that if the next member of V is removed and a cut-off has not been performed then a cut-off immediately occurs. The random selection should either choose to wait one shout or cut-off immediately as the limit of the random selection will tend towards a gap of one address between members of V. When there is no gap between members of V being removed, the random cut-off selection has no effect;

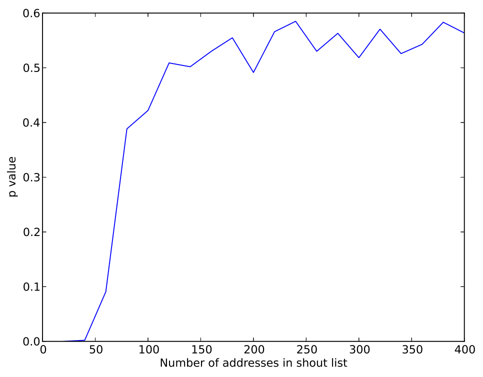

Figure 4.14: The change in p-value as the shout list varies in size

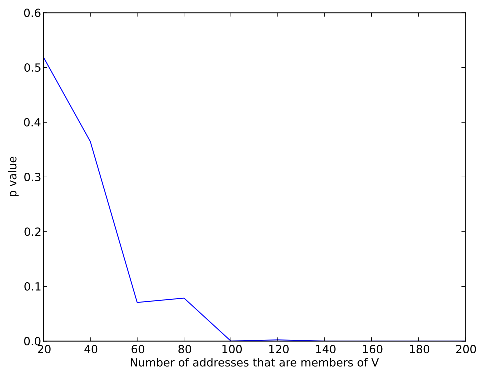

Figure 4.15: The change in p-value as the number of members of V variess

a member of V will always be the last address removed from S. Therefore, members of V are less likely to appear in the remainder of S after the cut-off.

Next, t is modified with fixed $|L|$ and $|V|$. Here $|L| = 200$ and $|V| = 40$. The results are shown in Figure 4.16. Again, it appears there is a plateau that occurs once t is one quarter the size of V. From the simulations performed with no threshold that leave S with exactly half of the members of V, it is known that the members of V can easily be identified. A threshold of $\frac{|V|}{2}$ will produce the same result as it requires exactly half of the members of V to be removed before the cut-off occurs. We may therefore deduce that a threshold of $\frac{|V|}{2}$ has a low p-value. Taking this into account, the best setting for the threshold is where the plateau begins, i.e. $t = \frac{|V|}{4}$.

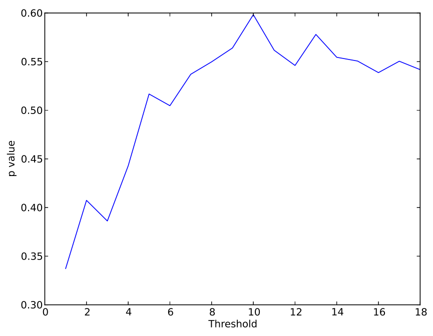

Figure 4.16: The change in p-value as the threshold varies in size

In summary, $|L|$ **should be at least ten times that of** $|V|$ **and** t **should be set to** $\frac{|V|}{4}$. This will yield the best level of protection for the given the parameters. However, this analysis was performed with two of the three variables fixed. **Further analysis should be performed** in order to determine the effects of altering two, or even all three variables at once.

4.2 Public Key Hiding

Here a cryptographic proof of security is presented showing that the public key hiding method is secure.

For the ElGamal case of public key hiding there are some group parameters G, g and q, a public key h, and a private key x. The public key is then hidden by producing the pair $(g\prime = g^r,\ h\prime = h^r)$ where r is a randomly selected from 1 ... q. Breaking the anonymity of this system involves finding h given G, g, q, $g\prime$ and $h\prime$.

Consider the following adversary, A. A can break the anonymity of the public key hiding by taking G, g, q,

$g\prime$ and $h\prime$ and producing h. Consider also B, which uses A to break the Decisional Diffie Hellman (DDH) problem. They are arranged as in Figure 4.17

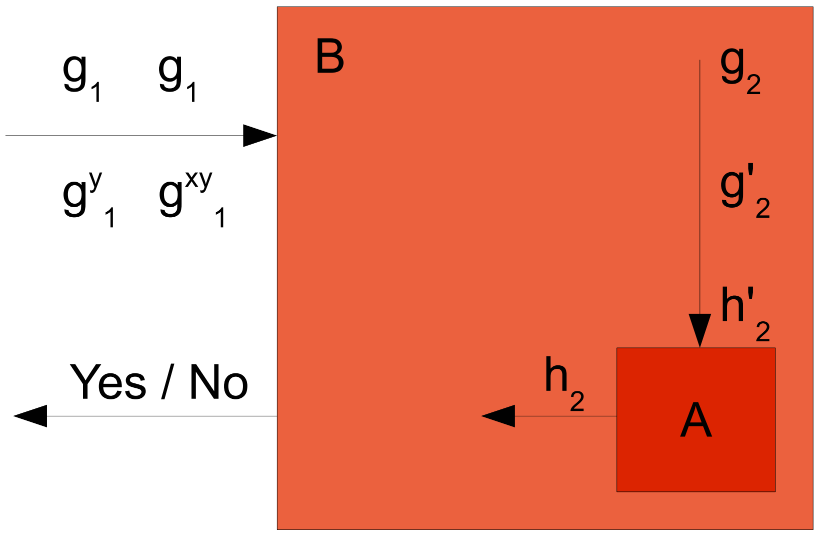

Figure 4.17: B uses A to solve DDH

B uses A as follows, it sets g_2 to g_1, $g\prime_2$ to g_1^y and $h\prime_2$ to g_1^{xy}. A returns h_2, if $h_2 = g_1^x$ then B returns "Yes", otherwise "No". Therefore, if the anonymity of the public key hiding can be broken, so too can DDH. **Therefore this public key hiding scheme is as at least as hard to solve as DDH.**

4.3 Network Structure

4.3.1 Simulation

I built a proof-of-concept simulation of the network structure. In this section, it is analysed to determine whether network expansion, network balancing, source routing, virtual address allocation and complete network knowledge all function as intended. The simulator simulates the nodes, the packets they send to other nodes and the connections they make between them. The simulator does not simulatate the shout mechanism or shout groups and no cryptographic element is implemented.

The simulator is designed to simulate the network components following the description of the network laid out in the previous chapter. To analyse it as a proof of concept, a scenario is conceived and then analysed to ascertain what the network *should* do in that situation. The simulator is then presented with the same scenario and then it is seen if it produces the predicted behaviour. If it does then it proves that the simulator is an accurate implementation of the network design and that the network design is a feasible network.

4.3.2 Network Expansion

In this simulation, the network starts as a single peer (the root peer) and then 7 additional nodes join the network. Each node contacts the root peer with its join request. From the network setup, the network in the simulation should undergo the following sequence of events.

The root peer (node #1) will receive a join request from the first joining node (node #2). Node #1 will see that the network is only big enough for one peer so it will perform an expansion, placing itself at all the new positions created. Node #1 will then create a certificate giving one of these newly created virtual positions to node #2; this certificate is sent to node #2 in the join response. Usually, the broadcast of this certificate would occur at the same time, however, as node #1 is the only peer in the network, nothing happens. Node #2 will then verify the certificate and then it will be part of the network. Node #1 then sends a direct virtual position certificate packet for its own position; this is a self-signed certificate allocating Node #1 the root peer.

Nodes #3 and #4 will join in the same manner, each receiving a certificate in their join responses from node #1. This time around, these certificates will be broadcasted to the network so that the other peers can update their complete network knowledges. Nodes #3 and #4 verify their certificates and become part of the network. Node #1 will send its network knowledge to each of the joining nodes so that they may begin life in the network with knowledge that is as complete as possible.

As node #5 joins, node #1 will see that there is no longer any room in the network for new peers, it will therefore conduct an expansion. It copies its own knowledge of the network layout so that it will obtains a layout that is four times its original size. Nodes #1, #2, #3 and #4 will all appear 4 times in this increased layout in positions dictated by the toroidal nature of the network structure. There will then be plenty of room for node #5. Node #1 will then create a certificate, broadcast it and send it to node #5 which will then verify the certificate and become part of the network. It is then sent network knowledge from node #1. Nodes #6 and #7 also join in this manner.

As node #8 joins, node #1 will have run out of virtual positions to give for the current level of network expansion. However, expanding the network will unbalance the number of connections per node because nodes #2, #3 and #4 will still own multiple virtual positions in the latest expansion. It would therefore be best if one of those peers gave node #8 a virtual position. To do this, node #1 will encapsulate the joining packet it will have received from node #8 and sends it to either of nodes #2, #3 and #4. Once the peer receives this encapsulation, it will join node #8 to the network in the same way. The certificate will be created, broadcasted and sent to node #8 and node #8 will verify it and become part of the network. The selected joiner peer will then send node #8 its own network knowledge.

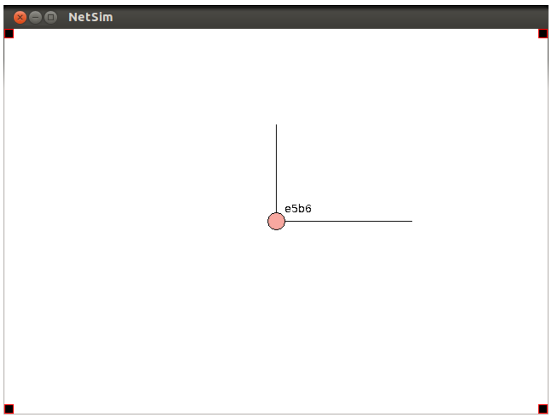

Figure 4.18: Initial network state.

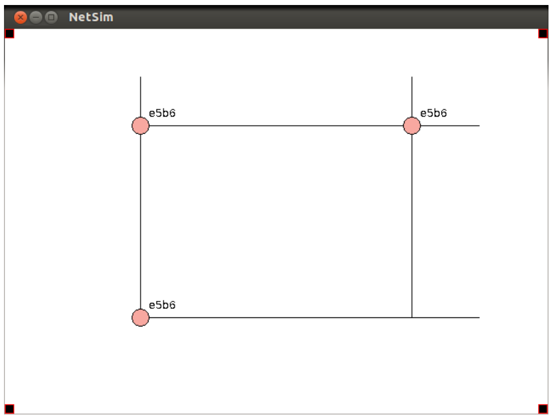

Figure 4.19: After first expansion.

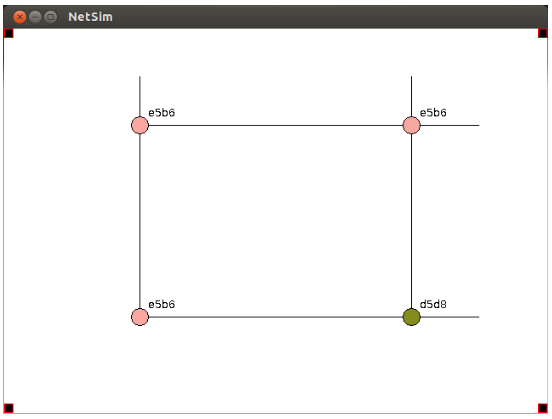

Figure 4.20: Node #2 joins the network.

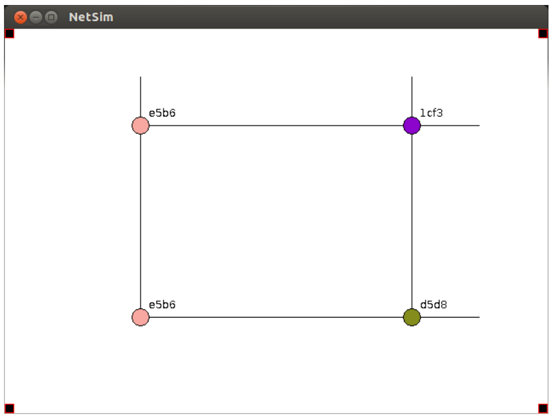

Figure 4.21: Node #3 joins the network.

A walkthrough of the simulation now follows with screenshots from the simulator as appropriate. Figure 4.18 shows the first state in the simulation, the network consists of one peer (node #1). The lines represent the uni-directional links to the other peers in the network (of which there are currently none) and the four hexadecimal digits are the first 4 hexidecimal digits of the node's Universally Unique Identifier (UUID). A node #2 then sends its join request to the root peer. Node #1 notes that the network requires expansion so it

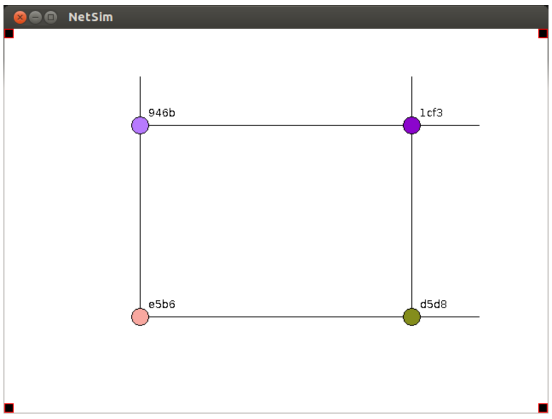

Figure 4.22: Node #4 joins the network.

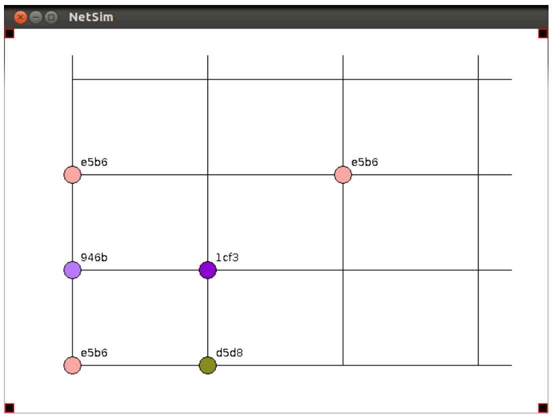

Figure 4.23: The second expansion.

does this. Figure 4.19 shows the expanded network, however, it has not placed itself in the position $(1, 0)$ as it is currently giving this position to node #2. Once node #2 has received the certificate, it joins the network in its assigned position as in Figure 4.20. Nodes #3 and #4 both join in a similar manner, shown in Figures 4.21 and 4.22 respectively.

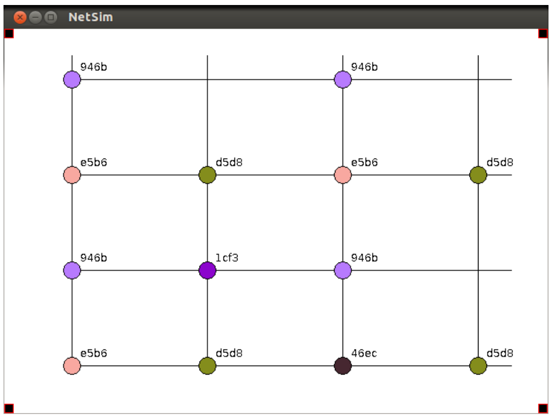

Figure 4.24: Node #5 joins the network and the certificate broadcast filter through.

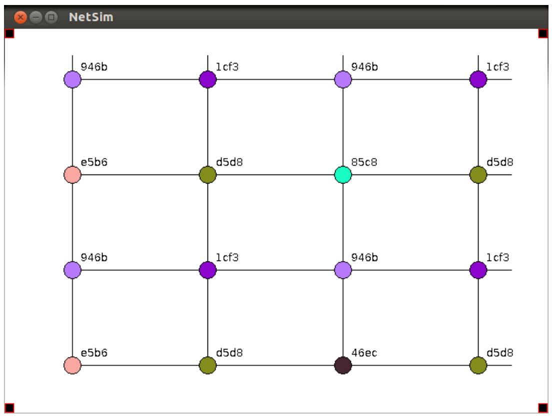

Figure 4.25: Node #6 joins the network.

When node #5 attempts to join, it causes another expansion as shown in Figure 4.23. Again, node #1 has not taken all of its virtual positions as it is giving one of the positions to node #5. None of the other peers have taken up their virtual positions yet as they are unaware that the network has been expanded. Node #5 completes its join in Figure 4.24. Note how the broadcasts of node #5's certificate have caused the peers to take up their virtual positions in the expanded network. This is because the peers learn that new peers have been joined beyond the current known boundary of the network, so they correctly conclude that the network is of a bigger size. Note that when any peer expands the network or notes that it has expanded, all the peer does is double the variable representing the size of the network in their network knowledge, no transmission needs to occur. As the simulation will only display a peer in a virtual position if that peer claims to be in that virtual position, the position appears empty until the broadcast filters through the network. Nodes #6 and #7 join the network in the same way as before as shown in Figures 4.25 and 4.26.

For node #8 to join the network, it is encapsulated within a generic data packet and sent to a peer with virtual positions available. Either of nodes #2, #3 or #4 would do, in this case it happens to be node #4 that is chosen. Node #4, upon receiving the message, chooses one of the virtual positions that it owns and then creates, broadcasts and sends the certificate to node #8. Node #8 joins the network at position $(2, 1)$. As the packet encapsulation and transmission was a success, it is clear that the complete network knowledge mechanisms are working to the required extent. Note that, in the simulator, there is no "packet loss" i.e. all packets are guranteed to arrive at their destination. In reality, the information request packets would be used frequently to gather complete network knowledge as required.

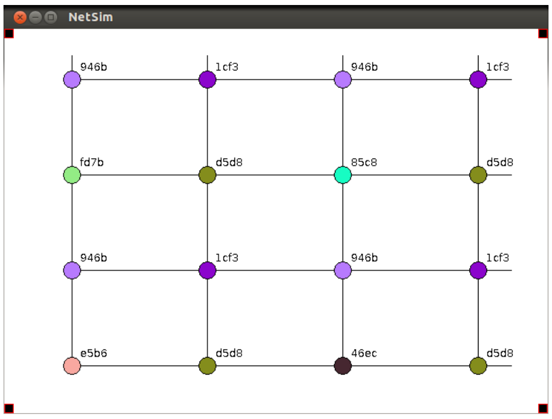

Figure 4.26: Node #7 joins the network.

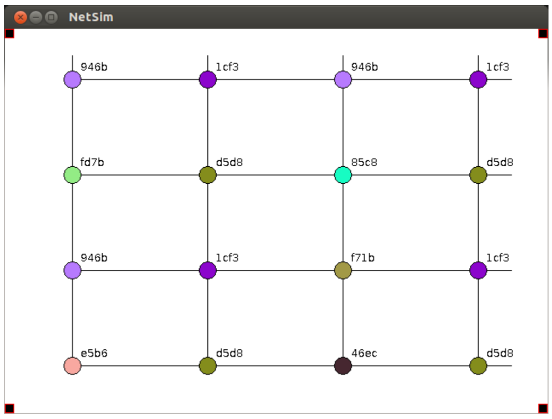

Figure 4.27: Node #8 joins the network after the join request is encapsulated and sent to a new joiner.

4.4 Network Balance

In this simulation, the network starts with 4 peers connected to it. Another node joins the network and then one of the original peers leaves. The new peer attempts to balance the network by assuming the position of the peer that left. The simulation should progress as per the following description.

Nodes #1, #2, #3 and #4 will already be in the network. Node #5 will then join and cause an expansion. Node #3 will then leave the network; to do this it returns its virtual position to the peer owning the parent virtual position which happens to be node #1. It will do this by creating and broadcasting a certificate to the network. Unlike when certificates are recorded in a peer's network knowledge from joining the network, this certificate will cause the certificate granting its virtual position to be erased. Control of node #3's virtual positions will revert to node #1.

At some point, node #5 will inspect its network knowledge and notice that it would be better placed where node #3 used to be. Therefore it will issue a balance request to node #1, which will own the positions in question at that point. This balance request will be encapsulated in a generic data packet and sent. Upon receiving the balance request, node #1 will inspect its network knowledge for a virtual position it owns that is in a better than the one node #5 will have as its local root at that point. Once it finds a position, it creates a certificate that will allow node #5 to move to that position. This will be encapsulated and sent to node #5. Node #5 will then leave its position by creating a certificate like it would as if it were leaving the network. It will then broadcast this certificate and the one it received from node #1. This will put node #5 in the position that #3 used to own. The position that node #5 used to be in reverts to the control of node #1. The peers in the network will then notice that the network is in a state where it can be contracted and so a contraction will be performed by all peers in the network. The resulting network will then be nearly identical to its initial state except that node #5 will be in the place of node #3.

The following screenshots show how the simulator behaves in this scenario. The simulation starts as in Figure 4.28 and then a new node (node #5) joins the network, causing an exapnsion. Once all the certifcate broadcasts have finished, the simulated network is in the expanded state shown in Figure 4.29.

Node #3 proceeds to leave the network by broadcasting the certificate it creates. Once node #1 obtains this certificate, it places itself at the positions that node #3 vacated. This is shown in Figure 4.30. After some time passes in the simulator, node #5 comes to realise it is better place where node #3 was and so sends an encapsulated balance request to node #1. When node #1 receives this request, it creates a certificate for the position node #3 used to own and removes itself from those positions leaving the network in the state shown in Figure 4.31. The certificate is then sent encapsulated back to node #5.

Once node #5 receives the encapsulated certificate, it creates a certificate for leaving the local root that it currently owns and then broadcasts both certificates. This causes it to enter the positions that node #3 used to own whilst leaving the point that it used to own. This is shown in Figure 4.32. Once node #1 receives the certificate from node #5 that returns its virtual position to node #1, node #1 enters the position as shown in Figure 4.33. Finally, as the peers receive node #5's certificate returning its virtual position, they note that

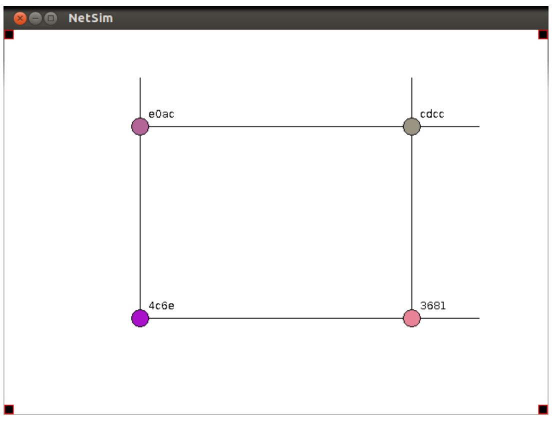

Figure 4.28: Initial network state.

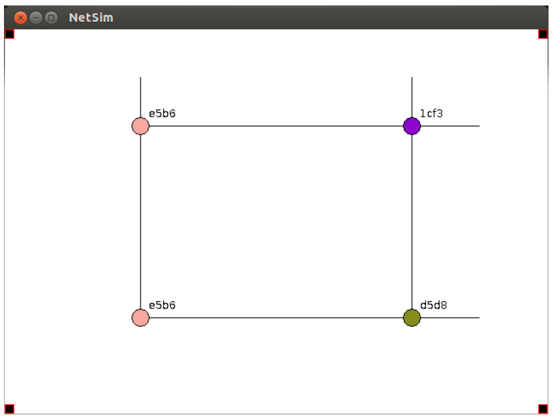

Figure 4.29: Node #5 joins the network, causing an expansion.

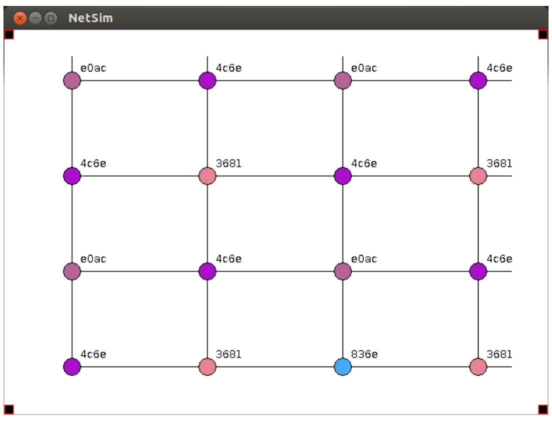

Figure 4.30: Node #3 leaves the network and node #1 obtains its virtual positions.

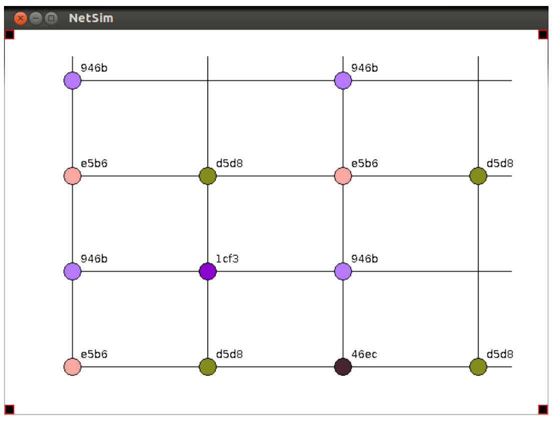

Figure 4.31: Node #1 makes a certificate to give node #5

the network is in a state where it can be contracted, therefore they do so. This causes the network to enter its final contracted state as shown in Figure 4.34. This is practically identical to the network's initial state except that node #5 replaces node #3. Clearly, the simulation has behaved as expected showing that the technique works.

4.5 Anonymity Provided by the Network Structure

To analyse the anonymity properties of the toridal structure, the network is first analysed as a mix network and then as a heat map.

4.6 As a mix network

A mix network is a network consisting of senders, receivers and intermediate 'mix' nodes. Senders send messages through the mix network to receivers. The mix network consists of a number of mixes through which messages pass. Each mix disguises the messages it receives to make them unrecognisable and reorders the messages randomly before sending them onwards. The sender gains anonymity from the mix network because for each mix the message passes through, the set of possible senders of a message becomes the union of the current set of possible senders and the set of possible senders sending messages to that mix.

Shadow P2P is arranged as a mix network. Each peer in the toroid plays the role of a mix and is a possible sender and receiver. Its properties of anonymity are now analysed using the metric described in [21]. The

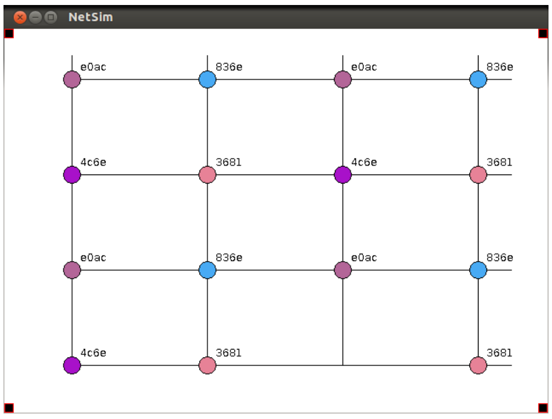

Figure 4.32: Node #5 broadcasts two certificates to move to a better virtual position.

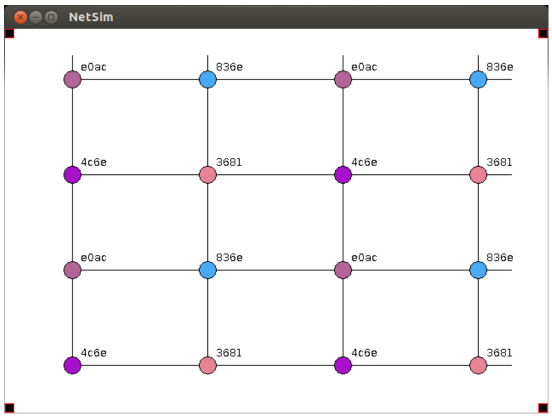

Figure 4.33: Node #1 gets node #5's old virtual position.

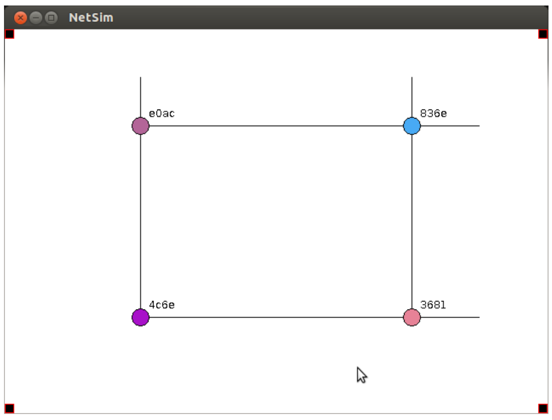

Figure 4.34: The peers in the network perform a contraction.

example in Figure 4.35 shows the possible senders and receivers of a message that was observed as being received by peer A. The blue lines show possible links that the message may have traversed getting to A and the green lines show links the the packet may traverse if it leaves A.

In the case of the sender, the possibilities are limited by the maximum length of the generic data's routing keys field; a maximum of 2 keys may be used for a network of this size. This limitation stops the links spreading very far. In addition to this, A is excluded from the possible senders as it is assumed that a peer would not send messages to itself. This leaves the possible senders as B, C and D.

According to the metric, the anonymity probability distribution of the links from D to both B and C are $\{(D, 1)\}$. This is because a message that has traversed both a link from D and a link to A must originate at D so the associated probability is 1. For B, it is known that it might have received the packet from D or it might have sent the packet itself. From the equation in section 3.1 of [21] we can deduce that the link $B \rightarrow A$ has the anonymity probability distribution of $\{(B, \frac{1}{2}), (D, \frac{1}{2})\}$. Similarly, C has $\{(B, \frac{1}{2}), (D, \frac{1}{2})\}$. A packet arriving at A then has the distribution $\{(B, \frac{1}{4}), (C, \frac{1}{4}), (D, \frac{1}{2})\}$ for the possible senders of the packet. To this we add probabilities of 0 for every node in the network that does not appear in the distribution; this yeilds $\{(A, 0), (B, \frac{1}{4}), (C, \frac{1}{4}), (D, \frac{1}{2})\}$. From this we can obtain a value for the "effective size" of the distribution using the entropy equation; this comes to 1.5 which is 75% of the possible maximum entropy.

For the receivers of the packet, it is equally likely that either of A, B or C are the final destination but it cannot possibly be D. Therefore the entropy for this comes to 1.58, 79.2% of the maximum. Note that A is included in the calculation as it is possible for A to be the intended recipient of a message that was received by A. It was excluded from the group of senders in the previous calculation because it is assumed that a peer would not send a packet to itself.

This shows that the fact that a peer has received a message reveals some information about its sender and

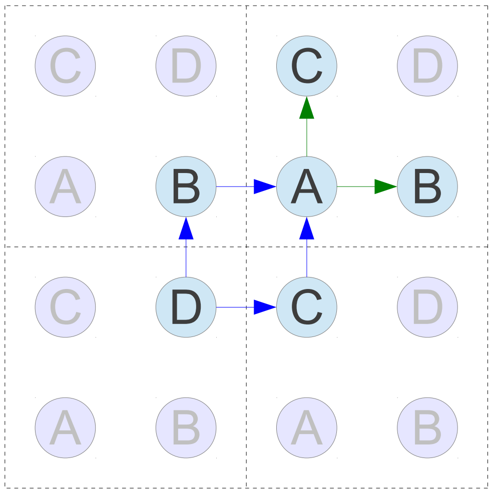

Figure 4.35: The possible senders and receivers of a message received by A.

	Sender		Receiver	
Size	Entropy	% of maximum	Entropy	% of maximum
2	1.5	94.6	1.390	87.7
4	3.642	93.2	3.309	84.7
8	4.321	72.3	3.862	64.6
16	4.366	54.6	3.899	48.8
32	4.366	43.6	3.899	39.0
64	4.366	36.4	3.899	32.5

Table 4.1: Entropies for different network sizes.

receiver (as the entropy of the scenario is not equal to the maximum). Next, the entropies are calculated for larger networks to ascertain whether they provide a higher level of anonymity. The results are shown in Table 4.1. As the network width doubles in each dimension, the level of entropy provided by that size of network quickly plateaus. This is an unexpected result, it would seem more logical that a larger network should gives messages a higher level of anonymity because there are more peers that the message may have come from or might go to.

To explain why this result occurs, consider a case where one peer in the network sends a single message to every other peer. Whilst every peer has the same probability of being the receiver any given message, the peers closest to the peer performing the sending process a much larger proportion of the traffic. Now consider the case where every peer sends one packet to every other peer and consider a single one of these senders again. The closest downstream peers to the sender will receive a much larger proportion of traffic from that one sender when compared with the traffic from every other sender. So although the traffic on the network may be evenly distributed, the sender and receiver of any given packet are more likely to be closer to where that packet was observered than they are to be further away.

The entropy plateaus because the expansion of the network only changes the proportions for peers distant to the sender / receiver. As the more distant peers contribute a very small proportion of traffic to the calculation, the change in the value of entropy calculated is negligible. This does mean that larger networks provide very little improvement to the anonymity of a packet's sender and receiver.

4.6.1 As a heat map

In order to analyse what consequences different levels of traffic between peers have on anonymity, heat maps are created to show the relative amount of traffic passing through each peer in the network. Such heat maps may be used by an adversary to determine if two peers are communicating with one another. In this analysis, a heat map will essentially be a 3D plot of the network, the first 2 axes are the dimensions of the network 2D network and the third represents the level of traffic received by each peer. We analyse traffic received rather than traffic sent as the shout mechanism used by the network will mean that any traffic observed will be completely void of information on the sender.

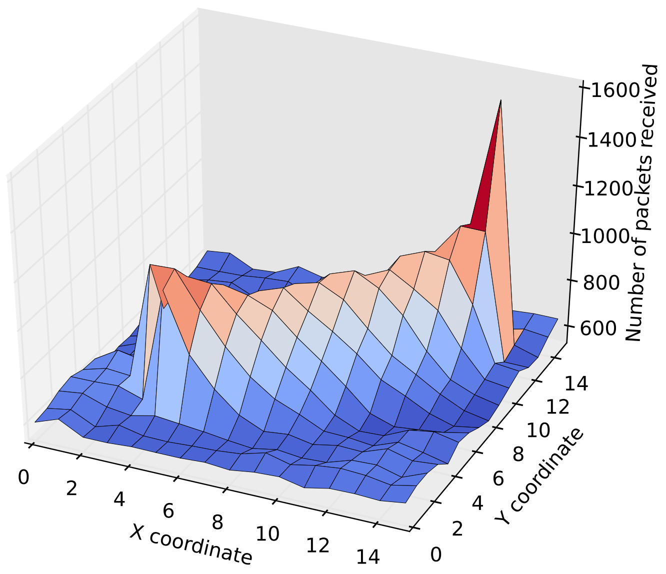

Figure 4.36: The number of messages received by peers when a random routing algorithm is used.

In this first scenario, one peer sends a large amout of traffic to another peer, routed randomly in the space between them. The other peers in the network engage in random communication between one another. The number of packets received by each peer is recorded and these values produce the heatmap in Figure 4.36. The ridge in the heatmap draws a line directly from sender to receiver; this is perhaps the worst possible case, traffic analysis has revealed both sender and receiver with practical certainty.

Recall that generic data packets in Shadow P2P are source routed. An advantage to this is that the sender can choose how the packets are distributed amongst the possible routes to the destination. In the following scenario, a peer sends a large volume of traffic to another peer but it intelligently routes packets such that peers at the same distance from the sender process the same amount of traffic going between the two peers. The result of using such an algorithm is shown in Figure 4.37. This is clearly a much more sensible approach for ensuring anonymity, whilst the size of the traffic still shows up clearly against the background traffic, there is no longer any ridge connecting sender and receiver.

The final part of this analysis entails determining what proportion of the total traffic a peer would need to send in a high volume transfer in order for a pattern to appear in the heat map that connects the sender and receiver with a high level of confidence. In order to do this, a binomial distribution is used to model the number of packets that are received by any given peer. From this, p-values are calculated that represent how unlikely it is to see a peer receiving that number of packets. A p-value is calculated for each point. Then, if a path can be constructed from sender to receiver that only goes through peers with a p-value lower than a given level of significance, we consider the anonymity of the transmission to be broken at that significance level. This test is performed on both methods of packet routing.

For the randomised routing method it is found that to break the $p = 0.05$ significance level, a transfer has to consume roughly 2% of the total network traffic. For the $p = 0.01$ significance level, it takes 3%. When the intelligent method of routing is used, a transmission needs to use 5% and 9% of total traffic for an adversary to be able to detect a significant path at the $p = 0.05$ and $p = 0.01$ significance levels respectively. From this we can conclude that peers can use source routing to their advantage in maintaining anonymity in their communications.

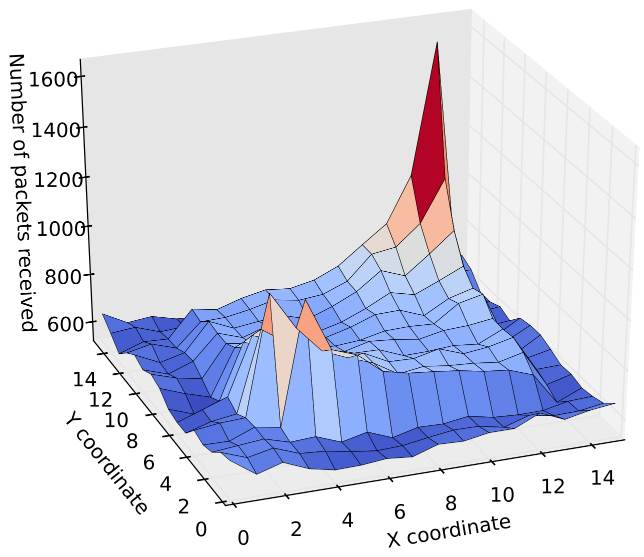

Figure 4.37: The number of messages received by peers when an intelligent algorithm attempts to disguise the traffic pattern.

Chapter 5

Conclusion

In this project I have created a design for a network that aims to provide more in the way of anonymity than any other network trying to achieve the same goal. In doing so I invented a method for communicating between peers where not even the peers themselves know the true identity of their neighbours; the 'shout'. This method is very inefficient in the transfer of data and is reliant on the state of the Internet where the peer is setup, however, as stated these are not primary concerns of the project.

It was seen that the shout method suffers a major drawback in that an adversary could search the shout list for the peer's true identity. I conceived the 'shout group' for the sole purpose of preventing such a search. These groups are effective in deterring any adversary from performing a search as an adversary would have to spend a very long time collecting a large amount of data from the interactions with the shout group in order to deduce any shout group member's identity. The time this takes is further lengthened by the use of a proof-of-work scheme that vastly increases the time between an adversary's attacks, further securing a peer's anonymity.

The network was constructed with the shout and shout group mechanisms at its core. The peers were then arranged into a uni-directional toroidal structure in order to eliminate direct bi-directional communication to increase the difficulty of traffic analysis. I created of a method for expanding and contracting this regular network structure to maintain its regularity and allow new nodes to join the network. A tree of cryptographically signed certificates ensures that peers can be sure of the location of peers within the network and so can resist mis-information from an adversary.

For the transmission of data, packets had to be changed so that they were completely unrecognisable from their previous form in order to prevent the tracing of packets. Public key hiding was invented to allow packets to contain all the information they needed to be routed and re-encrypted and so that this information could be scrambled itself without compromising its usefulness. Breaking this hiding method was proved to be at least as hard as a known computationally hard problem so the anonymity of the packet's receiver is assured. Furthermore, the use of intelligent source routing was shown to be effective in disguising a large data transfer amongst the normal level of network traffic, preventing traffic analysis from determining which two peers are taking part in a given transfer.

As it stands, the project provides the theoretical grounding for an implementation. Shouts, shout groups, public key hiding and the network structure have all been proven to work. In the case of the shout group, there are unsolved secure multiparty computing problems for which clear definitions of the problems have been provided. If these problems are solved in the future, it will allow shout groups to be created from untrusted peers instead of just from known trusted parties. The network simulator goes some of the way in showing that the network design is feasible however no complete implementation of the network has been created.

With respect to the initial aim of creating a network that is more anonymous than any other, I claim that I have although it is hard to show this. The difficulty in doing so is because the methods I have devised are not easily compared to existing methods. However, many of the techniques involve removing information from tasks that are performed in other anonymous networks. For instance, the shout removes information about the peers when commumication between peers occurs and the network structure attempts to remove location based information from the network of peers. So the features of Shadow P2P can be seen as doing necessary peer-to-peer tasks with added anonymity.

Any future development should start with implementing the network in full and then testing it either on a LAN or with a network of virtual machines. This will allow the techniques used to be refined and will allow any issues arising to be investigated and solved. Theoretically, the shout group technique is in need of more research. It is by far the hardest problem I have encountered in this project and still doesn't meet up to my expectations. Research in the area of SMC may solve the issue with trusted peers but the cut-off point selection still lacks in its ability to hinder search. To the best of my knowledge, this project is the first in using IP spoofing as a main communication component of the network; perhaps a completely different method that achieves the same effect could be achieved and thus the need for the shout group would be removed in doing so.

As a final point, whilst it approaches the limits on anonymity, Shadow P2P is far too impractical and inefficient to gain standing as a popular anonymity tool on the Internet. The existing networks are sufficient in providing anonymity where it is needed. Hopefully, the techniques developed here may go on to improving the anonymity provided by the current plethora of solutions as the challenges facing them change.

Bibliography

[1] Martin Abadi, Mike Burrows, Mark Manasse, and Ted Wobber. Moderately hard, memory-bound functions. *ACM Transactions on Internet Technology (TOIT)*, 5(2):299–327, 2005.

[2] The aol anonyised data incident. `http://www.nytimes.com/2006/08/09/technology/09aol.html?pagewanted=all&_r=0`.

[3] Avalon asic bitcoin mining rig. `http://launch.avalon-asics.com/`.

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

[5] Robert Beverly and Steven Bauer. The spoofer project: Inferring the extent of source address filtering on the internet. In *Proceedings of USENIX Steps to Reducing Unwanted Traffic on the Internet (SRUTI) Workshop*, pages 53–59, July 2005.

[6] Robert Beverly, Arthur Berger, Young Hyun, et al. Understanding the efficacy of deployed internet source address validation filtering. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 356–369. ACM, 2009.

[7] Bitcoin, the p2p virtual crypto-currency. `http://www.bitcoin.org/`.

[8] Magnus Bråding. Generic, decentralized, unstoppable anonymity: The phantom protocol.

[9] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology*, pages 10–18. Springer, 1985.

[10] Freenet. `https://freenetproject.org/`.

[11] Gnunet. `https://gnunet.org/`.

[12] Gnutella is a network created from many different client applications. `http://www.gnutellaforums.com/`.

[13] Oded Goldreich. Secure multi-party computation. *Manuscript. Preliminary version*, 1998.

[14] I2p. `http://www.i2p2.de/`.

[15] I. Jerebic and R. Trobec. Optimal broadcasting in toroidal networks. In *CompEuro '92 . 'Computer Systems and Software Engineering',Proceedings.*, pages 671–676, 1992.

[16] Dogan Kesdogan, Jan Egner, and Roland Büschkes. Stop-and-go-mixes providing probabilistic anonymity in an open system. In *Information Hiding*, pages 83–98. Springer, 1998.

[17] Lansim. `http://lansim.sourceforge.net/`.

[18] Ns3. `http://www.nsnam.org/`.

[19] Andreas Pfitzmann and Marit Köhntopp. Anonymity, unobservability, and pseudonymity—a proposal for terminology. In *Designing privacy enhancing technologies*, pages 1–9. Springer, 2001.

[20] Certicom Research. Standards for efficient cryptography. sec 1: Elliptic curve cryptography. pages 50–54, May 2009.

[21] Andrei Serjantov and George Danezis. Towards an information theoretic metric for anonymity. In *Privacy Enhancing Technologies*, pages 41–53. Springer, 2003.

[22] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[23] Matthew Tanase. Ip spoofing: an introduction. *Security Focus*, 11, 2003.

[24] The tor project. `http://www.torproject.org/`.

[25] Leslie G. Valiant. A scheme for fast parallel communication. *SIAM journal on computing*, 11(2):350–361, 1982.