AN ANONYMOUS AND SCALEABLE DISTRIBUTED PEER-TO-PEER SYSTEM

by

Sean Blanchfield

A thesis submitted in partial fulfillment of
the requirements for the degree of

B.A., B.A.I. (computer engineering)

The University of Dublin, Trinity College

The University of Dublin, Trinity
College

Abstract

AN ANONYMOUS AND SCALEABLE DISTRIBUTED PEER-TO-PEER SYSTEM

by Sean Blanchfield

Supervisor:	Mr Mads Haahr
	Department of Computer Science
Second Reader:	Mr Stephen Barrett
	Department of Computer Science

“Peer-to-Peer” networks have recently received much attention, particularly with the rise of systems such as Napster [Oram 2001a], Gnutella [Kan 2001], SETI@home [Anderson 2001] and many more. These systems represent an opportunity to take advantage of the massive untapped resources of the many machines at the edges of the Internet [Shirky 2001]. The popular goal of these systems is to provide anonymous and secure access to resources in a scalable fashion, however this has posed many obstacles. The aim of this document is to describe a system that overcomes these obstacles and achieves an anonymous, secure and scalable peer-to-peer network, utilised in this case for the purposes of a distributed file sharing application.

DECLARATION

I certify that this thesis does not incorporate without acknowledgement any material previously submitted for a degree in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person where due reference is not made in the text.

ACKNOWLEDGEMENTS

Several individuals deserve acknowledgement for their help during this project. Mads Haahr constructively lent his expertise throughout the design, and his enthusiasm was very helpful. Brian McDonnell was an intelligent sounding board in the initial stages, and quickly spotted areas in the design that deserved more thought. Owen Delaney and Tony O'Donnell also deserve thanks for the encouragement they provided throughout.

LIST OF FIGURES

Figure ***Page***

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

1.1 Background

1.1.1 Peer-to-Peer

The field of peer-to-peer is a new area of computing, to the extent that the first and only text on the area [Oram 2001a] was published as this project was nearing completion. Consequently the term peer-to-peer lacks a formal definition and there exists much disagreement regarding its precise meaning.

The success of Napster [Oram 2001a] heralded the widespread use of the term, however as Napster depends on the centralisation of various metadata there is some disagreement as to whether it is truly a peer-to-peer technology. Many believe that peer-to-peer should only describe those highly distributed *ad hoc* networks such as Gnutella [Kan 2001] and Freenet [Langley 2001a] that contain no centralised components whatsoever.

Unfortunately, this definition fails to capture the primary reason for the success and popularity of the various systems in question. [Shirky 2001] points out that the single essential feature peer-to-peer systems have in common is that they tap into the "dark matter of the Internet" – home PC's. When participating in a peer-to-peer network, the user's personal computer can participate in an active role (possibly via a dial-up connection). This may mean that the user donates disk space to the network as seen in the various file sharing networks, or alternatively

compute cycles as seen in SETI@home [Anderson 2001]. In either case, the user ultimately benefits by being given access to the combined resources of the network [Bricklin 2001]. [Shirky 2001] conservatively estimates that personal computers connected to the Internet represent a potential resource of 10^{10} Megabytes disk storage and 10^{10} Megahertz processing capability.

[Shirky 2001] also defines a "litmus test" for peer-to-peer as follows:

1. Does it allow for variable connectivity and temporary network addresses?
2. Does it give nodes at the edges of the network significant autonomy?

This definition for peer-to-peer is considered to be the most useful and is adopted in this document.

The question now arises as to the distinguishing characteristics of the different systems encompassed by this broad definition. [Hong 2001] classifies peer-to-peer systems into three categories:

- Centrally Coordinated. Member peers share resources, but must be facilitated by a central server. Napster is an example of such a system.

- Hierarchical. Peers are organised into hierarchically tree of groups. Communications are mediated by local coordinators, and are passed upwards to more senior coordinators if required. DNS represents a good example of such a system.

- Decentralised. No global coordination exists in the system at all, but instead peers operate a protocol that implicitly coordinates the entire network as a result of its execution. Peers coordinate communications on behalf of other peers local to them. Gnutella and Freenet are examples of decentralised peer-to-peer systems.

These categories allow us to usefully distinguish between the different varieties of peer-to-peer systems.

1.1.2 Anonymity

Decentralised peer-to-peer systems lend themselves naturally to anonymous communications, as there is no central point at which to collate information about users. However, careful design choices must be made to ensure this anonymity. As will be shown in Chapter 2, Gnutella fails to achieve anonymity as peers connect directly to exchange data, thus revealing their addresses to each other. In fact, this flaw has been exploited to entrap users of the system [Kan 2001]. Freenet on the other hand achieves anonymity by exchanging data via a chain of peers, where no peer but the recipient is aware where the chain terminates.

It is clear that there are several aspects to anonymity that must be considered in an anonymous system. [Dingledine 2000] defines the following types:

- Author-anonymity. In such a system, an author cannot be linked to a document that he or she has created.

- Publisher-anonymity. In such a system, a publisher cannot be linked to a document that he or she has made available to the network.

- Reader-anonymity. In such a system, a user cannot be linked to a request that he or she has made for a document.

- Server-anonymity. In such a system, a server cannot be linked to documents that it is storing.

- Document-anonymity. In such a system, a server cannot determine which documents it is currently storing. This property also implies that there is

no information to be gained about stored documents by an attacker that manages to compromise a server. A negative effect of this type of anonymity is that servers are unable to perform advanced matching of stored documents against queries. For example, a free-text search on document titles is impossible as the titles of stored documents are unknown.

- Query-anonymity. In such a system, a server cannot determine which document it is serving when satisfying a request.

The above forms of anonymity can be combined in several ways to produce notable properties:

- Document-anonymity and query-anonymity can be combined to produce the property of "deniability" on behalf of a server [Dingledine 01]. In this case, a server is both unaware of what documents it is storing locally and what documents it is serving to the rest of the network. It is thus unreasonable to hold a server responsible for serving any particular document.

- Reader-anonymity and server-anonymity together provide reasonable anonymity. Although queries may not be anonymous, they cannot be attached to any node in the system. In this sense, nodes generating or responding to a query "blend into a crowd," as outlined extensively in [Reiter 1997].

Author-anonymity is generally not an issue, as identification of the author of a document (or lack thereof) is implicit in the document itself. For example, the document may be a signed text, or a digital music file of a song by a well-known

artist. Such implicit identification of the authors of documents is outside the scope of most peer-to-peer systems.

It should also be noted that even in the best-case scenario the above anonymity properties provide no protection against more subtle attacks. For example, style analysis may be performed on a text in an attempt to identify its author, or perhaps the content of a particular document that was inserted into the network as a private communication may identify its intended recipient.

1.1.3 Scalability

Scalability has proven to be an important and difficult issue in decentralised peer-to-peer systems. This arises from two issues:

- The high popularity of such systems, as demonstrated by the Napster phenomenon [HREF 1] may result in rapid growth of network. For example, Napster proved to be popular enough to attract more than sixty million users within a twelve-month period.

- The decentralised nature of the system means that any peer in the system is potentially a bottleneck. Ideally anyone should be entitled to become a peer on the network, even those with low-bandwidth connections. Unless special precautions are taken in organising the infrastructure of the network, such peers may not be able to participate effectively, and as a consequence be unable to route network traffic on behalf of other peers. The Gnutella network is effectively crippled by this effect, an analysis of which has been carried out by Clip2 [DSS 2000b].

Clearly, it is desirable that a peer-to-peer network should be able to scale well. It is therefore of importance to design the network so that it can grow gracefully to any size.

The relation between network size and network traffic also deserves careful consideration. Each peer must dedicate some of its bandwidth resources to the routing of communications on behalf of the other peers in the network. It is possible that the bandwidth required to do so may grow to exceed the total bandwidth capabilities of some low speed users. The Clip2 analysis [DSS 2000b] demonstrates that the bandwidth required to route ten Gnutella queries per second is approximately 67,200 bps. They therefore deduce that as the Gnutella network grew in size and general network traffic grew correspondingly, low-speed users operating at the "56Kbps barrier" began to fail in routing the required level of traffic.

[Hong 2001] analyses how network traffic scales on both Gnutella and Freenet. His findings indicate that (assuming all traffic has an infinite time-to-live) Gnutella traffic scales linearly in relation to network size, and Freenet traffic scales logarithmically. Logarithmic scaling is assumed to be the best one could hope for in such a system, therefore Freenet obviously performs very well in this respect. However, Gnutella deals with the linear growth of its network traffic by introducing a time-to-live value for each packet that is sent [Kan 2001]. This value is decremented with each hop that the packet makes, and the packet ceases to be sent when it reaches zero. This results in a network "horizon" from the perspective of each peer on the network – i.e. each peer can only communicate with other peers within a certain number of hops from its location. This is a trade-off between the connectivity of each peer and the level of network traffic each peer must handle. This is reasonable so long as information is sufficiently replicated so that each peer can reach it within its own horizon. Whether a Freenet or Gnutella style approach is taken to dealing with this issue, it is clearly something that deserves careful consideration at the design stage.

The final point to consider in regard to scaling is that of minimising the bandwidth required by network traffic. This means that in order to cater for the large number of low-speed modem users that may potentially use the system, it is necessary to minimise the actual size of the packets. As mentioned above, Gnutella modem users are unable to cope with network traffic by quite a small margin. However, Gnutella operates over HTTP and therefore much of the data contained in the packets is redundant to the particular function of the packets. It is possible that the scalability problems that Gnutella has encountered may have been avoided through the use of small binary-encoded packets instead. Although XML formatted packets sent via HTTP, as described in [Wiley 2001] may be more elegant and allow the use of tried and tested code, this author believes that it is currently undesirable as most users are currently connected at bandwidths lower than 56Kbps.

1.2 Motivation

1.2.1 Benefits of Peer-to-Peer

The field of peer-to-peer has suddenly become a hotbed of activity and has sparked an unheard of level of interest from users and developers alike. The key reason for this is that by its very nature it requires users to be active participants, as opposed to passive consumers of information. The World Wide Web is more akin to television than its inventor originally intended, and it is very possible that peer-to-peer technology will eventually re-address this deficiency [O'Reilly 2001]. [Truelove 2001] describes the idea of the "transient web" – that is, the silent majority of computers connected to the Internet that are excluded by the DNS on the basis of their dynamically assigned IP addresses. [Truelove 2001] also argues that Gnutella may be the start of a movement to incorporate these transiently connected computers back into the Web in an active role.

Another important benefit of peer-to-peer technology is that it provides a very efficient distribution medium. For example, although movie studios have expressed concerns over copyright implications of such systems, they have also expressed interest in using them as a distribution channel of movie trailers [HREF 2]. The distribution of such trailers, which are often very large in size, from company-owned servers is very expensive. However file-sharing networks such as Gnutella have proven that many users are willing to host such trailers themselves (with or without the studio's permission).

It follows from this that the so-called "slashdot effect" does not generally apply to peer-to-peer networks. This term refers to the situation in a client-server system when popular information becomes unavailable due to server overload [Adler 2000]. In contrast, peer-to-peer networks lend themselves to natural replication of popular data. For example, Freenet explicitly replicates data in response to its popularity. On the other hand Gnutella achieves the same end, as most users should make data they download available for upload from their own locations.

For these reasons, a project in peer-to-peer technology provides an interesting challenge with clear applications, in an area in need of innovation.

1.2.2 Privacy and Anonymity

> "*Every man should know that his conversations, his correspondence, and his personal life are private.*" – Lyndon B. Johnson, President of the United States, 1963 – 1969.

Privacy is a right that all people should be entitled to, and anonymity is one means of achieving that right. In this context, anonymity is used in daily life and has been used throughout history to protect those who risk persecution of some kind by speaking out. This may be a sufferer of a narcotic dependency who

anonymously calls a radio show, a political dissident who anonymously publishes a pamphlet, or an employee who anonymously writes to a newspaper about his or her company's environmental pollution. In a more historical context, James Madison, Alexander Hamilton and John Jay anonymously published the Federalist Papers in the New York state newspapers in 1787 and 1789, helping to convince the state to ratify the proposed US constitution [Waldman 2001].

The right to remain anonymous was upheld in the 1996 lawsuit of the American Civil Liberties Union versus the State of Georgia. A new state law that was designed to prevent online fraud made it illegal to falsely identify oneself on a computer network [Dingledine 2000]. As a consequence of this it also became illegal to assume a false name on a computer network in order to achieve anonymity, and the Supreme Court declared the law unconstitutional in 1997.

Unfortunately our personal privacy is threatened in the digital age. As [Clarke 1999] points, out the perceived anonymity of the Internet is caused by a lack of education about the technology, rather than being a built-in feature. It is in fact possible to automate the monitoring of Internet (and digital communications in general) intelligently, and on a grand scale.

Although it is possible to encrypt Internet communications using technologies such as SSL or PGP, sufficient security is still not guaranteed. The simple fact that the communication took place, and the identity of the two parties involved, cannot be concealed. Furthermore, in some countries, the simple fact that an encrypted communication took place may be enough to incriminate both parties [Oram 2001b].

Another worrying consequence of the ability to extensively monitor communications is the ability to censor them. The "Great Wall of China" – a

nation-wide firewall – provides such an example [Yurcik 1996]. This firewall prevents Chinese citizens from accessing "subversive" sites on the Internet.

Although information policies in the East do much to motivate an argument for an anonymous communication channel, there are also extensive electronic surveillance systems in operation in the western world that support the argument just as persuasively.

In the United States, the FBI is attempting to introduce a system called "Carnivore," which will enable Internet wire-tapping to be performed more easily. This is essentially a secure "black-box" machine, which all Internet service providers will be obliged to install at each of their centres. The ISPs will also be obliged to give the Carnivore machine full access to monitor all traffic through the centre. The FBI has given assurances that no traffic will be logged unless they have received permission from a court beforehand, however this has been met with much scepticism [HREF 3]. The American Civil Liberties Union (ACLU) is spearheading the campaign against the system in conjunction with ISPs such as Earthlink. In July 2000 the ACLU filed a freedom of information act request to obtain the source code for the system, however this request was not met. The American Department of Justice subsequently performed an independent technical review of the system, which among other conclusions, found:

> "While operational procedures or practices appear sound, Carnivore does not provide protection, especially audit functions, commensurate with the level of the risk." [Smith 2000].

The issue was further clouded when details were discovered which brought the independence of the reviewers into question [McCullagh 2000].

Systems such as Carnivore are not isolated to the United States. [HREF 4] asserts that the Regulation of Investigatory Powers Act (UK) 2000 is designed to facilitate a analogous interception system, and that the Council of Europe is deliberating over similar bills which would give agencies Internet interception capabilities in over forty countries.

It is clear that the Carnivore and related systems are a threat to the privacy of many people, and many organisations and individuals are taking it very seriously.

Although the Carnivore system is worrying, the Echelon program is far more so. This prompted the European Parliament Scientific and Technological Options Assessment (STOA) to produce an instructive working document on the subject of Echelon and communications intelligence [STOA 1999]. The Echelon program is an automated signals intelligence (sigint) system, run by the UKUSA alliance. The UKUSA alliance is a collaborative global communications intelligence organisation that was founded by the UK and the USA following the Second World War. It was confirmed that Canada, Australia and New Zealand are also members when the organisation was first publicly acknowledged in 1999. It is also rumoured that the Republic of Ireland may also be a member as of June 2000 [Goodwins 2000].

The capabilities of Echelon include automated monitoring of telephone, fax and email communications using a keyword search algorithm. Through an extensive network of land and satellite based interception centres, and links with organisations such as the maritime satellite system Inmarsat [Goodwins 2000], it is believed that Echelon coverage is now sufficient to capture the majority of digital communications worldwide. In a report released by the European Parliament in January 1998 it was stated that:

> "within Europe, all e-mail, telephone and fax communications are routinely intercepted by the United States National Security Agency." [Loeb 1998]

The European Parliament voted to assemble a panel of enquiry to investigate Echelon and the United Kingdom's involvement in March 2000 [Rothenberg 2000]. This was named the "Temporary Committee on Echelon" and began work on 11 September 2000 [HREF 4]. It is still in deliberation.

Apart from the infringement on personal privacy that results from the deployment of such large "comint" systems, there is also a clear danger of economic espionage. [STOA 1999] outlines several alleged cases where the US government has been accused of using the Echelon system to supply American companies with information covertly gained from foreign companies or organisations, in order to provide them with a competitive advantage.

It is desirable that a system should exist that removes the possibility of such extensive surveillance of individuals and companies around the world. A peer-to-peer project with goals of anonymity and security may move us a step closer to the ideal of a network infrastructure that can support varied applications while maintaining both the freedom and privacy of its users.

Chapter 2

RELATED WORK

This chapter surveys some of the other peer-to-peer systems that bear most relevance to this project, and explores some of their key features and deficiencies.

2.1 Gnutella

Gnutella [Kan 2001] is considered to be the first completely decentralised peer-to-peer protocol to be created. The first client was written largely as an experiment by developers at Nullsoft, a subsidiary of AOL. Upon launch, Gnutella was swiftly labelled an "unauthorised freelance project" by AOL and removed from the Nullsoft website. The Open Source community soon continued its development and there now exist numerous clients operating the protocol. As Gnutella is a file sharing protocol it quickly attracted much media attention and became very popular. It is without doubt the most studied of all decentralised peer-to-peer systems, and provides a very illustrative example of the underlying principles and challenges associated with this kind of network.

The Gnutella protocol is essentially quite simple [DSS 2000a]. Each node on the network maintains a number of connections to randomly selected neighbours, normally ranging from three to six. When a node wishes to search for a particular file, it sends a query to each of its neighbours. These neighbours in turn broadcast to each of their own neighbours, and so on. As [Kan 2001] describes, broadcasting is effectively achieved via TCP unicast. This broadcasting mechanism is subject to some caveats, specifically:

- The query ceases to be broadcast after it has travelled a specified number of hops from its origin. To facilitate this a time-to-live field is specified in the packet.

- A node will not continue to broadcast a packet that it has previously broadcast. This precaution is necessary as a query packet may reach a node via many routes, and it would be redundant for a node to send a packet to each of its neighbours when it is implicit that it has already done so. A node can recognise a packet by its globally unique identifier (GUID).

If a node possesses data matching that requested by a query packet, it can respond via a special unicast mechanism. This mechanism relies on each node in the network keeping a list of recently seen packets, including the neighbour each respective packet was received from. A response is made by sending a packet with a GUID similar to the original query to the neighbour that the original packet was received from. In turn that neighbour node will send it back in the direction that the original packet was received. In this fashion the response packet hops from node to node back along the path of the original packet until it reaches the origin of the query.

The above mechanism allows downloads to be negotiated with quite good reader-anonymity and server-anonymity, as it would be a highly complicated task for a third party to trace the route of each packet in the network. Unfortunately this anonymity is negated by the fact that actual downloads are performed via direct HTTP connections between the two peers, thus revealing their identities to each other. This fact has been utilised to entrap users of Gnutella in the past [Kan 2001].

[Hong 2001] explains the topology of Gnutella in terms of *Graph Theory*. Graph theory is a formalised system that can be used to model interconnected networks. It has typically been used to explain the characteristics of social networks between

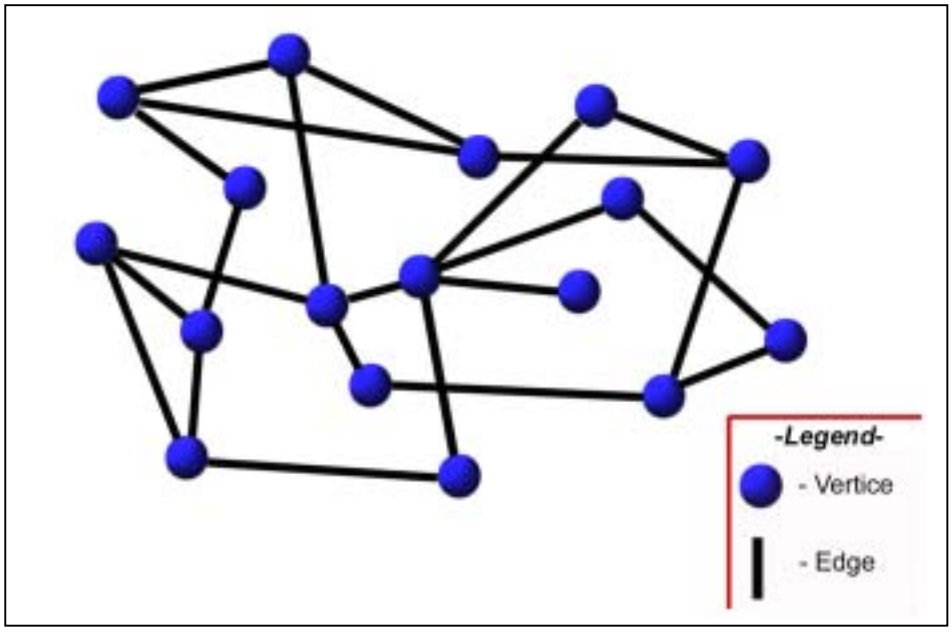

Figure 2.1 A graph theory illustration of a Gnutella network

people, but can similarly be used to explain the characteristics of a randomly connected system such as Gnutella. In terms of graph theory, each node is considered a *vertex*, and each connection is considered an *edge* (Figure 2.1). Gnutella relies on maintaining a small characteristic path-length in order to be effective. The characteristic path length is an important property of such a network and is defined as the average number of intermediary vertices between any two vertices. [Watts 1998] describes the characteristics of "small-world" network dynamics. As Gnutella is in effect a large and random small-world network, it can be calculated that the characteristic path-length is approximately equal to log n/log k, where n represents the total number of vertices in the system and k represents the average number of edges at each vertex.

As a typical example, we can estimate that in a Gnutella network there are 10000 nodes and that on average each node has approximately 4 neighbours. We can then calculate the characteristic path-length to be approximately 6.4 hops, which is small enough for us to deduce that all nodes are quite efficiently contactable. Even if we increase the number of nodes by a factor of 10, the characteristic path-length only increases to approximately 8.3.

With this in mind, the randomly connected nature of the Gnutella network is a great advantage. However the randomised nature of the network also means that no discrimination is made between high-bandwidth and low-bandwidth nodes. The lack of any kind of organising principle is one of the main reasons for the near-collapse of the network in late July and early August of 2000 [DSS 2000b]. This period correlates to the so called "Napster flood," the result of large numbers of low-bandwidth Napster users moving to Gnutella upon news of legal action taken against Napster by the Recording Industry Artists' Association.

These new users were incapable of proper participation due to their bandwidth, and furthermore, they were incapable of handling the network traffic demands placed on them by their peers. As a result these bottleneck nodes prevented much communication from taking place, and the network became severely fragmented. The network currently exists in an intermediate state between scaling and collapsing [DSS 2000b].

Another notable challenge that was faced by Gnutella is that of "free-riding". A study carried out by Xerox PARC [Adar 2000] indicates that more than 50% of files available on Gnutella are being provided by fewer than 1% of the peers. It also indicates that over 70% of users of the system provide no files whatsoever. This is commonly quoted in peer-to-peer systems as a classic case of the "Tragedy of the Commons" [Hardin 1968]. This refers to the general situation where an unchecked communal resource becomes depleted as its users pursue

self-interest above the interests of the community. Peer-to-peer systems are especially susceptible to this syndrome as they lack a channel over which to exert a sense of social ethics upon users. It is therefore appropriate to include a method of either rewarding good practices or punishing bad practices on the part of the users. This can be built in at either the protocol level or the client level. Unfortunately Gnutella takes neither approach and as a result there remains no motivation for users to behave as good "citizens" of the network. To compound matters, the original and most popular version of the Gnutella client, version 0.56, does not take the elementary step of making the user's download directory available to other peers by default.

2.2 Crowds

Crowds is a system developed at AT&T Labs to provide anonymity for web transactions [Reiter 1997]. It operates as a HTTP proxy on each peer machine. Whenever a URL is requested from the user's Web browser, the Crowds HTTP proxy will most probably forward the request to another known peer in the system or otherwise send the HTTP request to the destination web server. If it forwards the request to another peer, then that peer will perform a similar action – i.e. with high probability forward it to another peer, or otherwise send the request to the destination web server. This system results in any HTTP request actually being sent from an arbitrary peer, possibly the peer that actually generated it. Thus all peers "blend into the crowd" when sending requests. Not even other members of the crowd are aware which peer was responsible for any particular request. Figure 2.2 illustrates the algorithm.

Information returned by the web server is passed back along the intermediary peers in the crowd until it reaches the peer that made the original request.

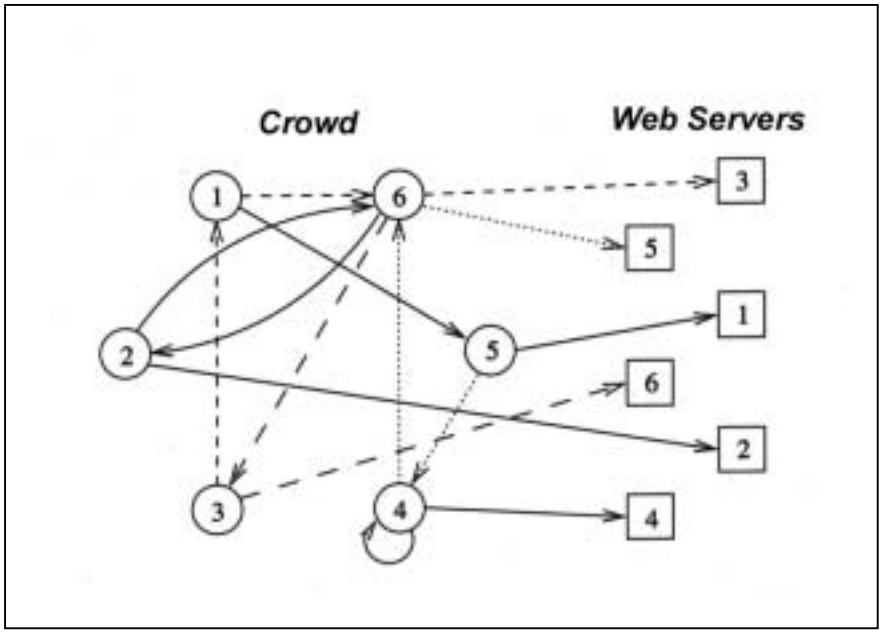

Figure 2.2 Operation of the Crowds algorithm

Crowds provides reader-anonymity only, but does so in a fashion that is analogous to some other peer-to-peer systems. The basic objective is to make it impossible to determine which participant generated a particular request. We see a similar effect in the Gnutella searching algorithm. In Gnutella a query is passed from node to node to such an extent that it is difficult for any peer or external third party to determine which node actually originated it.

The scalability of Crowds has never been put to the test. The system has not attracted a large number of participants in comparison to some other peer-to-peer projects, as in order to join a Crowd one must make an application and fulfil several requirements.

Crowds deals with low bandwidth users by forbidding them membership of the Crowd. A low-bandwidth peer participating in a generally high-bandwidth Crowd would significantly bottleneck the overall performance. It is possible however that a group of modem users could form together to create their own low-

bandwidth Crowd. Unfortunately, the members of this Crowd would undoubtedly suffer a significant performance drop, as all their Internet traffic would be routed via a number of other modem users before being sent or received.

Crowds successfully proves the viability of an anonymising algorithm. It is unfortunate however that it does not attempt to provide greater anonymity to enable the publication of information and that it does not attempt to provide for low-bandwidth users, as these are two of the most central ideas behind peer-to-peer technology.

2.3 Freenet

Freenet was created as a final year project at the University of Edinburgh by Ian Clarke in 1999 [Clarke 1999]. As is the case with the project described by this report, Freenet is motivated by the belief that all individuals are entitled to freedom of information and personal privacy.

Freenet is a decentralised peer-to-peer system, however it operates according to a significantly different model than Gnutella. It implements reader-anonymity, server-anonymity, query-anonymity and document-anonymity. This means that it is not only impossible to determine the identities of a user requesting data and the servers satisfying the request, but it is also impossible for a server to determine what documents it is storing and what requests it is satisfying. These design requirements have resulted in quite a complicated system, which shall only be described in brief here.

Freenet also implements a caching algorithm, which replicates data as it is requested throughout the network. This algorithm has the effect of moving data towards those areas of the network where it is most in demand.

[Langley 2001] provides an excellent description of the operation of Freenet. A brief outline is provided below.

Random searches for documents meeting certain criteria are not possible on Freenet, as the system implements document-anonymity. Instead specific documents may be requested using their respective keys. In contrast to Gnutella, no broadcasting takes place when a user requests a document. Rather, the request is passed node to node in an intelligent manner until the request is satisfied. In a sense this represents a depth-first search as opposed to the breadth-first search performed in Gnutella. If a request reaches a node with matching data – i.e. the key can be matched to an encrypted document in its cache – then that node will send that data back along the chain of nodes that it was contacted by. Nodes in that chain then cache that data.

An important point to note is that nodes learn about other nodes via the receipt of responses to requests. This means that most of the peers that any node is aware of will be those that have satisfied similar requests in the past. The assumption is that nodes that have satisfied previous requests are likely to satisfy requests in the future, and that therefore the network will become connected in a manner that reflects the various areas of interest.

Whether or not a node decides to cache data that it is handling depends on the popularity of the data. A certain amount of disk space is set aside for the cache, and documents displace other documents on the basis of the frequency of explicit requests for them. This of course implies that less popular data (e.g. political "whistle-blowing" documents) may be replaced quickly with more popular data (such as music files).

Documents must be explicitly inserted into the network before becoming available. A data insert message, containing the encrypted data, is forwarded

through a chain of nodes until the message times out. Each node in the chain retains a local copy of the data in their caches, which will remain there unless pushed out by more popular data.

Freenet is an ambitious system with very clearly defined goals in the area of ensuring freedom of speech. It effectively removes the ability of any party to censor a document. However it achieves this largely through the implementation of document-anonymity, which introduces its own problems. The use of document-anonymity means that documents can only be requested by their assigned keys. This introduces a key distribution problem, similar to the classic problem of key distribution in cryptography. In the case of cryptography it is necessary to somehow distribute keys via some secure channel, which itself requires some prior key to have been distributed. In the case of Freenet it is necessary to distribute keys via some anonymous channel. Unfortunately, in order to use Freenet as this anonymous channel, a key must already have been distributed. It would be further inappropriate to distribute a list of document keys through Freenet as it would be impossible to update.

Another disadvantage of Freenet is that it is impossible to introduce dynamic documents into the system. A much-praised possibility of Gnutella is the opportunity for each peer to interpret queries in different ways, and therefore return dynamic content that it is singularly suited to supply. In Freenet a key for a document must be known before that document can be requested, therefore this exciting possibility is excluded.

Document-anonymity also removes the ability of clients to examine documents for potential metadata that could be used in random searches. For example, Napster examines locally held music files for metadata such as bit-rate quality, artist name other track information. It then uses these to help it match its files against queries that it receives. This powerful feature is precluded by the fact that

stored documents cannot be read. Furthermore, Freenet's depth-first request algorithm is based on its key system and could not operate based on unconstrained conditions such as word-matching, metadata matching etc.

Freenet requires each peer to dedicate a large amount of disk space towards its data cache. This space is the cost of participating on the network, and has no other value as its contents will always be unreadable. Unfortunately this may prove to be too high a price for many people. Although the price of disk space will continue to fall, it is likely that people will wish to store ever-larger amounts of data on the network. As a result the amount of cache space required will increase to compensate.

Freenet also requires users to explicitly insert data into the network in order to make it available. The amount of content on Freenet therefore depends on the active enthusiasm of its users to deliberately make documents available. The "Tragedy of the Commons" [Hardin 1968] effect may prove to apply here, as users may be unmotivated to expend their time and bandwidth on uploading files.

It is therefore concluded that although Freenet is an advanced, ambitious and robust system, several of the choices made in its design may limit its popularity and mainstream usefulness.

2.4 Free Haven

Free Haven was designed by Roger Dingledine in partial fulfilment of his masters degree at the Massachusetts Institute of Technology in 2000 [Dingledine 2000]. Free Haven has not yet been deployed, and as a result there is no information available on the real-world success of its design. Nonetheless it illustrates another approach to the goal of an anonymous peer-to-peer publishing system.

A peer sends a message by simultaneously communicating with every other node that it knows of, via the Cypherpunks remailer network [HREF 5]. The Cypherpunks network is an anonymous remailer service that uses that strategy of "onion routing" [Langley 2001a] to anonymise emails. Each peer uses a pseudonym provided by Cypherpunks to identify itself to other peer. Through this combination of pseudonyms and the remailer service, anonymity is achieved.

In order to insert a document into Free Haven, the document is broken into a number of pieces using Rabin's information dispersal algorithm [Rabin 1989]. This algorithm allows a document to be divided into *n* parts, in such a way that the document can be recreated using any *k* of those parts. Splitting the documents in this way provides a measure of robustness in case some of the document parts are unavailable. A key pair is then generated for the document, and each of the *n* parts of the document are signed using the document's private key. Each signed part of the document is then stored, along with the document's public key, on a peer on the network. The signature and the public key can now be used by any peer retrieving the document to ascertain its validity.

The Free Haven system allows an expiration date to be included with any document being inserted into the network. The document will be maintained on the network until this expiration date elapses. This is an effort to avoid the effect that is seen in Freenet where unpopular information is lost.

The issue of finding a peer to initially store the document onto is not addressed by the design. It is assumed that those publishing to the system have access to a peer, or that some peers will voluntarily make themselves available for storage.

In order to retrieve a document a peer sends its public key along with a message digest of the document to every other peer that it is aware of. All peers receiving this message attempt to match the message digest against all the documents parts

that they are storing. All peers that successfully do this encrypt those parts with the public key and send them back to the requester. Once the requesting peer receives *k* parts, it uses the information dispersal algorithm to reassemble them.

There are two other algorithms of note that operate within the Free Haven system: share trading, and the buddy system. The share trading system allows peers to exchange parts of files that they are holding. This allows a peer to trade a document part that it does not wish to hold for ethical or legal reasons, and also allows document parts to become more dispersed within the system. The buddy system is designed to promote accountability. Each peer has a "buddy" that monitors its transactions. If a buddy peer notices that its corresponding peer has become unreliable it can announce this to the entire network. Each peer can therefore keep track of the trustworthiness all the other peers in the system.

As mentioned above, Free Haven has not yet been deployed. There exist several challenges to its design before its deployment can be undertaken.

The most difficult of these is the question of how to inform users of what documents exist on the system. In order to request a document one must know the document's public key. This is another example of a key distribution problem. A directory of documents and their public keys must be made available somehow, and accessing this directory "out-of-band" will defeat the anonymity that the system is attempting to achieve.

As [Dingledine 2001] notes, Free Haven is very inefficient in its broadcast mechanism. All peers in the system are simultaneously contacted every time a document is requested. [Dingledine 2001] also points out that outgoing messages may be queued before being sent, thereby introducing a significant delay for the user making a request.

Free Haven does not address the issue of anonymity itself, but relies on a third-party remailer network instead. This is a weakness in itself, as performance, organisational or legal issues surrounding the remailer network can be neither addressed nor resolved. The remailer network currently in use is designed for the routing of email messages and it is unlikely that it would be able to cope if Free Haven was deployed on a large scale.

As Free Haven is still a very much a work in progress, there are aspects of the system that have not yet been clearly documented (such as the method of introducing new peers to the system). It is impossible to appraise these parts of the system. Nonetheless it is apparent that there are yet scalability issues to be addressed in the initial design, and much work to be done before a useful implementation can be reached.

Chapter 3

DESIGN

3.1 Aim

The aim of this project is to design a protocol that, when executed on an arbitrary number of machines, will form an ad hoc decentralised peer-to-peer network that supports anonymous and secure communication. It will support the following characteristics:

- Scalability. The network will be designed to scale to any number of potential users.
- Decentralisation. All nodes on the network will be equal peers, and none of them shall occupy a special role of any kind. It will be possible to form the network in an ad-hoc manner, without any initial coordination.
- Anonymity. Sufficient anonymity will be implemented to protect any user from association with any communication or piece of data.
- Security. All communication on the network will be encrypted to protect it from eavesdropping by any agent that is not party to the communication.

It is a further goal of this project to extend the protocol to support the application of file sharing. When implemented this should provide a popular application which will enable the underlying concepts of the network design to be tested on a large scale.

3.2 Design Challenges

The survey of related work performed in Chapter 2 revealed the most prominent difficulties that have been encountered by similar peer-to-peer systems. In this section these difficulties are considered and the initial design decisions relating to them are outlined.

3.2.1 Efficiency

The protocol must be designed to achieve the maximum efficiency possible. An efficient protocol will benefit the scalability of the protocol, as it will keep reduce the overall level of traffic on the network.

Efficiency will also result in greater responsiveness in whatever applications are built on top of the network. These applications will in turn become more useful and usable. To illustrate the importance of this we can consider the Free Haven system, where requests may be queued for a period of time before being sent [Dingledine 2001]. This results in significant delays for users of the system.

A particular challenge that must be met is that of designing an efficient but anonymous method of transferring large amounts of data between any two peers. A specific example of this is the mechanism of transferring a file anonymously. Both Freenet and Crowds transfer files in the same manner that they transfer any other communication – i.e. by passing the data through a chain of intermediary peers. This is very inefficient and is undesirable. Gnutella avoids communicating file data in this manner in order to conserve efficiency, and instead opts to transfer the file directly between the two relevant peers. However this removes any anonymity factor that may have otherwise been achieved.

Section 3.11 describes the approach adopted by this system. Using the anonymous addressing scheme described in section 3.4.3, a random proxy is selected from the network to anonymise non-trivial transactions such as file transfers. This proxy forwards data from the source peer to the destination peer, so that they need not directly contact each other (and thus reveal their identities).

3.2.2 Scalability

As the system is decentralised, there is no control over the number of participants. Scalability is therefore a very pertinent issue to the design.

Gnutella is the only comparable system whose scalability has been tested by its popularity. Section 2.1 described the two causes for its failure to scale. The first of these was related to the efficiency of its protocol. [DSS 2000b] calculated that the average level of traffic on the network was greater than the bandwidth of the users that were connected via modems. If the broadcast mechanism used was more efficient or if the average payload of packets was minimised, the collapse of the network could have been avoided. Section 3.4.1 describes the efficient broadcast design used in this project. A deliberate attempt is also made in the design of the protocol definition units (Appendix A) to keep the size of packets to a minimum.

The second factor in the scalability issues faced by Gnutella related directly to the placement of low-bandwidth users in the network. Such users are more likely to drop packets, and deserve special consideration in structuring the topology of the network. Section 3.10 describes how the protocol specified by this project organises the network into high-bandwidth and low-bandwidth zones. The intention is that nodes within a low-bandwidth zone will not prove to be critical to the operation of the rest of the network.

3.2.3 "Tragedy of the Commons" Effect

The "Tragedy of the Commons" effect [Hardin 1968] was discussed in Chapter 2. It is necessary to take measures to encourage users of the system to share their own resources, as well as benefit from the shared resources of others.

Two such measures are taken, both at the level of the client implementing the protocol:

- As the client implementation operates a file-sharing application, the elementary step is taken of making the contents of the default download directory available to other users. The user's download directory is therefore part of the communal resource that is available through the network. In order to defeat this, a user would have to deliberately move files out of the directory after download.

- The hops-to-live value of broadcasts that a node generates is tied proportionally to the number of files shared by that node. This means that a node's broadcasts become more effective as that node shares more files. It is hoped that this will provide an incentive for a node to share as many files as it can.

The measures above are necessarily specific to the application of file-sharing. They represent an effort to increase the communal resource of the system – in this case files. Clients designed to operate different applications should implement similar rules that are appropriate to whatever communal resource is relevant.

3.2.4 Anonymity

The protocol must implement sufficient anonymity so that no peer can be associated with any particular communication or piece of data. In order to achieve this it is only necessary to implement reader-anonymity and server-anonymity.

Section 2.3 described the difficulties that Freenet faces as a result of implementing document-anonymity. In general document-anonymity reduces the flexibility, usefulness, simplicity and power of the system. It is also the case that implementing document-anonymity results in users sacrificing their resources for data that they are unable to read or benefit from. It is possible that many users may be uncomfortable with storing unknown data on their machines. Such deterrents to use of the system must be avoided.

The system described here aims only to implement reader-anonymity and server-anonymity, and does not implement document-anonymity or query-anonymity. Although this means that it will be possible to detect many communications on the network, it will be impossible to associate them with any peer.

3.3 Topology

3.3.1 The Lattice

The network formed by the protocol described by this project termed the *lattice*. This name is derived from its grid-like von-neumann style topology. This contrasts sharply with Gnutella, which exhibits a very amorphous and random topology. The lattice can be visualised as a two dimensional grid, with nodes positioned at each intersection. Each node maintains connections to four other nodes, which are positioned at the intersections in the lattice directly north, south, east and west of it. Section 3.3.2 describes how nodes are incorporated into the lattice in a manner that maintains this consistent topology. This network structure

is based on work by Ben Houston [Houston 2000], proposing a mechanism for efficient broadcasting in a Gnutella clone. This proposal is extended here to uncover some further useful properties, most importantly the anonymous addressing system described in section 3.4.3.

3.3.2 Connecting Nodes to the Lattice

In order to achieve this topology, the following conditions are imposed when a node is connecting to the lattice:

- Each node has four neighbour sites: North, South, East and West.

- When a node adjacent to a vacancy is informed about a new node petitioning to fill the vacancy, it decides whether the petitioning node is suitable, based on its bandwidth. If it accepts it will notify the petitioning node. It then proceeds to notify the surrounding nodes, those that will become the petitioning node's neighbours, as shown in the Figure 3.1.

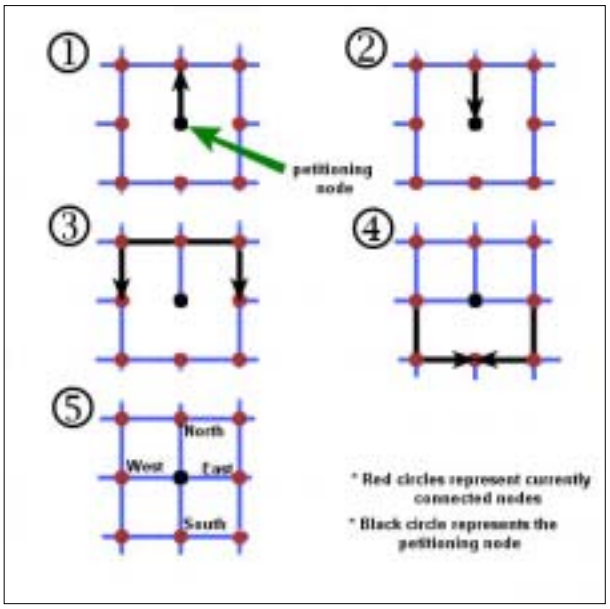

Figure 3.1 A node joining the lattice

When these rules are obeyed a lattice-like network is established.

When a new node is notified that it has been accepted as a neighbour, it sends its public key to its new neighbour. The neighbour then generates a shared key, encrypts it with the public key and sends the encrypted shared key back. In this fashion, each node establishes a shared key for each neighbour to which it is connected. All communications between the neighbouring nodes are encrypted with this key. This makes it difficult for a third party to deduce the origin of any particular packets by analysing the contents of network traffic, simply because one packet is indistinguishable from another in encrypted form.

3.4 Communication Modes

The lattice enables peers to communicate anonymously with each other. The basic communication algorithms available are described here. These provide a basis for any type of communication between peers.

3.4.1 Broadcasting

By following some simple rules, it can be ensured that each node receives each packet once and only once during a broadcast. They are as follows:

- If a packet is received from the east neighbour, pass it on to the west neighbour.
- If a packet is received from the west neighbour, pass it on to the east neighbour.
- If a packet is received from the south neighbour, pass it on to the east, west and north neighbours.

- If a packet is received from the north neighbour, pass it on to the east, west and south neighbours.

Figure 3.2 depicts this algorithm in action.

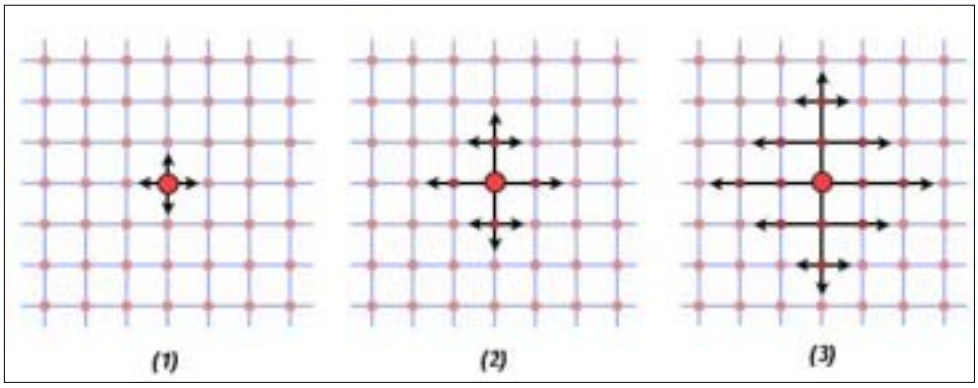

Figure 3.2 The propagation of a broadcast in the lattice

A hops-to-live value (HTL) is included in all broadcast packets to prevent the nodes on the lattice from becoming choked if the lattice were to grow too large. The HTL is decremented by each node that forwards the broadcast, and when it reaches zero the packet ceases to be relayed.

The value for the HTL is calculated to be proportional to two factors:

- An estimate of the bandwidth of an average node. If the average bandwidth of nodes on the lattice grows, the lattice will be able to handle the increase in traffic incurred by a larger HTL. The benefit is that broadcasts will be more effective.

- An estimate of the value of the resource that the node is providing to the network. In the case of a file-sharing application this can be considered to

be equivalent to the number of files that are available from the node. This is an attempt to encourage users to share their resources.

It can be seen from Figure 3.2 that the traffic over the horizontal links in the lattice will be significantly higher than that over the vertical links. One effect of this is that any particular node can tolerate a lower speed connection to its north or south without noticing a drop-off in the responsiveness of the lattice.

This effect of vertical vs. horizontal load is ultimately not of concern however. Each node uses the same network connection for all the north/south/east/west connections. Since the same bandwidth is shared between all the connections, it does not matter that some of them will be significantly more active than others.

3.4.2 Unicasting

For the sake of efficiency it is also possible to unicast packets, however this is only possible when the destination node and the origin have recently been in communication (i.e. one of the nodes has recently sent a communication to the other relating to the unicast to be performed). A unicast works by allowing a packet to hop along the path that was originally taken by the previous communication between the two nodes.

Each node keeps a list of recently received packets that it has relayed to its neighbours, indexed by the Unique Transaction ID (UTID) of each packet. When a node receives a packet, it records the UTID in this list, along with which neighbour it received it from and what kind of packet it was (unicast, broadcast, etc.). This list is also used to store some lattice housekeeping information that is piggybacked on broadcast packets, described in further detail in section 3.6.

If a node receives a unicast packet, it compares the unicast's UTID to UTIDs in its list of recently received packets. By doing so it can find which of its

neighbours it received the original packet from, and can then pass the unicast on to that neighbour. Thus a unicast packet hops from node to node back along the path the original packet travelled, until it arrives at its destination (the originator of the original packet). This node will recognise the packet UTID and realise that it is the destination.

The list of recently received packets needs to be cleaned regularly in order to prevent it from growing to an ungainly size. The main consideration is what age an entry in the table should be before it is old enough to be removed. The majority of unicasts are initiated automatically in response to a broadcast, without any human element. However this cannot always be assumed to be the case. For example, a human user of a node can decide that she wants to download a file after receiving a unicast search hit. The possible human delay is far greater than the possible network delay, and therefore it is the human delay that must determine the age of entries in the table. This maximum age of an entry is set arbitrarily to ten minutes.

3.4.3 Coordinate-Seeking

Coordinate-seeking packets are based on the fact that the lattice has a very regular structure. Because of this, any node can define a Cartesian coordinate system relative to itself, with vertical links in the lattice representing unit steps in the *j*-direction, and horizontal links representing unit steps in the *i*-direction. This allows any node to communicate with another node based on its relative Cartesian position. This constitutes an anonymous addressing system, as no node possesses enough information to correlate the Cartesian position of another node with its actual address (with the exception of the node's four neighbours). The coordinate-seeking communication mode is the mechanism that allows nodes to utilise this anonymous addressing system.

A coordinate-seeking packet contains two fields representing the relative *i* and *j* coordinates of the destination node. A node creating such a packet first sets these values to reflect the position of the destination node. It then sends the packet on to one of its neighbours, in such a way that the packet will be closer to its destination with respect to the coordinate system defined by the lattice. If it cannot do this because it does not have an appropriate neighbour yet (i.e. there is a vacancy in the way), it sends it in a direction orthogonal to the direction it would have sent it without the vacancy.

When a node receives a coordinate-seeking packet, it first corrects the *i* and *j* values to reflect the packet's new position. For example, if it receives it from the west, it increments *i*, if it receives it from the east it decrements *i*. In this way, the values of *i* and *j* always indicate the correct relative coordinate of the destination node. If both the *i* and *j* values now equal zero then the packet has reached its destination and the node will take whatever action is appropriate.

Otherwise, the node then sends the packet on to one of its neighbours (other than the one it received it from), in such a way that either the *i* or *j* value at the packet's next stop will be adjusted in the direction of zero. If it cannot do this because there is vacancy in the way, it sends it in a direction orthogonal to the direction it would have sent it without the vacancy.

This algorithm is robust even if the path between the origin and destination nodes is littered with many vacancies or even contiguous vacancies.

3.4.4 Unidirectional

Unidirectional packets provide a mechanism for nodes on opposite edges of the lattice to communicate with each other. An example of the use of this is described in section 3.8. A unidirectional packet travels in straight lines either horizontally or vertically across the lattice. A node simply sends a unidirectional

packet in the desired direction. A node that receives a unidirectional packet forwards it in the direction opposite to the one it received it from. For example, if a node receives such a packet from the east, it will send it west.

3.4.5 Out-of-Lattice

Out-of-lattice packets refer to packets that are sent directly to another node via a TCP/IP connection. This is obviously a more efficient form of communication than passing the packet through a chain of nodes. However as it is not anonymous it is only used for non-sensitive communication. Typical examples include negotiating new connections between nodes.

3.5 Dealing with Vacancies

It is necessary to find a mechanism to enable packets to find an alternate route around any vacancies they encounter. There are different ways that this can be accomplished.

The most straightforward method would be for nodes surrounding a vacancy to communicate directly across the vacancy. However, this would distort the regular structure of the lattice (which several features of the protocol are dependant on).

The alternative approach is instead adopted of utilising the coordinate-seeking communication algorithm to send the packet to the node on the opposite side of the vacancy. The coordinate-seeking algorithm has the ability to find alternate paths around vacancies and is therefore very suited to this problem.

Figure 3.3 illustrates an example of a packet being routed around a vacancy using the coordinate-seeking algorithm. A packet that is travelling eastwards encounters a vacancy. The node adjacent to the vacancy cannot continue to relay it directly eastwards, so instead uses the coordinate-seeking algorithm to send it to the node with coordinates i=2, j=0, i.e. (2,0). This is the relative coordinate of the node on

the opposite side of the vacancy. The figure demonstrates how each node on the path of the packet continues to send it until the relative coordinates are (0,0). When the coordinates reach this value the packet is sent as normal in its original direction.

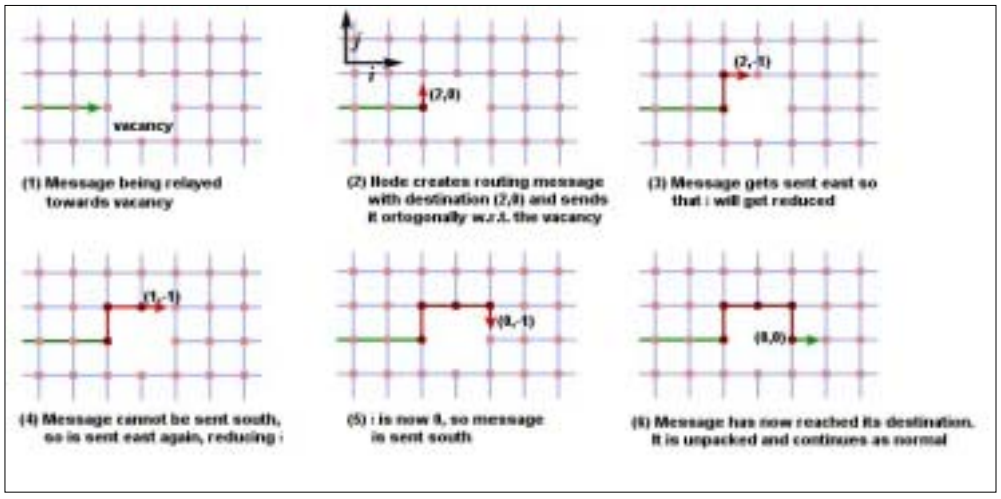

Figure 3.3 A visualisation of packets being routed around vacancies

The coordinate-seeking algorithm used here not only works for routing packets around single isolated vacancies, but for routing around many adjacent vacancies.

It is necessary to introduce a mechanism for nodes that do not have a full compliment of neighbours to detect whether they are adjacent to a vacancy or the actual edge of the lattice. This enables the algorithm described in section 3.8 to operate.

By attaching a hops-to-live value to all packets that are routed around vacancies it is possible for a node to detect when an attempt to route around a vacancy has failed. If this happens, the node can assume that the vacancy is actually the edge of the lattice. The HTL of the packet is decremented with each node that it is routed through. If the HTL reaches zero a packet is unicast back along the path

that it has taken. When this packet is received the node that attempted to route the packet can assume that it is actually at the edge of the lattice.

3.6 Filling Vacancies

As nodes disconnect from the lattice, vacancies appear in the location that they previously occupied. For efficiency it is essential that these be filled.

Along with every broadcast packet passed along the lattice, each node with a neighbouring vacancy indicates the vacancy by modifying the packet appropriately. We define eight distinct categories of connection bandwidth. An 8-bit field is set apart in each broadcast packet – one bit for each bandwidth category. If a node relaying the packet has a neighbouring vacancy then it sets the bit for its own bandwidth category to 1. This indicates to all nodes that subsequently receive the packet that there is a node on the path of the packet in need of a neighbour of that bandwidth category.

Figure 3.4 demonstrates the sequence of events that take place when a node requests to join the lattice. The node wishing to connect first of all sends a join request packet to another node that is already a member of the lattice. This node will respond to it by reviewing recent entries in its list of recently received packets, looking for entries that indicate vacancies in the lattice. Each such entry will also indicate which direction the packet indicating the vacancy was received from (north, south, east or west). It will base its selection on (a) finding the most recent entry and (b) finding one that indicates a vacancy in the same bandwidth category as that of the node that is requesting to connect to the lattice.

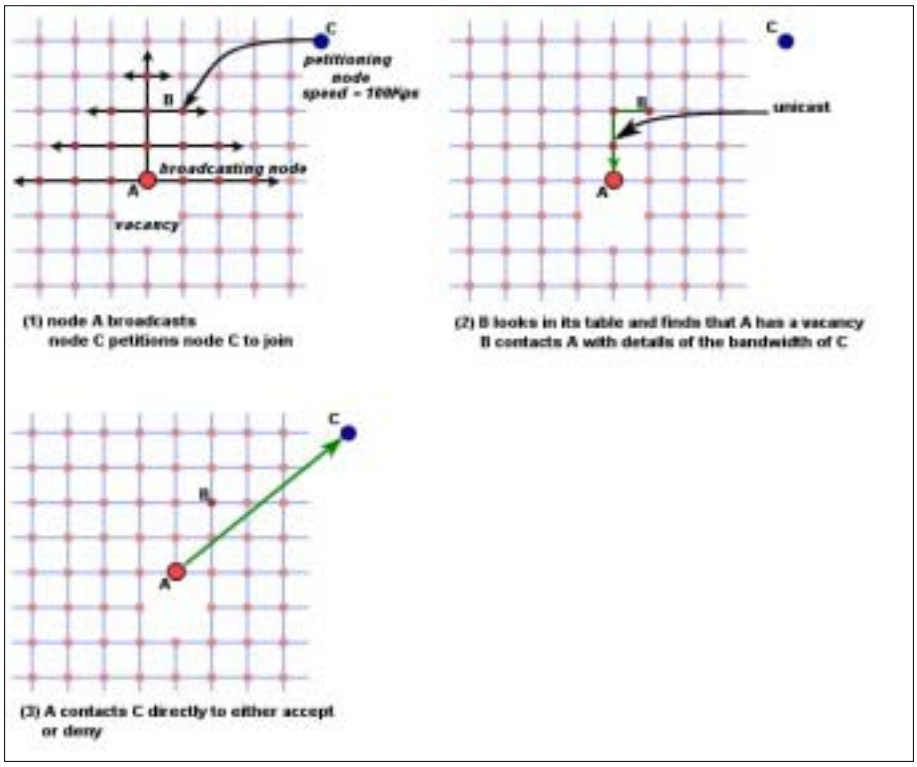

Figure 3.4 A node joining the lattice and filling a vacancy

When the selection is made, a packet containing the Internet address of the node petitioning to be connected is unicast in the appropriate direction, eventually to be picked up by the node with the neighbouring vacancy. It will then contact the petitioning node, accepting or denying the request. If the request is accepted then the petitioning node will be connected as described in section 3.3.2.

If the request is denied the process will start again with the refusing node taking responsibility this time for finding a suitable vacancy to be filled.

If for some other reason something fails (e.g. a reasonable time elapses without the petitioning node being contacted) then it will ask the first node it contacted to try again.

It should be noted that due to the method of matching nodes wishing to connect up with vacancies, i.e. on the basis of bandwidth, the lattice naturally organises itself into zones of different bandwidths. We will find zones of high bandwidth nodes and other zones of low-bandwidth nodes. The steepness of the gradient between these zones is determined by the precision with which we consider the bandwidth of the petitioning node. We use an 8-bit field to denote 8 different speed categories for a node requesting to connect. If we were to use a 4-bit field, we would expect the gradient to be much steeper. It follows that the steeper these gradients are, the more distinct zones we will find in the lattice.

3.7 Dealing with Misbehaving Nodes

It is necessary to introduce a mechanism that allows nodes to remove one of their neighbours if it is misbehaving. This is complicated by the necessity that all nodes surrounding the unsatisfactory neighbour must disconnect from it, not just the node experiencing difficulties with it.

A node experiencing difficulty with a neighbour can request the nodes surrounding it to disconnect via a special request that is sent using the coordinate seeking algorithm. It calculates the coordinates of the other three neighbouring nodes and sends a request to each of them.

In order to provide protection against misuse of this algorithm, a confirmation message must be exchanged before a node will drop the neighbour. A node receiving a request to drop a neighbour generates a short random code, and includes this in a packet that is sent back to the packet making the request. It then waits until it receives a response containing this code before disconnecting the node. This ensures that rogue nodes in the lattice cannot attack other nodes by forcing them to be disconnected. In order to do this they would have to be in possession of the code.

3.8 Connecting the Edges of the Lattice

The network as it has so far been described is a flat two-dimensional lattice of nodes, arranged in a grid-like pattern. Unfortunately nodes that are positioned at the very perimeter of the lattice enjoy less connectivity than the nodes in the centre of the lattice. However, for every node situated on an edge of the lattice, there is a matching node on the opposite edge. It is therefore possible to connect these nodes together. For example, a node on the eastern edge of the lattice will allow its counterpart on the western edge to connect to it as its east neighbour, and visa versa. These connections, termed *rubber connections*, provide a wrap-around in the lattice, so that traffic reaching the eastern edge of the lattice will reappear on the western side. This results in the lattice forming a torus, as depicted in Figure 3.5. In order to allow the lattice to grow, new nodes wishing to connect will always supersede neighbours that are connected via rubber connections.

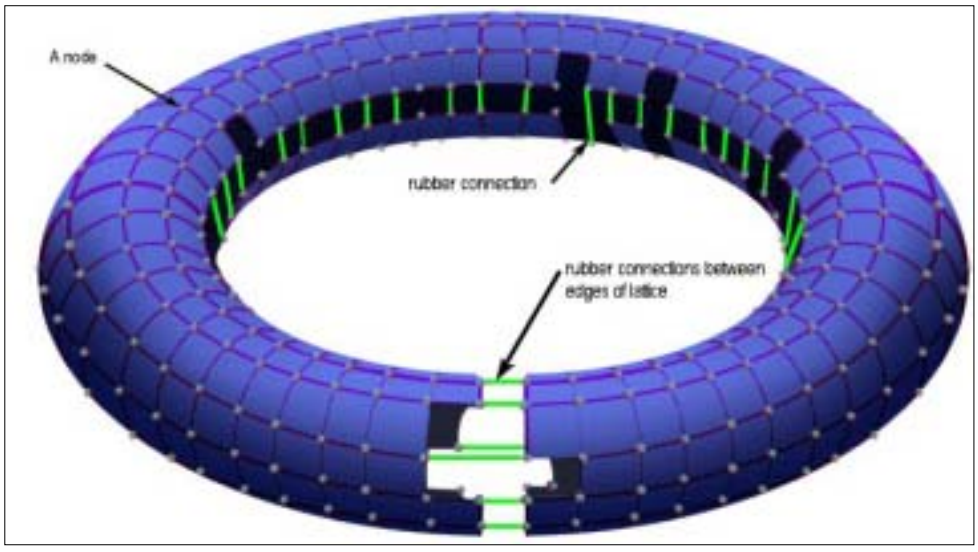

Figure 3.5 A visualisation of the connections between the edges of the lattice

When a node detects that it is positioned at the edge of the lattice, it will send a packet that includes its IP address directly across the lattice using the unidirectional algorithm described in section 3.4.4. When this packet reaches the corresponding node at the opposite edge the two nodes make contact and become neighbours.

There are three concerns regarding rubber connections:

- An isolated rubber connection may become clogged with traffic as all nearby nodes try to route their traffic through it. This is only an issue if nodes with rubber connections become sufficiently isolated from other nodes with such connections.

- Efforts to ensure that high-bandwidth nodes and low-bandwidth nodes are kept in their own zones (via the connection algorithm in section 3.6) are redundant at the edges, as high-bandwidth nodes may be opposite low -bandwidth nodes on the lattice and they will therefore connect directly to each other. This will of benefit to the low-bandwidth node, however it is possible that it may pose a bottleneck to the high-bandwidth node. Despite this, the high-bandwidth node benefits better by being connected to a low-bandwidth neighbour than no neighbour at all.

- Rubber connections between nodes mean that the lattice topology is distorted at the edges – horizontal distances will vary depending on latitude and visa versa. This has an acute effect on the coordinate-seeking algorithm, which relies on the lattice being representable by a Cartesian coordinate system. This can only be resolved by modifying the coordinate-seeking algorithm to behave slightly differently if a packet is sent via a rubber connection, as described in section 3.9.

3.9 Communication via Rubber Connections

Attempts will be made to use the coordinate-seeking algorithm to send packets over the rubber connections that join the edges of the lattice. As shown in Figure 3.6, the edges of the lattice may be uneven, and if this is the case the coordinate plane represented by the lattice becomes warped at the edges. The figure shows a coordinate-seeking packet that has been routed to the correct destination node, however the relative coordinates contained in the packet do not indicate this fact. The *i*-coordinate has a value of –1, indicating that the packet has yet to travel west. The specific problem is that the length of a rubber connection is undefined with respect to the *i*-*j* plane of the lattice, and that therefore either the *i* or *j*

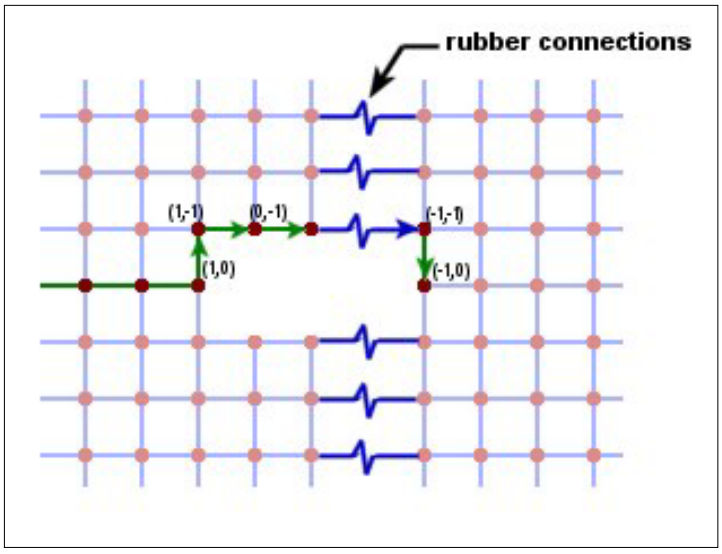

Figure 3.6 An example of the difficulties of routing coordinate-seeking packets over rubber connections

coordinate will become unreliable when the packet is sent over a rubber connection.

To avoid this situation, we slightly modify the coordinate-seeking algorithm. To do this two new fields are introduced into every coordinate-seeking packet. The

first is a flag that will be set if the packet is sent over a rubber connection. The second field indicates whether the rubber connection was horizontal or vertical.

If the packet crosses a horizontal rubber connection, its *i*-coordinate essentially becomes undefined. Similarly, if it crosses a vertical connection its *j*-coordinate becomes undefined. The objective of the algorithm is now to route the packet on the basis of the remaining valid coordinate. For example, if the packet crossed a horizontal rubber connection, only its *j*-coordinate can still be considered valid. It is therefore routed until this coordinate reaches 0 – i.e. it has arrived at the correct latitude. A corresponding procedure is carried out for vertical rubber connections.

This system enables a coordinate-seeking packet to traverse the edges of the lattice, however it is necessary to be aware that when used in this way the algorithm is imprecise in exactly which node receives the packet. This is discussed further in Chapter 6, *Future Work*.

3.10 Accommodating Low-Bandwidth Nodes

The bandwidth of a node is set upon the first use of the client to a general setting (e.g. modem, ISDN, DSL/Cable, ADSL, T1 etc.). Upon further use of the client the bandwidth in kilobytes per second is internally modified to reflect the maximum bandwidth that the node copes with. This statistic is used when negotiating a connection to the lattice, whereupon the node is usually connected to neighbours with matching bandwidth requirements.

As all nodes discriminate potential neighbours on the basis of bandwidth, the situation exists where high-speed nodes and low-speed nodes are grouped naturally into different zones on the lattice.

It is desirable that this be the case, otherwise the situation would exist where unreliable low-bandwidth connections would be critical to the propagation of all packets, thereby making the lattice unreliable. However those zones of the lattice that are populated by low bandwidth nodes will suffer the difficulty of being quite unresponsive, as packets must be potentially passed through several other low-

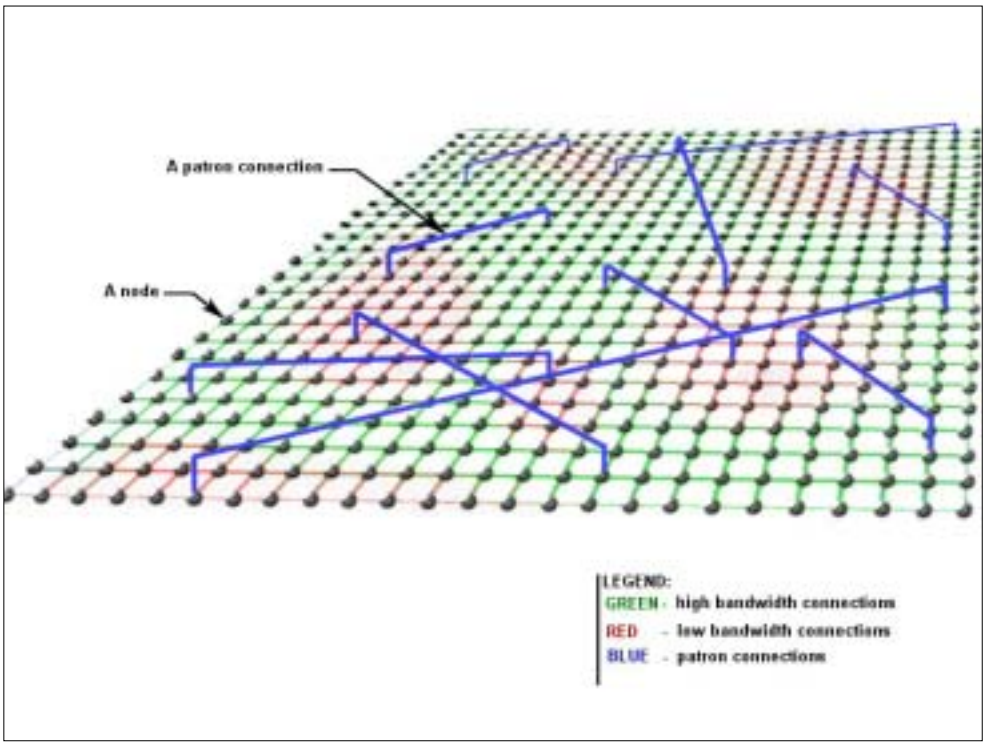

Figure 3.7 A visualisation of patron connections

bandwidth nodes before reaching the richer areas of the lattice.

To combat this, a special direct connection, termed a *patron connection*, can be established between a high-bandwidth node (the *patron*) and a low-bandwidth node (the *dependant*). This behaves much like a wormhole between the two nodes on the lattice. Figure 3.7 depicts a visualisation of patron connections on the lattice.

Patron connections are relevant only to broadcast and unicast packets. All broadcasts that the dependant node generates or encounters are sent to the patron node, which then proceeds to restart the broadcast at its own location. Any unicasts that the patron receives in response such a broadcast are forwarded back to the dependant.

It is not necessary that all low-bandwidth nodes should have such patron connections to high-bandwidth nodes, but just that such are sufficiently distributed so as to increase the overall performance within low-bandwidth zones.

A low-bandwidth node decides to try to establish a patron connection based on the following criteria:

- The responsiveness of the lattice that it experiences
- The frequency with which nearby nodes are making similar patron requests (the higher this frequency, the less likely it is that the node make such a request itself)
- Some probability function

A patron request packet is sent via the broadcast algorithm, and includes the node's IP address and connection bandwidth. A node is less likely to try to establish a patron connection if it observes many similar requests being broadcasted. This is a preventative measure against the possibility of a storm of patron requests, which could occur if the overall responsiveness of the lattice was lessened (perhaps due to some denial of service attack).

A node that receives a patron request evaluates it by determining whether it can reasonably support the low-bandwidth node. If it decides to accept the

connection, it contacts the node in question. If it declines it will continue to forward the request on.

Patron connections represent a controlled compromise of the rigid structure of the lattice. They benefit the design primarily in two ways:

- They allow nodes in low-bandwidth zones access to the resources of higher bandwidth zones, in a fashion that is in no way critical to the operation of the nodes in those higher bandwidth areas.

- They increase the inter-connectivity of the lattice from the perspective of nodes in lower-bandwidth zones. This results in the topology more closely resembling the "small-world" model, which [Hardin 1968] has shown to reduce the characteristic path-length. This means that broadcasts will have less distance to travel before reaching their targets.

It should be noted that an issue arises relating to the negotiation of non-trivial data transfers and patron connections. This is discussed in section 3.11.

3.11 Negotiating Non-trivial Data Transfers

Although it would be possible to transfer large amounts of data by using the unicast mechanism (analogous to data transfers in Freenet and Crowds), this is avoided for efficiency reasons. This section describes an alternative method, in which a proxy node is intelligently selected from the lattice to act as an intermediary for the data transfer. This enables anonymity to be preserved (unless the proxy node is in collaboration with one of the nodes who are uploading or downloading the data).

When a node wishes to download a hit that it has received in response to a search request, a proxied connection must first be established with the node that will be serving the data.

The challenge is to determine a proxy node in such a way that neither the requesting node nor the node providing the data has any influence on its identity. If either did have influence over this, it could simply select a node that it is in cooperation with, thereby removing the anonymity of the transaction.

To remove any possible influence, the decision of which proxy to use is handed to third node in the lattice, which is jointly chosen by the data-requester and the data-provider. The data-requester and the data-provider communicate with each other anonymously via the unicast algorithm in order to select this third node.

Figure 3.8 illustrates the sequence of events during a data transfer. The mechanism is as follows:

- A search-hit packet is generated the file-provider in response to a broadcast. A randomly chosen signed integer is included in the packet, which represents a relative *j*-coordinate in the lattice. The file-provider then unicasts this packet to the data-requester. Every node that handles this packet will decrement this number if it receives it from the south, increment it if it receives it from the north and leave it unchanged if it receives it from the east or west (in a similar fashion to the routing of coordinate-seeking packets).

- A download-request packet is unicast to the data-provider by the file-requester. This packet includes a signed integer representing a relative *i*-coordinate. This coordinate is decremented by every node on the path if it receives it from the west, incremented if it receives it from the west and

left unchanged if received from the north or south. The file-requester also includes its public key in the packet.

- Both nodes now have a set of relative coordinates that specify the position of a third node somewhere in the lattice. The data-requester and the data-requester now send a packet to this node using these coordinates and the coordinate-seeking algorithm. The third-party node is now charged with selecting a suitable proxy node for the transaction. It does so by examining a list of nodes that it is aware of, and selecting one on the basis of bandwidth and availability. It sends a packet out-of-lattice to the selected node, requesting it to become a proxy. If the potential proxy accepts, it notifies the data-provider to begin sending data. Otherwise it undertakes the duty of finding another proxy.

- The data-provider now generates a shared key, encrypts this key with the data-requester's public key, and sends the encrypted key to the proxy node. It then begins sending encrypted data to the proxy. The proxy forwards both the shared key and the encrypted data on to the data-requester as it receives them.

If the proxy happens to disconnect from the lattice, then the nodes will automatically ask the jointly selected third node to select a new one, and the download will be resumed. If the jointly selected node has disconnected from the lattice then a new one will be selected.

The above method ensures anonymity for both the data-requester and the data-provider from each other in a relatively efficient way. The only cost to efficiency is the unavoidable use of a proxy.

It should be noted that a complication exists when the data-requester and the data-provider are communicating via unicasts that involve a patron connection being crossed. The negotiation of the third party node that performs the proxy selection is dependent on each node on the path of the unicast adjusting a relative coordinate to reflect the packet's new position. Currently a patron node does not have enough information to perform this adjustment. Chapter 6, *Future Work*, describes an algorithm that overcomes this difficulty by allowing a patron node to measure its distance from its dependant node.

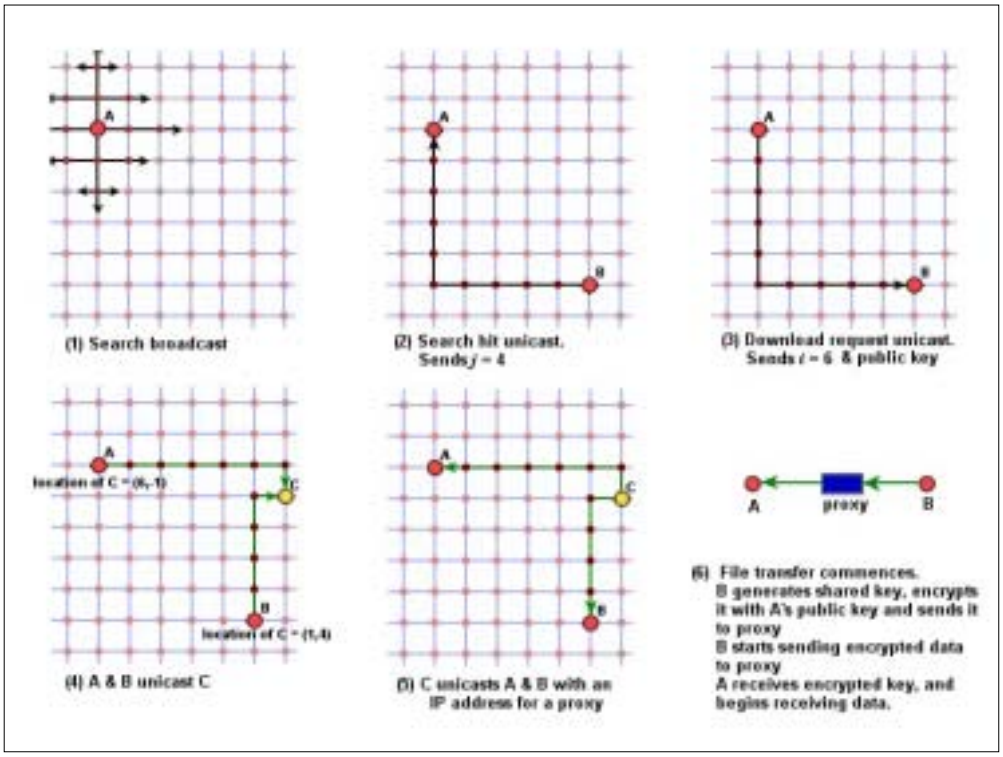

Figure 3.8 Negotiation of a non-trivial data transfer

Chapter 4

IMPLEMENTATION

It was undertaken to implement a client operating the protocol for the purposes of file sharing. This was coded in C++, which offers the advantage of speed and object-orientation. This chapter describes the implementation of the client. A description of the protocol definition units can be found in Appendix A, and a state transition diagram describing the logic behind the handling of packets can be found in Appendix B.

4.1 Node State Information

Each node in the lattice must maintain certain state information necessary to its operation. This section briefly describes this state information and the C++ classes used to store it.

4.1.1 File Index

The file_index class was implemented to allow fast retrieval of information about files that are available for download by other nodes from the client. This class uses a hash table to store information about all shared files, such as the filename, path and size. When the class is constructed it recursively examines a list of directories and populates the table with information about each file that it finds. A list of files matching a particular search can be retrieved from this class by passing the search string into the appropriate method. This method tokenises the search string for keywords and matches these keywords against tokens in the filenames of shared files.

This class is accessed each time a file search broadcast packet is received.

4.1.2 File Hit Index

The file_hit_index class maintains a hash table that stores information about all hits that have been received for file searches. A new entry is added to this table each time a file hit unicast packet is received in response to a file search broadcast generated by the client. This class is used by the front end to display the results of a search.

4.1.3 Download Index

The download_index_class maintains information about all files currently being downloaded by the client. This information consists of the name of the file, the shared key that it is encrypted with, and the UTID of the transaction that the download is part of. This information is accessed every time the client receives more file data.

4.1.4 Upload Index

The upload_index_class maintains information about files that the client is currently negotiating to upload to another node. This allows the client to retrieve information about which file to upload in a particular transaction.

4.1.5 Received Packet Index

The received_packet_index is a class that maintains a hash table containing information about all packets that have recently been received from neighbours. This information consists of the neighbour that each packet was received from, the UTID of each packet and (if a packet is a broadcast) an 8-bit field extracted from the packet indicating any vacancies on the path that the packet has taken.

This information is accessed in order to be able to route unicast packets correctly and also to find information about vacancies when negotiating the connection of a new node.

4.1.6 Sent Packet Index

The sent_packet_index class maintains a hash table about packet that the client has recently generated and sent. This is accessed in order for the unicast algorithm to function. Every node receiving a unicast can access this table to ascertain whether to UTID of a unicast packet matches that of a packet that it recently generated and sent. If a match is found then the client knows that it is the destination of the unicast.

4.1.7 Select Proxy Index

The select_proxy_index class is used when the client has been selected by other nodes to select a proxy for a data transaction between them. The class maintains a hash table containing address and bandwidth information for both the node requesting the data and the node providing the data for the transfer. The client enters this information as it receives it from the two parties. Once both nodes have provided the relevant data it can go about selecting the actual proxy.

4.1.8 Proxy Transaction Index

The proxy_transaction_index class is used when the client is acting as a proxy to two nodes transferring data. It uses a hash table to lookup the address of the node that data should be forwarded to when the client receives data from the node providing the data. As it is possible that the client may be acting as a proxy for more than one transaction at a time, this address information is indexed by the UTID of the relevant transaction.

4.1.9 Lattice Bandwidth

The lattice_bandwidth class is used to estimate the average bandwidth of other nodes on the lattice. As documented in Appendix A, every broadcast packet includes information about the bandwidth of the node that originated it. This is used to update this class every time a broadcast is received. The benefit of this is that the hops-to-live value of every broadcast that this client generates can be

adjusted to a value that is lattice is capable of handling, on the basis of the lattice's average bandwidth capability.

4.1.10 Node Bandwidth

The node_bandwidth class is used to estimate the maximum bandwidth of the internet connection that the client is using. This is performed by updating the class with the throughput achieved during each non-trivial data transfer. It is essential that the client have a realistic estimate of its bandwidth, as this information forms the basis for the organisation of the lattice.

4.1.11 Other State Information

Basic state information is stored using the global_parameters struct. This holds information such as which of its neighbours are currently connected, whether the client currently has a patron node or a dependant node and which (if any) of the neighbours are connected via a rubber connection. The addresses and encryption keys of the neighbours, patron node and dependant node are also held in this struct.

It is also used to hold pointers to each of the classes described in sections 4.1.1 to 4.1.10. Collectively these classes manage all the state information not held by the global_parameters struct. This allows all state information relating to the client to be accessed through this struct. By passing a pointer to this struct to each of the constructor of the classes that provide the main functionality of the client, we allow each of them access to all the state information that they may need.

4.2 Packet Formatter Class

The packet_formatter class provides methods to construct each of the packets listed in Appendix A. Each of these methods takes parameters relevant to the

particular packet in question, and returns a pointer to area in memory where the completed packet has been stored. This pointer must be deleted after the packet has been sent.

4.3 Packet Sender Class

The packet_sender class provides methods to send packets using each of the communication modes described in section 3.4. For example, a pointer to a packet to be broadcast may be passed to the method that sends packets using the broadcast algorithm.

4.4 Packet Handler Class

The packet_handler class manages the logic involved whenever a packet is received from another node. When this class is constructed, a new thread is started that waits for new incoming connections. Whenever a new connection is accepted, a new thread is spawned to receive the incoming packet. When this data has been received, the class's main method is called to process the packet.

Processing the packet may result in state information being modified, and new packets being sent in response. A complete state transition diagram is provided in Appendix B that specifies the processing of an incoming packet.

4.5 Front End GUI

The class HttpFrontEnd implements a front end for the client. This operates as a simple web server with basic CGI (Common Gateway Interface) functionality. This web interface is accesses by entering http://localhost in a web browser. The client returns a web page that allows searches to the initiated and downloads to be started based on displayed search hits. It is also possible to access options that

allow various setting to be changed, such as the default upload/download directory and the default port that the client operates on.

By implementing the front end in this way we enable the client to be accessed remotely, even by multiple users simultaneously. The implementation code also becomes highly portable as no platform specific user interface libraries are used.

Chapter 5

EVALUATION

5.1 Performance Analysis

The aim of an efficient protocol is to minimise the level of traffic that is sent on the network while avoiding any impact on functionality. This is necessary so that low-bandwidth nodes are able to handle the level of traffic and therefore function properly as peers.

As a client operating the protocol has not yet been deployed it is not possible to provide data on the real-world performance of the protocol. We can however derive a mathematical model from which we can estimate the average amount of traffic that each node handles.

Individual nodes performing searches on the network have the greatest impact on the level of traffic experienced on the lattice. Other communications are trivial in comparison and are neglected in the following mathematical model.

The first step in the derivation is to calculate the number of nodes contacted when a broadcast is made. To do this we visualise the lattice as a coordinate plane, with the node generating the broadcast at the origin. We first examine just one quadrant of the plane as shown in Figure 5.1.

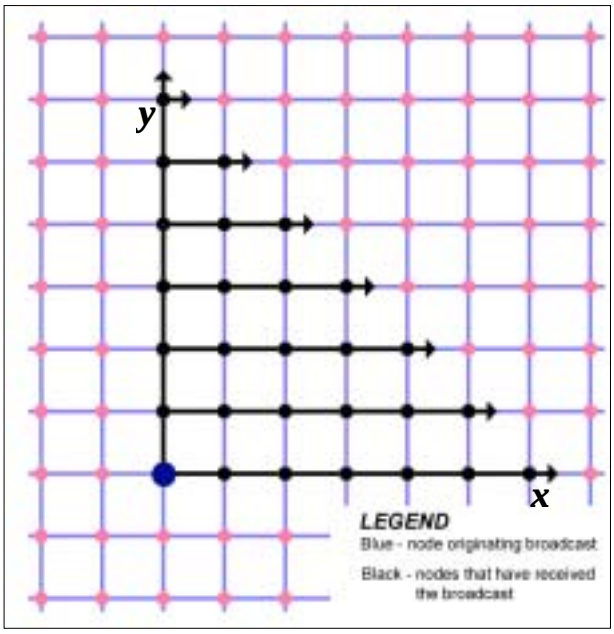

Figure 5.1 One quadrant of the propagation of a broadcast

In the figure we can see that the broadcast has so far travelled 6 hops from the origin. If we temporarily ignore the nodes that lie on the y-axis, directly above the origin, we see that the number of nodes contacted is 6 + 5 + 4 + 3 + 2 + 1. This resembles the series:

$$\sum_{n=0}^{n=h} k_n$$ [Expression 1]

Where *h* is the number of hops taken by the packet

If we now include the nodes on the y-axis, and extend this expression to include the other three quadrants we get:

$$2h + 4\sum_{n=0}^{n=h} k_n - 2h$$ [Expression 2]

The first *2h* term accounts for the nodes that lie on the y-axis, and the second *2h* term corrects for the fact that by multiplying the series, we accounted for the nodes on the x-axis twice.

By cancelling terms and converting to an equivalent mathematical expression we get:

$$4\sum_{n=0}^{n=h} k_n = 4\left(\frac{h^2+h}{2}\right) = 2h^2 + 2h \qquad \text{[Expression 3]}$$

If we let h_{max} represent the hops-to-live of an average broadcast, then it follows that the number of nodes contacted by an average broadcast is $2h_{max}^2 + 2h_{max}$, assuming a large number of nodes on the lattice.

There are two levels of traffic that a node may handle for any search performed on the lattice. A node will handle the first level of traffic if it lies on the y-axis with respect to an originating node, and forwards broadcast traffic to three of its neighbours – and therefore may receive related unicast traffic from those three neighbours. These nodes are termed *type-A nodes* with respect to any particular broadcast. All nodes that are not type-A nodes handle the second level of traffic as they send broadcast packets to only one neighbour, and receive associated unicasts only from that neighbour. These are termed *type-B nodes*. The number of type-A nodes for a particular broadcast is equal to $2h_{\max}$. The fraction of nodes that this represents is:

$$\frac{2h_{\max}}{2h_{\max}^2 + 2h_{\max}} \Rightarrow \frac{1}{h_{\max}+1} \qquad \text{[Expression 4]}$$

It follows that the fraction of type-B nodes is:

$$1 - \frac{1}{h_{\max}+1} \Rightarrow \frac{h_{\max}}{h_{\max}+1} \qquad \text{[Expression 5]}$$

We can now calculate the level of traffic that each of these types of nodes handle. The cost of receiving and forwarding a broadcast is straightforward to calculate. Type-A nodes must receive a broadcast packet and forward it three times –

resulting in $4S_b$ bytes of data being transferred through the node, where S_b represents the size of an average broadcast packet. Type-B nodes must merely receive the broadcast and send it once, resulting in a cost of $2S_b$. The number of unicasts that type-A and type-B nodes receive in response to the broadcast is more complicated to calculate.

A broadcast packet will be forwarded to ($h_{max} - h$) nodes after passing through a type-B node, where h represents the number of hops that the packet has already taken from the origin. If we let P_u represent the probability that a node will reply to a search broadcast, then we can say that the number of unicasts that a type-B node will handle is $P_u(h_{max} - h)$. Taking a median value of h, this has an average value of $P_u(h_{max}/2)$.

A type-A node forwards broadcasts to a greater number of nodes than a type-B,

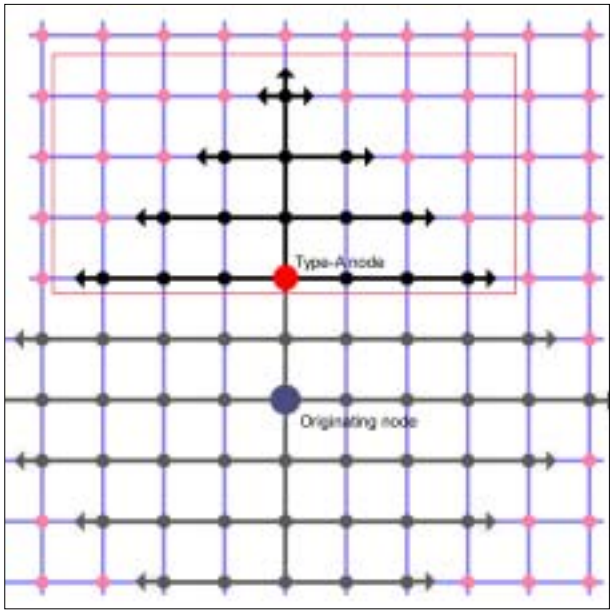

Figure 5.2 A broadcast forwarded by a type-A node

as shown in Figure 5.2.

To calculate the number of nodes on the path of the broadcast after it passes through the type-A node, we use expression 1 as follows:

$$\text{No. nodes contacted} = 2\left(\sum_{n=0}^{n=h_{\max}-h} k_n\right) + \left(h_{\max} - h\right)$$

$$\Rightarrow 2\,\frac{\left(h_{\max} - h\right)^2 + \left(h_{\max} - h\right)}{2} + \left(h_{\max} - h\right)$$

$$\Rightarrow \left(h_{\max} - h\right)^2 + 2\left(h_{\max} - h\right) \qquad \text{[Expression 6]}$$

Where h = no. of hops the packet has already taken

The average value of this term over all possible values of h is:

Avg. no. nodes contacted

$$= \frac{\sum_{h=0}^{h=h_{\max}} \left(\left(h_{\max} - h\right)^2 + 2\left(h_{\max} - h\right)\right)}{h_{\max}}$$

$$= \frac{\sum_{h=0}^{h=h_{\max}} \left({h_{\max}}^2 - 2h_{\max}h + h^2 + 2h_{\max} - 2h\right)}{h_{\max}}$$

$$= {h_{\max}}^2 + 2h_{\max} + \frac{\sum_{h=0}^{h=h_{\max}} h^2 - 2h_{\max}\sum_{h=0}^{h=h_{\max}} h - 2\sum_{h=0}^{h=h_{\max}} h}{h_{\max}}$$

$$= {h_{\max}}^2 + 2h_{\max} + \frac{\left({h_{\max}}^3/3\right) + \left({h_{\max}}^2/2\right) + \left(h_{\max}/6\right) - 2h_{\max}\left(\left({h_{\max}}^2/2\right) + \left(h_{\max}/2\right)\right) - 2\left(\left({h_{\max}}^2/2\right) + \left(h_{\max}/2\right)\right)}{h_{\max}}$$

$$= {h_{\max}}^2 + 2h_{\max} + \left({h_{\max}}^2/3\right) + \left(h_{\max}/2\right) + 1/6 - {h_{\max}}^2 - h_{\max} - h_{\max} - 1$$

$$= \left({h_{\max}}^2/3\right) + \left(h_{\max}/2\right) - 5/6 \qquad \text{[Expression 7]}$$

If P_u is the probability that a node will respond to the search with data, then the number of unicasts that a type-A node will receive is $P_u((h_{max}^2/3)+ (h_{max}/2)\text{-}5/6)$.

We can now bring the above equations together to estimate the average amount of traffic that an average node must handle for each broadcast.

Avg. traffic per node per broadcast =
(fraction of type-A nodes)$(2\ P_u((h_{max}^2/3)+ (h_{max}/2)\text{-}5/6)S_u + 4\ S_b)$
+ (fraction of type-B nodes)$(2P_u(h_{max}/2)S_u + 2\ S_b)$

$$= \left(\frac{1}{h_{\max}+1}\right)\left(2P_u\left(\left(h_{\max}^{\ 2}/3\right)+\left(h_{\max}/2\right)-5/6\right)S_u + 4S_b\right)$$
$$+\left(\frac{h_{\max}}{h_{\max}+1}\right)\left(2P_u\ h_{\max}\Big/2\ S_u + 2S_b\right)$$
$$= \left(\frac{1}{h_{\max}+1}\right)\left(\frac{2}{3}P_u h_{\max}^{\ 2} S_u + P_u h_{\max} S_u - \frac{10}{6}P_u S_u + 4S_b + P_u h_{\max}^{\ 2} S_u + 2h_{\max} S_b\right)$$
$$= \left(\frac{1}{h_{\max}+1}\right)\left(\left(\frac{2}{3}P_u S_u + P_u S_u\right)h_{\max}^{\ 2} + \left(P_u S_u + 2S_b\right)h_{\max} + \left(-\frac{10}{6}P_u S_u + 4S_b\right)\right)$$

[Expression 8]

Where,
h_{max} = avg. time-to-live of broadcast packets
S_u = avg. size of a unicast packet
P_u = probability of a unicast response per node
S_b = avg. size of a broadcast packet

We now introduce a factor, F_b, which represents the rate at which an average node starts searches on the lattice. If we multiply expression 8 by F_b and by expression 3 (the total number of nodes within range of an average broadcast), we get an expression giving us the bandwidth cost of participation for an average node. This gives us:

Bandwidth cost per node =

$$\left(\frac{10}{3}P_u S_u F_b\right)h_{\max}^{\ 3} + \left(2P_u S_u F_b + 4S_b F_b\right)h_{\max}^{\ 2} + \left(-\frac{10}{3}P_u S_u F_b + 8S_b F_b\right)h_{\max}$$

[Expression 9]

We can estimate the value of S_u and S_b from the protocol definition (Appendix A). Figure 5.3 shows the relationship between the bandwidth load on each node, the average hops-to-live value assigned to broadcasts and the probability factor P_u.

The graph assumes a value for F_b of 0.0016 – i.e. that an average node performs a new search every 10 minutes.

The HTL value (which is equivalent to h_{max}) can be controlled by design; however P_u is uncontrollable as it relates to usage patterns by individual users. As the client is not yet deployed no information is available that would allow us to estimate this factor. Nonetheless the graph clearly demonstrates that most of the values of h_{max} and P_u plotted lie below the 56000 bits/second level. This means that even in

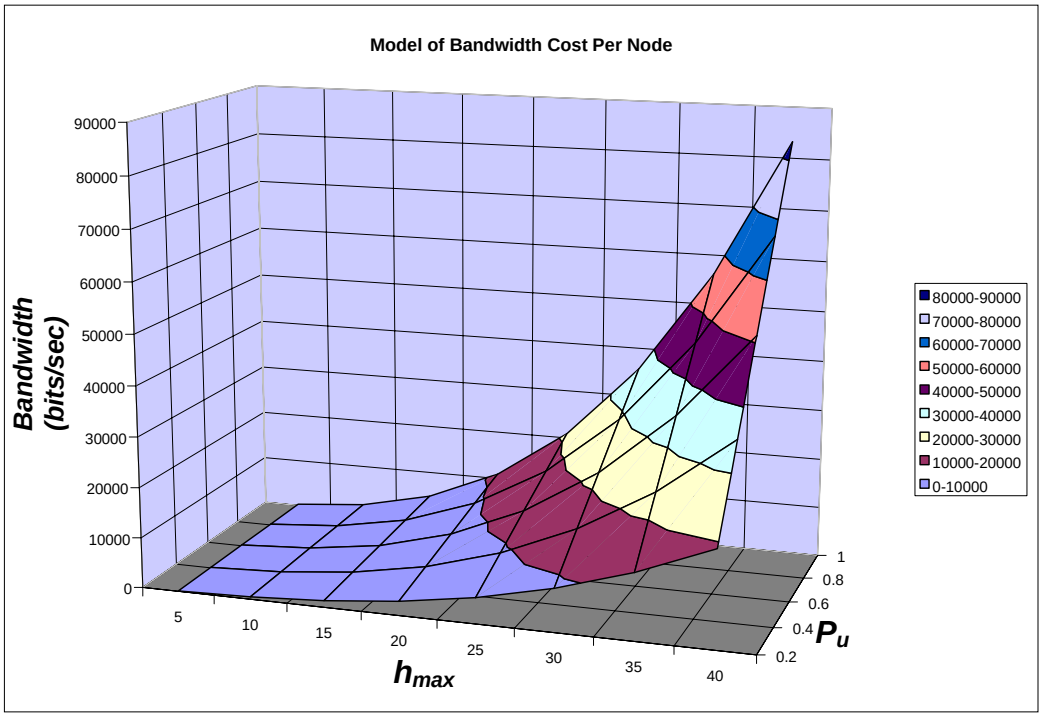

Figure 5.3 A graph of the bandwidth cost of participation on the network

the worst-case scenario, where P_u = 1 (i.e. all nodes reply to all broadcasts), modem users are still able to effectively and actively participate up to a HTL of approximately 40.

Neglecting the effect of patron connections, expression 3 shows that a HTL of 40 still represents a possible 3280 contactable peers, which is an acceptable number.

It can therefore be concluded that the protocol does indeed achieve the level of efficiency that the design set out to achieve.

5.2 Scalability

It is necessary that the protocol be able to support the participation of any number and type of nodes.

The type of node that is of particular concern are those that are connected via dialup connections. Such nodes are considered to be the main cause for the scalability difficulties faced by Gnutella [DSS 2000b]. As shown in section 5.1, users connected at dialup bandwidths of 56 Kbits/sec do not in fact pose a problem to the protocol at all, and they are fully capable of participation on the network without causing any kind of bottleneck.

Support for any number of potential participants is provided through the use of a hops-to-live (HTL) value in all broadcasts. Expression 3 shows that as a result only $2h_{max}^2+2h_{max}$ nodes are contactable by any broadcast, where h_{ma} = HTL. This means that nodes communicate with a maximum of $2h_{max}^2+2h_{max}$ peers, and the addition of nodes to the lattice above and beyond this number does not have a global effect. Expression 9 demonstrates the bandwidth load on each node when there are greater than $2h_{max}^2+2h_{max}$ participants. From this expression we can see

that when there are a large number of participants, the level of traffic that a node must handle is in no way related to the total size of the lattice.

It is therefore expected that the lattice will be able to scale to an arbitrary number of participants when it is deployed.

5.3 Possible Attacks on Anonymity

The design set out to achieve both reader-anonymity and server-anonymity. These aims were largely satisfied and it would be difficult for any party to associate either a reader or a server with any piece of data, or request for that data. There are ways in which this anonymity may be attacked however, and these are examined briefly here.

The most direct attack that could be made on the system is to saturate it with collaborating nodes. This would challenge the anonymity in two ways. Firstly, the lattice could be sufficiently saturated that the enemy nodes will have entirely surrounded certain nodes. If this were the case, broadcasts generated by the surrounded nodes could be directly linked to them. Secondly, if the lattice is sufficiently saturated, entrapment may become possible as an enemy may be randomly chosen to act as a proxy for a data transfer in which an enemy node is already either acting as the data-requested or the data-provider. If this were to happen, the identity of the non-enemy node would be revealed.

The resources required to mount such an attack are dependant on the total number of nodes participating in the lattice and would be very significant. Under the best of circumstances such an attack would be very unreliable as it is impossible for an enemy to complete saturate the lattice (i.e. remove all honest nodes from the lattice). It should also be noted that it would be impossible to specifically target particular nodes.

Anonymity could also be attacked through the use of timing attacks and size analysis. This would require the enemy to have the ability to observe incoming and outgoing traffic on a significant number of nodes, and also to have a presence on the lattice. Packets are encrypted before being sent between neighbours, so it is impossible for an observer to deduce anything from the data itself. It may be possible however for the observer to correlate information about the size of particular packets and the timing with which they are received and sent at many nodes. This may enable the enemy to track the path that packets take in the lattice, gather information about the layout of the lattice, or possibly deduce the origin of packets.

Mounting such an attack would be extremely complicated, and would require access to the majority of ISP's through which nodes are operating. Precautions to thwart such an attack are discussed in Chapter 6, *Future Work*.

Chapter 6

FUTURE WORK

6.1 Anonymity

As mentioned in section5.3, the anonymity of the system is susceptible to analysis of the timing and size of packets sent from any particular node. It would be worthwhile to counter this type of attack.

To counter timing attack all nodes could batch send all packets at a standard interval, on the order of a fraction of a second. To counter a size analysis attack, all nodes could use a combination of splitting and padding packets so that they are all of a standard size. Both of these measures would be relatively uncomplicated to implement and would significantly fortify the system against enemy analysis in general.

6.2 Topology

6.2.1 Rubber Connections

Section 3.2 discussed the use of rubber connections on the lattice. A difficulty would arise in the design if nodes lying on opposite sides of the edge of the lattice attempted to negotiate a data transfer, as the coordinates that they cooperatively negotiate for the third node (that selects a proxy for them) may become distorted when the communication messages traverse the rubber connections connecting the two edges. The reason for this difficulty is that the edges of the lattice may be uneven, thereby distorting the regularity that enables the network to be visualised

as a coordinate plane. When the two nodes attempt to contact the third node via coordinate-seeking message, it is possible that the coordinates that they have negotiated will not uniquely identify a third node at all.

It is desirable that this deficiency be overcome in some way. It is trivial for a node with a rubber connection to measure the number of hops between it and its counterpart on the opposite side of the lattice. Such a mechanism is actually implemented already in the protocol definition in Appendix A. One must simply send a unidirectional packet across the lattice that contains a counter that is incremented with each hop that is taken. The value of this counter when it is received at the opposite edge corresponds directly to the diameter of the lattice at that point. Using this information, nodes at the opposite ends of a rubber connection can mutually agree for the rubber connection to be equivalent to a number of hops. This number can be chosen so that the result of adding it to the actual diameter of the lattice is a universally agreed large number, such as 100,000.

This would mean that every rubber connection would be negotiated to represent a number of hops on the lattice such that the virtual diameter of the lattice at any point (including the rubber connections themselves) would be 100,000. The edges of the lattice are thus evened out, and coordinate-seeking packets would always be sent correctly.

6.2.2 Patron Connections

Section 3.10 outlined the design of patron connections. In an analogous situation to that described in the previous section, patron connections distort the regularity of the lattice, and could result in coordinate-seeking messages being sent to the incorrect nodes. This should be corrected, and a possible solution is similar to that outlined for rubber-connections.

A patron-node and a dependant-node could use a unicast packet holding a set of *i,j* counters to determine the relative distance between them. As the unicast packet is forwarded, each node on the path modifies the coordinates to reflect the hop the packet has just taken. When the packet reaches its destination, the coordinates will reflect the horizontal and vertical distances between the two nodes. These distances can be used to correct any packets the patron and dependant exchange that contain coordinate fields.

6.2.3 Multiple Lattice Layers

A possibility that should be carefully investigated is that of separating the lattice into multiple layers, on the basis of bandwidth. This would allow for a more strict separation between nodes of different bandwidths.

The lattice is currently organised so that low and high bandwidth nodes separate into different areas with patron connections stretching across the surface between such areas. As all nodes are on the same layer low-bandwidth nodes must deal with the traffic that high-bandwidth nodes are generating. As it is very likely that the users of high-bandwidth nodes would often be uninterested in downloading data from low-bandwidth nodes, much of the traffic may be redundant.

By separating the lattice into layers we can allow each node to specify which layers it would like each broadcast to reach. There would exist a layer for each major bandwidth category that it is decided to define, and each layer would be allowed to establish patron connections with each layer higher than it. Unlike the current design, patron connections could be used to send traffic bi-directionally. By default broadcasts from low bandwidth layers would be sent to patrons in higher layers, and if explicitly desired broadcasts from higher layers could be sent to lower layers.

This refinement of the architecture requires further consideration, but may well prove to be worthwhile.

6.3 Control of Remote Use

The front end of the client is currently designed to operate using HTTP on the peer machine. It is generally considered a benefit that this allows remote use of the node from any connected machine with a web browser, however it is desirable that some kind of user control be included. This can be simply resolved by allowing a user to enter hostnames or IP addresses into the settings page of the client, and allowing only remote requests from those addresses to be accepted.

6.4 Optimising Node Placement

The possibility of optimising the neighbours that a node is given to be those with a similar area of interest should be investigated. This would allow us to structure the network on the basis of categories of data, which could allow the network to become more efficient in satisfying requests. Freenet operates a similar mechanism, using the concept of a *key space* [Langley 2001a]. The key space allows areas of interest to be mathematically mapped to an integer. A node can determine how close in interest it is to another node by using a *closeness function* that operates on the key space. This is enabled in Freenet by its document-anonymity and the resultant necessity that data is explicitly inserted into the network. To design a similar mechanism into the lattice, some method of categorising arbitrary data would have to be devised, and this would be highly complicated. Nonetheless, this would prove a very interesting addition to the protocol.

6.5 Client Implementation

A full implementation of a file-sharing client for the protocol is nearly complete, however requires some further refinement and debugging. It is the author's intention to incorporate those areas discussed in Chapter 6, and to release the completed client under the GNU general public licence [HREF 6].

Chapter 7

CONCLUSIONS

This project has been successful in three ways.

- The aims as set out in section 3.1 have been met and a distributed peer-to-peer network has been designed that allows people to communicate securely and anonymously with each other. It has been further demonstrated that the network should scale to any number of potential users and will not suffer if a popular application is designed for it.

- The protocol has been designed in an application-agnostic way, and much potential remains for applying it to a diverse range of areas. File sharing is relatively straightforward and obvious application; however many others exist, such as chat forums and even distributed computation. SETI@home [Anderson 2001] and Popular Power [HREF 7] have demonstrated the viability of peer-to-peer networks to form such a platform for computing.

- The protocol has also taken a unique approach to a problem that there is much interest in solving. The design and approaches documented here should be instructive to future systems, and represent a step forward in peer-to-peer design.

BIBLIOGRAPHY

[Adar 2000] Adar, E., Huberman, B. A. 2000. The impact of Free-Riding on Gnutella. Online at http://www.firstmonday.org/issues/issue5_10/adar/index.html

[Adler 2000] Adler, Stephen. 2000. *The Slashdot Effect: An Analysis of Three Internet Publications*. http://ssadler.phy.bnl.gov/adler/SDE/SlashDotEffect.html

[Anderson 2001] Anderson, D. 2001. SETI@home. In Oram, A., (ed) *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*, O'Reilly and Associates, Inc., Sebastopol, California.

[Bricklin 2001] Bricklin, D. 2001. Cornucopia of the Commons. In Oram, A., (ed) *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*, O'Reilly and Associates, Inc., Sebastopol, California.

[Clarke 1999] Clarke, I. 1999. A Distributed Decentralised Information Storage and Retrieval System. Unpublished undergraduate thesis, University of Edinburgh. Online at http://freenet.sourceforge.net

[Dingledine 2000] Dingledine, R. 2000. The Free-Haven Project: Design and Deployment of an Anonymous Secure Data Haven. Unpublished Masters thesis, Massachusetts Institute of Technology.

[Dingledine 2001] Dingledine, R., Freedman, M. J., Molnar, D. 2001. Free Haven. In Oram, A., (ed) *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*, O'Reilly and Associates, Inc., Sebastopol, California.

[DSS 2000a] Clip2 Distributed Search Solutions 2000. The Gnutella Protocol Specification v0.4. Online at http://dss.clip2.com/GnutellaProtocol04.pdf

[DSS 2000b] Clip2 Distributed Search Solutions 2000. Bandwidth Barriers to Gnutella Network Scalability. Online at http://dss.clip2.com/dss_barrier.html

[Goodwins 2000] Goodwins, R. 2000. *Echelon: How it Works.* ZDnet News UK. Online at http://www.zdnet.co.uk/news/2000/25/ns-16248.html

[Hardin 1968] Hardin, G. 1968. The Tragedy of the Commons. *Science,* (162) p. 1243.

[Hong 2001] Hong, T. 2001. Performance. In Oram, A., (ed) *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*, O'Reilly and Associates, Inc., Sebastopol, California.

[Houston 2000] Houston, B. 2000. The Grid P2P Topology Proposal. Online at http://www.exocortex.org/p2p/grid.html

[HREF 1] The Financial Express 2000. *Napster among fastest-growing Net tech.* Online at http://www.financialexpress.com/fe/daily/20001009/fco09018.html

[HREF 2] O'Reilly Network Weekly Open Source Roundtable. 2000. *O'Reilly's Peer-to-Peer Summit.* Online at http://www.oreillynet.com/pub/a/linux/2000/09/22/rt.html

[HREF 3] O'Reilly Network Weekly Open Source Roundtable. 2000. *Is Carnivore Eating You?* http://www.oreillynet.com/pub/a/linux/2000/07/18/rt.html

[HREF 4] Cyber-Rights & Cyber-Liberties 2001. *Echelon Watch.* Online at http://www.cyber-rights.org/interception/echelon/

[HREF 5] The Cypherpunks remailer network. Online at http://www.csua.berkeley.edu/cypherpunks/remailer/

[HREF 6] GNU General Public Licence. Online at http://www.gnu.org/copyleft/gpl.html

[HREF 7] Popular Power homepage. Online at http://www.popularpower.com/index2.html

[Kan 2001] Kan, G. Gnutella. 2001. In Oram, A., (ed) *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*, O'Reilly and Associates, Inc., Sebastopol, California.

[Langley 2001a] Langley, A. 2001. Freenet. In Oram, A., (ed) *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*, O'Reilly and Associates, Inc., Sebastopol, California.

[Langley 2001b] Langley, A. 2001. Mixmaster Remailers. In Oram, A., (ed) *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*, O'Reilly and Associates, Inc., Sebastopol, California.

[Loeb 1998] Loeb, V. 1998. *NSA Admits to Spying on Princess Diana.* Washington Post. Online at http://washingtonpost.com/wp-srv/national/daily/dec98/diana12.htm

[McCullagh 2000] McCullagh, D. 2000. *Carnivore Review Team Exposed!* Wired News. Online at http://www.wired.com/news/politics/0,1283,39102,00.html

[O'Reilly 2001] O'Reilly, T. 2001. Remaking the Peer-to-Peer Meme. In Oram, A., (ed) *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*, O'Reilly and Associates, Inc., Sebastopol, California.

[Oram 2001a] Oram, A. 2001. Preface. In Oram, A., (ed) *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*, O'Reilly and Associates, Inc., Sebastopol, California.

[Oram 2001b] Oram, A. 2001. Afterword. *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*, O'Reilly and Associates, Inc., Sebastopol, California.

[Rabin89] Rabin, M. O. 1989. Efficient Dispersal of Information for Security, Load Balancing and Fault Tolerance. *Journal of the ACM*, vol. 36, no. 2, (335).

[Reiter 1997] Reiter, M. K., Rubin, A.V. 1997. Crowds: Anonymity for Web Transactions. *DIMACS Technical Report*, 97(15).

[Rothenberg 2000] Rothenberg, M. 2000. *Europe weighs Echelon threat.* ZDnet News UK. Online at http://www.zdnet.co.uk/news/2000/11/ns-14310.html

[Shirky 2001] Shirky, C. 2001. Listening to Napster. In Oram, A., (ed) *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*, O'Reilly and Associates, Inc., Sebastopol, California.

[Smith 2000] Smith, S. P. et al. 2000. *Independent Technical Review of the Carnivore System: Final Report*. United States Department of Justice. Online at http://www.usdoj.gov/jmd/publications/carniv_final.pdf

[STOA 1999] European Parliament Scientific and Technological Options Assessment. 1999. *Development of Surveillance Technology and Risk of Abuse of Economic Information: an Appraisal of Technologies for Political Control*. European Parliament Directorate General for Research, Luxembourg.

[Truelove 2001] Truelove, K. 2001. *Gnutella and the Transient Web*. O'Reilly Network. http://www.openp2p.com/pub/a/p2p/2001/03/22/truelove.html

[Waldman 2001] Waldman, M., Cranor, L. F., Rubin, A. 2001. Publius. In Oram, A., (ed) *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*, O'Reilly and Associates, Inc., Sebastopol, California.

[Watts 1998] Watts, D. J., Strogatz, S. H. 1998. Collective Dynaimcs of 'Small-World' Networks. *Nature* 393. p. 440.

[Wiley 2001] Wiley, B. 2001. Interoperability Through Gateways. In Oram, A., (ed) *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*, O'Reilly and Associates, Inc., Sebastopol, California.

[Yurcik 1996] Yurcik, W., Tan, Z. 1996. The Great (Fire)Wall of China: Internet Security and Information Policy Issues in the People's Republic of China. 24th Annual Telecommunications Policy Research Conference. Online at http://www.pitt.edu/~wjyst/ascii_tprc96.txt

Appendix A

PROTOCOL DEFINITION UNITS

This appendix outlines the Protocol Definition Units (PDUs) in detail. These are described here in terms of *packet types* and *function types.* There are five kinds of packet type, each of which reflect one of the communication modes described in section 3.4. Each packet type has a number of associated function types. The function types represent every kind of packet that may be sent by a node. By expressing the PDUs in this way, we clearly outline the commonality that exists between the many different kinds of packet and allow them to be understood in a more elegant and hierarchical way.

General Packet format:

NAME	SIZE(bits)	TYPE	DESCRIPTION
Maj.Version #	4	Unsigned int	Major version number
Min.Version #	4	Unsigned int	Minor Version number
UTID	24	Unsigned int	Unique Transaction ID
Packet Type	8	See packet types	Packet Type of this packet
Function Type	8	See function types	Function Type of this packet
Parameters' length	32	Unsigned int	Length in bits of all the parameters of the packet (the rest of the packet)
Packet parameters	variable	N/A	Parameters associated with Packet Type
Function Parameters	Variable	N/A	Parameters associated with Function Type

Notes:

- ❖ UTID is a unique identifier for each transaction over the network. For example, if a unicast is made in response to a broadcast, it will be given the same UTID, as they are considered part of the same transaction.
- ❖ Possible Packet Types and Function Types are outlined below.

Possible Packet Types:

NAME	CONTENTS	DESCRIPTION
broadcast	000	To be broadcast
unicast	001	To be unicast
Uni-directional	010	To be sent in a single direction
Coordinate-seeking	011	To be sent to a particular relative coordinate
Out-of-lattice	100	Sent directly

Possible Function Types:

NAME	PACKET TYPE	CODE		DESCRIPTION
patron request	broadcast	000	0000	Request by low-speed node to high-speed node
join notify	broadcast	000	0001	Sent by recently joined node to notify others of its address
file search	broadcast	001	0000	File search query
edge notify	unicast	000	0000	To notify that HTL on a route packet reached 0
request join	unicast	000	0001	Sent in direction of probable vacancy
file hit	unicast	001	0000	Sent by a node with file(s) matching a file search
download request	unicast	001	0001	Sent to download a file indicated by a file hit
rubber connection request	uni-directional	000	0000	Request a rubber connection with opposite node
diameter notify	uni-directional	000	0001	Notify nodes at a particular longitude/latitude of diameter of network
route	Coordinate seeking	000	0000	Route a packet to a particular relative coordinate
join	Coordinate seeking	000	0001	Instructs surrounding nodes to fill a vacancy with a particular new node

Select proxy	Coordinate seeking	000	0010	Instructs a node to select a proxy for a heavyweight data transfer
drop neighbour	Coordinate seeking	000	0011	Instructs surrounding nodes to disconnect from a particular neighbour
drop neighbour confirm	Coordinate seeking	000	0100	Confirms dropping of neighbour
request join	Out-of-lattice	000	0000	Request by node petitioning to join lattice
accept join	Out-of-lattice	000	0001	Accepts a new node to be a neighbour
refuse join	Out-of-lattice	000	0010	Refuses a new node to be a neighbour
request proxy	Out-of-lattice	000	0011	Requests a node to become a proxy for a data transfer
accept proxy	Out-of-lattice	000	0100	Node accepts request to be a proxy
refuse proxy	Out-of-lattice	000	0101	Node refuses request to be a proxy
accept patron	Out-of-lattice	000	0110	Sent by high bandwidth node accepting a patron request
accept rubber connection	Out-of-lattice	000	0111	Sent by node accepting a rubber connection request
proxy notify	Out-of-lattice	000	0110	Sent by a new proxy to client and server of data transfer
public key notify	Out-of-lattice	000	0111	Sends a public key to a node
shared key notify	Out-of-lattice	000	1000	Sends a shared key to a node
file data	Out-of-lattice	001	0000	Sends encrypted contents of a file to a node

Note:

- The contents of the CODE field are not unique, however they are unique with respect to each packet type. No two packet types contain functions with the same code.
- The CODE field has two parts. Interpreted as a unsigned integer, values in the range 0–15 indicate protocol control functions. Values in the range

16 – 127 indicate application functions, such as file sharing etc. This leaves plenty of room for future expansion of the protocol to other applications.

6.6 Format of packet parameters

Broadcast:

NAME	SIZE(bits)	TYPE	DESCRIPTION
Vacancy indicator	8	Unsigned int	Indicates vacancies in bandwidth ranges on path of packet
Avg. bandwidth	16	Unsigned int	Avg. throughput of originator of packet in Kps
Max. bandwidth	16	Unsigned int	Max. throughput of originator of packet in Kps
HTL	8	Unsigned int	Hops-to-live of packet

Unicast: none

Uni-directional: none

Coordinate-seeking:

NAME	SIZE(bits)	TYPE	DESCRIPTION
i	8	Signed int	Relative i coordinate of packet destination
j	8	Signed int	Relative j coordinate of packet destination
Rubber connection flag	8	boolean	Flag indicates whether packet

			has crossed a rubber connection
Orientation	8	N/A	0 indicates horizontal rubber connection. 1 indicates a vertical connection
HTL	8	Unsigned int	Hops-to-live of packet

Out-of-lattice: none

6.7 Format of function parameters

patron request (broadcast):

NAME	SIZE(bits)	TYPE	DESCRIPTION
Sender address	32	IP address	IP address of the originator
Port #	16	Unsigned int	Port # of originator

join notify (broadcast):

NAME	SIZE(bits)	TYPE	DESCRIPTION
Sender address	32	IP address	IP address of the originator
Port #	16	Unsigned int	Port # of originator

file search (broadcast):

NAME	SIZE(bits)	TYPE	DESCRIPTION
Min. bandwidth	16	Unsigned int	Minimum bandwidth for responding nodes (Kps)
Min. size	32	Unsigned int	Minimum size of files to be returned as hits

			(bytes)
Max. size	32	Unsigned int	Maximum size of files to be returned as hits (bytes)
Search string	variable	String	The search string

edge notify (unicast): none

request join (unicast):

NAME	SIZE(bits)	TYPE	DESCRIPTION
address	32	IP address	Address of new node
port	16	Unsigned int	Port # of new node
Max. bandwidth	16	Unsigned int	Max. bandwidth throughput of petitioning node

file hit (unicast):

NAME	SIZE(bits)	TYPE	DESCRIPTION
File ID	16	Unsigned int	Index of the file
Size	32	Unsigned int	Size of the file (bytes)
j	8	Signed int	Relative j coordinate of node to select a proxy
Bandwidth	16	Unsigned int	Max bandwidth of node
File name	variable	String	Name of the matched file

download request (unicast):

NAME	SIZE(bits)	TYPE	DESCRIPTION
File ID	24	Unsigned int	Index of the file
i	8	Signed int	Relative i coordinate of node to select a proxy
File range	32	Unsigned	Range of the file

		int	required (for resuming interrupted downloads)
Public key	1024	binary	Public key of the file requester

rubber connection request (uni-directional):

NAME	SIZE(bits)	TYPE	DESCRIPTION
Sender address	32	IP address	IP address of the originator
Port #	16	Unsigned int	Port # of originator
Hop counter	16	Unsigned int	Counts the number of hops that the packet takes

diameter notify (uni-directional):

NAME	SIZE(bits)	TYPE	DESCRIPTION
diameter	16	Unsigned int	Diameter of network (derived from hop counter, above)

route (coordinate-seeking):

NAME	SIZE(bits)	TYPE	DESCRIPTION
Original direction	2	N/A	Indicates the direction that the packet was traveling in before being routed
Message	variable	binary	Original message being routed

join (coordinate-seeking):

NAME	SIZE(bits)	TYPE	DESCRIPTION
address	32	IP address	Address of new node
port	16	Unsigned int	Port # of new node
neighbourhood	8	N/A	How to connect to the new node (North/South/East

			/West)

select proxy (coordinate-seeking):

NAME	SIZE(bits)	TYPE	DESCRIPTION
role	8	N/A	Indicates role of originator in transfer (client/server)
address	32	IP address	Address of originator
port	16	Unsigned int	Port number of originator
Max. bandwidth	16	Unsigned int	Max. throughput of originator of packet (Kps)

drop neighbour (coordinate-seeking):

NAME	SIZE(bits)	TYPE	DESCRIPTION
address	32	IP address	Address of neighbour to be dropped
port	16	Unsigned int	Port number of neighbour to be dropped
Neighbourship	8	N/A	Direction of sender from neighbour to be dropped (North/South/East /West)

drop neighbour confirm (coordinate-seeking):

NAME	SIZE(bits)	TYPE	DESCRIPTION
address	32	IP address	Address of neighbour to be dropped
port	16	Unsigned int	Port number of neighbour to be dropped
Random_code	8	binary	Random code generated to be used in confirmation

request join (out-of-lattice):

NAME	SIZE(bits)	TYPE	DESCRIPTION

Max. bandwidth	16	Unsigned int	Max. bandwidth throughput of node

accept join (out-of-lattice):

NAME	SIZE(bits)	TYPE	DESCRIPTION
Neighbourhood	8	N/A	How to connect to the new node (North/South/East /West)

request proxy (out-of-lattice):

NAME	SIZE(bits)	TYPE	DESCRIPTION
Server address	32	IP address	Address of node server
Server port	16	Unsigned int	Port number of server
Client address	32	IP address	Address of node server
Client port	16	Unsigned int	Port number of server
Throughput	16	Unsigned int	Estimated throughput of the transfer

accept proxy (out-of-lattice): none

refuse proxy (out-of-lattice): none

accept patron (out-of-lattice): none

accept rubber connection (out-of-lattice): none

proxy notify (out-of-lattice): none

public key notify (out-of-lattice):

NAME	SIZE(bits)	TYPE	DESCRIPTION
key	1024	binary	Public key

shared key notify (out-of-lattice):

NAME	SIZE(bits)	TYPE	DESCRIPTION
key	128	binary	Shared key

file data (out-of-lattice):

NAME	SIZE(bits)	TYPE	DESCRIPTION
Sequence #	32	Unsigned int	Sequence number of this block of the file (1 block = 65536 bytes)
Data	variable	binary	Contents of file

Appendix B

STATE TRANSITION DIAGRAM

This appendix provides a state transition diagram that describes the logic that should be performed when a packet is received from another node. This diagram forms the basis for all interaction between nodes and is of great importance to the system.