

# NAANOU:

A SCALABLE, MODERATED P2P NETWORK

Clint Heyer

clint@thestaticvoid.net

Pages: 68

Words: 19,454

INFORMATION ENVIRONMENTS HONOURS THESIS

# TABLE OF CONTENTS

<b>1</b>	<b>INTRODUCTION .....</b>	<b>6</b>
1.1	PEER-TO-PEER .....	6
1.2	COMMUNITIES .....	6
1.3	PROJECT OUTLINE .....	7
1.4	THESIS OUTLINE .....	7
<b>2</b>	<b>DESIGN .....</b>	<b>8</b>
2.1	INTRODUCTION .....	8
2.2	GOALS .....	8
2.3	RELATED WORK .....	9
2.4	CONCLUSION .....	10
<b>3</b>	<b>NETWORK SERVICES .....</b>	<b>11</b>
3.1	INTRODUCTION .....	11
3.2	LITERATURE REVIEW .....	11
3.2.1	<i>Generation 0: Hybrid P2P – Client-Server</i> .....	12
3.2.2	<i>Generation 1: Tree Structure</i> .....	13
3.2.3	<i>Generation 2: Unstructured Networks</i> .....	13
3.2.4	<i>Generation 2: Structured Networks</i> .....	15
3.2.5	<i>Other work</i> .....	16
3.3	CONCLUSION .....	17
<b>4</b>	<b>ANTISOCIAL BEHAVIOURS .....</b>	<b>18</b>
4.1	INTRODUCTION .....	18
4.2	BEHAVIOURS .....	18
4.2.1	<i>Free-riding</i> .....	18
4.2.2	<i>Behaviours Poll</i> .....	19
4.2.3	<i>Consequences of antisocial behaviour</i> .....	21
4.3	FREE-RIDING ANGER .....	21
4.4	RELATED IMPLEMENTATIONS .....	22
4.5	CONCLUSION .....	24
<b>5</b>	<b>ALTRUISTIC PUNISHMENT .....</b>	<b>26</b>
5.1	INTRODUCTION .....	26
5.2	GROUP COOPERATION .....	26
5.3	PUNISHMENT .....	27
5.4	FORMS .....	28

---

5.5	CONCLUSION .....	29
<b>6</b>	<b>IMPLEMENTATION.....</b>	<b>30</b>
6.1	INTRODUCTION .....	30
6.2	NETWORK .....	30
6.2.1	<i>Introduction</i> .....	30
6.2.2	<i>Transport</i> .....	31
6.2.3	<i>Peer Discovery</i> .....	31
6.2.4	<i>Chord Improvements</i> .....	33
6.3	KEYS.....	34
6.3.1	<i>Introduction</i> .....	34
6.3.2	<i>Replication</i> .....	34
6.3.3	<i>Node Ids</i> .....	35
6.3.4	<i>Publishing</i> .....	35
6.3.5	<i>Metadata</i> .....	36
6.3.6	<i>Searching</i> .....	36
6.4	ALTRUISTIC MODERATION.....	39
6.4.1	<i>Introduction</i> .....	39
6.4.2	<i>Security Implications</i> .....	42
6.4.3	<i>Conclusion</i> .....	46
6.5	RESOURCE EXCHANGING .....	46
6.5.1	<i>Requirements</i> .....	46
6.5.2	<i>Design</i> .....	47
6.6	USER INTERFACE.....	49
6.6.1	<i>Requirements</i> .....	49
6.6.2	<i>Design</i> .....	49
6.7	CONCLUSION .....	52
<b>7</b>	<b>CONCLUSION.....</b>	<b>53</b>
7.1	FUTURE WORK.....	53
7.1.1	<i>Summary</i> .....	53
7.1.2	<i>Network</i> .....	53
7.1.3	<i>Privacy &amp; Security</i> .....	54
7.2	SUMMARY.....	54
	<b>ACKNOWLEDGEMENTS .....</b>	<b>55</b>
	<b>REFERENCES .....</b>	<b>56</b>
	<b>APPENDIX A - POLL RESULTS.....</b>	<b>60</b>
	POLL 1 .....	60

---

---

POLL 2 .....	62
<b>APPENDIX B – DEFINITIONS.....</b>	<b>65</b>
<b>APPENDIX C – SOURCE CODE .....</b>	<b>67</b>
<b>APPENDIX D – PROJECT SCHEDULE.....</b>	<b>68</b>

# ABSTRACT

With the increasing population of peer-to-peer file sharing users these communities are faced with emerging problems. Inefficiencies present in the network structure hamper the scalability of the community. As the population exceeds the network's limits, the associated increase in bandwidth usage causes a slowing of the entire network and potentially leads to failure of network functions. Originally designed for only small-scale usage, many P2P systems are faced with two options: redesign or restrict network growth, thus limiting the full potential of a peer-to-peer network. With the introduction of a new class of peer-to-peer network - distributed hashtables - high performance and efficient scalability is possible, far beyond the capabilities of existing systems. This thesis explores applying a distributed hashtable system to the P2P file sharing domain, introducing a user-centred designed client that offers features available in other file sharing clients but providing a significantly higher degree of scalability and performance. The means by which traditional P2P file sharing features could be mapped to the novel distributed hashtable-based system had not been explored at the commencement of this work and many important issues in the area are yet to be explored. The thesis presents a variety of designs and an implementation for metadata-based key publishing and searching, and distributed file exchanging.

Also central to this work is how online communities, and in particular file sharing communities can achieve a level of self governance. Antisocial behaviours are discussed, as well as the results of two questionnaires conducted on the subject. These results demonstrated the level of antisocial behaviours present in file sharing communities - and how participants react to these behaviours. Altruistic punishment is suggested as a novel method for reducing antisocial behaviour within a file sharing community. Research on the positive effects of altruistic punishment on group cooperation levels is discussed, along with how this research applies to the P2P domain. A method for implementing decentralised moderation in a distributed hashtable system is introduced and the security implications of the method are explored.

# 1 INTRODUCTION

## 1.1 PEER-TO-PEER

Peer-to-peer (P2P, for a definition see Appendix B) file sharing communities are growing in prominence, increasing in both well as wider community awareness (see [1] for figures and implications of this growth). People generally join the communities in order to find or share various types of resources, typically music and video with other potentially likeminded people. P2P architecture in many ways lives up to the lofty goal of web publishing – anyone can put material ‘out there’, available to the masses. P2P lowers the entry bar; all a person has to do is download a client and pick which files to share, and the client takes care of all the networking and administrative functions without the user’s involvement. While it is relatively simple to put up a web page these days, there is very clear hierarchy of administration and authority governing what material can be put up, and by whom. This may vary in severity from a free web-host’s terms of service, to government censorship laws. P2P however, is a decentralised anarchic publishing network that when well designed, cannot be controlled by any one party. By having the load of exchanging and indexing files shared by everyone on the network, P2P networks are able to grow to a much larger size than traditional client-server systems and with less cost. P2P networks have proved immensely popular in recent times, as shown in [2].

## 1.2 COMMUNITIES

With the increasingly larger population of users of online communities such as chat channels, online gaming, forums and P2P networks, the small ‘frontier-town’ spirit of the community is quickly being displaced by that of a large bustling metropolis. Undesirable social behaviour rises in prominence and affects the community in both performance and morale terms (discussed in 4.2.2). Behaviours such as ‘leeching’ (downloading a proportionally large amount of resources compared to what is being shared by the leecher), or ‘free riding’; abusive chat messages; spamming; hijacking and other, selfish behaviours are evident. While other online communities have various mechanisms for dealing with these types of behaviours, they are notably absent from most P2P networks.

This could be attributed to the difficulty with implementing a decentralised solution that is reasonably secure.

## 1.3 PROJECT OUTLINE

This thesis introduces the design and implementation of a new P2P system that attempts to address antisocial behaviours in online communities. An implementation was designed based on extensive research into the various available network protocols, the literature exploring antisocial behaviour, and the literature describing the means by which the effects of antisocial behaviour can be reduced. The implementation - 'Naanou' - is a fully-functional peer-to-peer file sharing system and serves as proof of concept of many ideas and designs put forth in this thesis. Naanou has been successfully tested publicly and has potential as a test-bed for further development in the area of distributed hashtable decentralised systems, decentralised moderation and privacy within a peer-to-peer network.

## 1.4 THESIS OUTLINE

Chapter 2 looks at the overall goals of the project, followed by chapter 3 which surveys existing peer-to-peer network protocol research. Drawing on survey data collected as part of the thesis, chapter 4 examines online antisocial behaviours: what they are, how widespread they are, and what in particular bothers P2P users. In chapter 5, the concept of altruistic punishment is introduced, along with background information on group cooperation and discussion of what form punishment should take in a P2P context. Chapter 6 documents the implementation details and design decisions taken for the prototype implementation, *Naanou*. Much of this work is informed by chapters 2-5, and covers how a structured distributed network was implemented in a P2P domain, as well as how traditional P2P features were mapped onto said network. The design for moderation is presented, as well as the resource exchanger, and finally the user interface is presented via a series of screen shots. Chapter 8 concludes the thesis, with the appendices covering the full questionnaire results, term definitions, source code and project schedule.

## 2 DESIGN

### 2.1 INTRODUCTION

A peer-to-peer file sharing network needs to have several qualities to be attractive to a potential user given that the field features a large number of disparate and incompatible technologies. In a study conducted for this thesis, subjects were asked to rate how much they valued seven qualities of a network. High content quality, high content quantity and high-speed content download rated as the most important features (in that order), with large network size ranking fourth. A high quality P2P system needs be built upon an efficient scalable distributed protocol. Otherwise, the network's growth would be limited, affecting the number of users it attracts (and can support) which in turn would hamper other aspects of the network. This type of problem has been evidenced with the Gnutella[3] network, which was originally designed for only small-scale growth, but with the demise of Napster[4] quickly became the only choice for thousands of users. With this huge influx of new users, Gnutella's performance degraded significantly and became largely unusable.

Another, more obvious, attribute required for a P2P file sharing network is the mechanism for sharing and exchanging files. File availability, and the speed at which files can be exchanged between peers needs to be maximised. Files also need to be able to be located easily via a search mechanism. Finally, all this functionality needs to be packaged into a usable interface.

### 2.2 GOALS

The goals for Naanou's implementation are to provide a feasible alternative to current P2P systems by offering functionality equal to or better than alternatives and to offer moderation mechanisms unavailable elsewhere. In the future, when this implementation is available, testing could occur to judge the success or failure of these moderation mechanisms.

In order to inform the specific design goals for the project, a short poll was conducted in an effort to identify which aspects of a P2P network users most value. The participants were asked to rank on a 1-10 scale (10 being the highest) the value they

placed on a list of seven attributes; or, to list and rate any additional relevant attributes not included in the poll. The results of the 140 anonymous online respondents<sup>1</sup>, in descending order of the average score were:

1. High content quality (8.8)
2. High content quantity (8.4)
3. High-speed content download (8.2)
4. Large network size ( 7.4)
5. Anonymity (6.9)
6. Presence of authority figures who moderate environment (3.5)
7. Like minded, interesting people to chat with (3.4)

These results helped guide the design of Naanou, with particular care taken to include the first four points. Obviously, it is hard to address some of these points explicitly; for example, one cannot design a network that instantly is very large, rather one must design a system that is *capable* of growing to support a large number of users.

After weighing poll results against ease of implementation, a set of design goals were established:

1. An efficient, scalable network
2. Effective moderation mechanisms
3. Efficient download system that maximises file availability and throughput.
4. Ease of use

For additional privacy, Naanou does not expose users' IP addresses<sup>2</sup> and by default only shares media files, and not sensitive system or user files, preventing some of the privacy concerns discussed by Good and Krekelberg[5].

## 2.3 RELATED WORK

Only two P2P clients using a distributed hashtable system are currently available; Overnet[6] and VARVAR[7]. The difference between the two clients is unlikely to be great, as both are based on the Kademlia<sup>3</sup> protocol, and the lead Overnet programmer is also a programmer for VARVAR. As there is very little published information on how many aspects of these systems work, it is difficult to compare them with Naanou. Other

---

<sup>1</sup> Full results presented in Appendix A.1

<sup>2</sup> Although such information can still be sought at the operating system level

<sup>3</sup> Discussed in 3.2.4

than offering a more scalable network substrate, Overnet and VARVAR do not provide any new features beyond those found in existing P2P clients. Moreover, neither attempt to employ moderation.

## 2.4 CONCLUSION

A peer-to-peer file sharing network has obvious prerequisite features, such as the ability to share files, and to have a peer-to-peer architecture. Users of existing P2P networks were polled to discover what features they most valued in order to inform the design of Naanou. The results were then used to establish a set of design goals for the system. The goals for the system are: for the network to be efficient and scalable; have effective moderation mechanisms; an efficient download system; and be easy to use.

# 3 NETWORK SERVICES

## 3.1 INTRODUCTION

A protocol is required to organise disparate nodes into a cohesive network. Once organised into a network, this protocol is responsible for maintaining the network, routing messages, locating data and storing data. This protocol to a large part determines how decentralised and distributed the system ends up becoming, and is the most difficult component of a P2P program to implement. In this thesis, the system by which nodes on a P2P network store and lookup keys will be referred to as the *lookup service*, and the system which carries and routes messages for the lookup service will be referred to as the *network service*. In a hybrid client server-P2P network like Napster[4], the lookup service's functionality resided primarily with the server and clients only query the server when the user performs a search. The network service in such systems is a traditional client-server model. Distributed hashtable (DHT) systems such as Chord[8] and Pastry[9] couple the lookup service and network service together. Looking up a key for an object is exactly the same as locating a specific node on the network. Other systems such as Freenet[10] take this idea further, by combining lookup, network and storage services together.

## 3.2 LITERATURE REVIEW

For Naanou, an efficiently scalable P2P substrate was required to provide services in a potentially large network. There are many different P2P protocols in use, or undergoing research which can be broken up into three major generations:

0. Hybrid P2P – client-server, highly limited scalability
1. Tree structure, limited scalability
2. Highly scalable, unstructured and DHT networks

These protocols can be considered *overlay* networks, in that they are implemented above the actual TCP/IP network layer, and form structures without necessarily any regard to the physical network. Each has various strengths and weaknesses which will be examined:

### **3.2.1 Generation 0: Hybrid P2P – Client-Server**

This is one of the earliest implementations of a ‘peer-to-peer’ network, popularised by Napster in August 1999. Clients connect to a central, well-known server and send a file list of resources the client has shared. The server keeps track of which clients are online, and maintains a central file index. Searches are sent to the server, with results containing the direct addresses of other peers that have matching files. The client can then initiate a direct connection to that peer, and retrieve the file. It was this latter feature that gave Napster its ‘peer-to-peer’ moniker, and in a time where FTP and the World Wide Web were the primary methods for exchanging files it certainly did appear to be decentralised. When the popularity of Napster increased, more servers were added to handle the load, in a similar fashion to popular websites, but the reliance on these central Napster Inc., controlled servers was always present. Having these centralised points of failure meant that Napster Inc., could control the network at all times, which was Napster’s undoing not only in performance terms, but when legal proceedings were undertaken against it by entertainment industry groups[4].

DirectConnect[11] is an example of a system that has taken the Napster model and scaled it down to make it workable. Instead of attempting to have hundreds of thousands of users connect to a few well-known servers, DirectConnect releases the server software as well as the client software so users can run their own ‘hubs’. The architecture remains the same, but now the ratio of clients to servers is more balanced, and reduces the profile of individual servers. Hubs are often formed around an interest area or geographic region, which improves the speed and quality of resources available. While multiple hubs can be linked together to form a larger network, most remain solo which means the entire DirectConnect community is spread into separate walled networks resulting in small, disparate populations, and less quantity of resources. Locating and joining a hub is accomplished by downloading a hub list that is offered by several websites, or being invited to a private hub. DirectConnect features an integrated chat system for hub members that can increase the sense of community in a small hub. As the creators of DirectConnect do not themselves run any services, the risk of legal liability or responsibility is lessened.

The benefits of hybrid P2P are its performance (when there are adequate server-side resources available), and simple implementation. Because the servers are usually selected for having large computational or network resources, they can often outperform a number of more lowly computers doing a similar task. Also, by having all clients

connect to a set of known servers, no peer discovery mechanisms are required and content publishing/searching is greatly simplified. The side effects of this simple architecture are severely limited growth and single points of failure. Growth is limited as servers become saturated with connections or requests, and can only be fixed by increasing the available servers (or upgrading their capacity). If these servers become unavailable, there is no way for a client to reach other clients, and the network is effectively broken.

### 3.2.2 Generation 1: Tree Structure

FastTrack[12] is an evolved form of the hybrid structure. Instead of having only two classes of entities on the network - server and client, FastTrack's network automatically organises nodes into a tree structure. Nodes with more computing power or network bandwidth available become 'super nodes' which take on more of server's role in a hybrid network. Clients join under a super node hierarchy, and send requests to their parent node, with file transfers handled in a similar fashion to Napster. The benefit of this model is that the ratio of clients to controller-nodes is smaller; reducing the impact should one of these super nodes fail. Due to the tree structure, the more client nodes connected, the more elevated-status super nodes are required to manage them. These super nodes in turn must have nodes that manage them, and so on. As the network grows, the amount of overhead for managing the network grows at a much greater rate, limiting the network's scalability. However, the scalability potential for a tree network is still much higher than a hybrid network.

### 3.2.3 Generation 2: Unstructured Networks

The Gnutella Protocol[13] while originally not designed for very large growth, has proved popular with P2P users, and with developers working to modify the protocol to improve scalability[3, 14] Gnutella should remain popular for some time. Messages are broadcast to a set of peers that a client knows about, who then rebroadcast the message to peers *they* know about and so on until a message is answered successfully or has expired. Gnutella does this in conjunction with primitive routing heuristics for searches and maintaining the network. Data is not published or advertised as in many other P2P systems, rather queries are forwarded around the network until they find a node that can fulfil the request. Using a reducing time-to-live count, requests eventually get dropped out of the network to prevent infinite-length searches. There has been analysis of the original Gnutella protocol[15] to demonstrate its limited scalability, due to its reliance on

message broadcasting – as the number of peers grows, the traffic required for messaging increases disproportionately. Newer revisions of the Gnutella protocol introduce some structure into the GnutellaNet, with FastTrack-like ‘super nodes’ reducing the amount of broadcast traffic required by acting as proxies for several clients.

Freenet[10] is a system that seeks to not only offer efficient scalability, but also a high-degree of privacy, security and redundancy. Being an unstructured network, Freenet nodes only ever have a small localised view of the network. Requests are forwarded by nodes to remote nodes that are judged most likely to fulfil the request. A hops-to-live limit stops requests being forwarded around the network forever, and there are mechanisms to stop loops forming in the forwarding chain. As requests branch out through the network, their status back-propagates down request path to varying levels. This back-propagation serves to update routing tables by informing clients about distant clients they may not have known about before – over time improving the efficiency of forwarding. Instead of file transfers being out-of-bound<sup>1</sup> of the lookup service as the case for most networks, transfers of data in Freenet follow the same procedure as anything else on the Freenet network. As data is fetched from a remote node, it is cached at all intermediate nodes along the path to the requestor. This behaviour increases the availability of the file in case the original source becomes unavailable, and caches data at points where any subsequent requests for that data are likely to travel. When paths are determined, lexicographical distance is used to route requests to nodes that answered a similar request. This has the beneficial side effect of nodes becoming specialised in key space areas, in both the node’s routing knowledge of nodes that have close data, as well as storage of data. Thus a node will have requests routed to it that are ‘similar’ to requests it has previously serviced. As requests cache data along their path, the node will slowly build up its cache of similar data, and can answer requests directly. As requests are made and files transferred, the nodes can arbitrarily change the header values to disguise who actually made the request or stored the data. In summary, Freenet offers many attractive, unique features such as load balancing, anonymity, data storage replication and security. The negative aspects of Freenet are a complex implementation and relatively high network usage, due to the transferring of all data and broadcast messaging.

---

<sup>1</sup> Meaning, data transfers are not carried by the lookup service, and is usually accomplished via another connection such as HTTP.

In conclusion, unstructured networks are characterised by a highly fault-tolerant nature at the expense of slower, more bandwidth intensive, and often non-determinable requests.

### 3.2.4 Generation 2: Structured Networks

Chord is a DHT that forms an ordered logical ring structure. It offers the capability to perform key lookups in  $O(\log N)$  hops, with a single node only having to maintain information about  $O(\log N)$  other nodes. Nodes have *finger tables* in which pointers are maintained to exponentially further away nodes. Thus, a node knows about more nodes closest to it, and less distant nodes, yet can still locate a distant node in logarithmic time with the help of the fingers. This node-hopping via fingers is illustrated below in Figure 1.

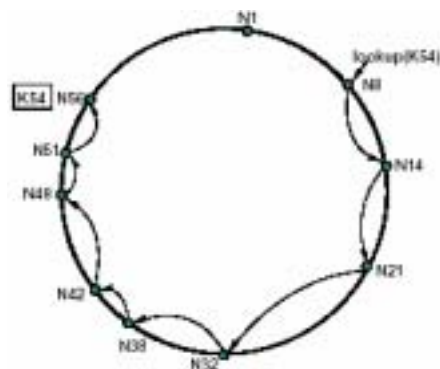


Figure 1: Chord lookup via a finger table, figure from [8]. A lookup commencing from N8 rapidly converges to the correct node, N56. In-between nodes are not shown.

Nodes and data are keyed via a consistent, uniform hashing algorithm (such as SHA-1[16]) to produce a ‘position’ for the item on the Chord ring. Keys are stored at the node responsible for that part of the keyspace. Queries for keys are accomplished by hashing the query, locating the node responsible for the key and then sending a request to that node. Chord has been shown to be resilient to a large number of node failures, and can handle a large number of simultaneous node joins or parts, which is required in a haphazard P2P network. The Chord protocol itself is simple in terms of implementation and use in higher application levels, and allows more complex applications to run on top of it, such as a distributed file system[17]. Chord offers guaranteed performance characteristics, and all queries have a determinable state (as opposed to Gnutella or Freenet where it is often unclear if a request failed because the sought data did not exist, or just was not found).

Kademlia[18] is also a distributed hashtable system, but has more flexible routing than Chord. Chord's routing tables are highly rigid - finger tables must always contain the numerically closest node for each finger. Kademlia has a system similar to Chord's finger tables, but allows for finger selection on other metrics such as network latency. Kademlia maintains ' $k$ -buckets', a collection of up to  $k$  nodes sorted by time last seen for address prefixes. As remote nodes contact the local node, their address is inserted into the appropriate  $k$ -bucket, based on their prefix. As  $k$ -buckets fill, Kademlia discards nodes that are no longer contactable, and favours older nodes over newly discovered nodes. This attribute is based on work done in Saroiu *et. al.* [19], in which it was shown that the longer a node has been online, the more likely it is to continue to be online. Having  $k$  possible nodes to route queries to gives Kademlia greater flexibility for routing selections, whereas a basic Chord implementation only has one routing option for finger locations.

Global Information Sharing Protocol (GISP) [20] is a protocol that leverages much of DHT design from Chord and others but primarily presents an implementation, rather than a theoretical foundation. Closely related to Chord is Pastry[9], Tapestry[21], CAN[22] and the Plaxton protocol[23]. Chord has similar performance characteristics to these, except for a weaker worst-case performance guarantee. Tapestry, Plaxton and Pastry benefit from locality awareness, as they take 'network distance' (for example, hop counts) into account when routing messages. Tapestry and Pastry also route on address prefixes rather than address numerical distance as Chord does, but this has little bearing on performance. A structured network design lends itself to key and data caching, as queries from different clients often converge on the same route. In the structured networks discussed, there was no attempt to address the issue of privacy, nor was there any attempt to provide a high degree of security.

### 3.2.5 Other work

To improve the latency of both unstructured and structured networks a technique of *binning* topologically nearby nodes together can be used[24]. This binning can be accomplished in a scalable, efficient manner and can complement a P2P network protocol. Nodes are binned based on their network latency to a small set of well-known landmark nodes (research indicates that 8 to 12 landmarks would be sufficient for the current size of the Internet[24]) and can be used for selection of finger nodes in a protocol such as Chord, or in a download system when determining which node to download from when multiple sources are available.

### 3.3 CONCLUSION

Peer-to-peer networks can be split into three generations; generation 0 being a hybrid client-server/P2P network (such as Napster) which is characterised by high performance but very small scalability; generation 1 which has a rough tree structure (such as FastTrack), which evolves the Napster design and improves scalability and finally generation 2 structured (such as Chord) and unstructured (Freenet) networks which are characterised with high performance and high scalability. Chord was selected as the network services protocol for Naanou based on its high performance, straightforward implementation and clear, concise protocol specification. While Chord lacks locality awareness which could potentially affect performance, the work discussed in 3.2.5 could be integrated into the implementation. Chord does not provide any privacy protection, but such functionality could also be incorporated.

# 4 ANTISOCIAL BEHAVIOURS

## 4.1 INTRODUCTION

Given the ongoing popularity and growth of P2P networks, it could be argued that is unlikely that there are any cooperation-related problems that need to be addressed. As P2P populations grow however, antisocial behaviours performed within the community also grow, and when left unchecked can grow faster than the population<sup>1</sup>. In a file sharing community, if the exchanging of files is seen as the primary objective, cooperating can be defined as sharing files, and not cooperating defined as not sharing. While this could be seen as the primary form of antisocial behaviours (and indeed, has been demonstrated to be widespread in a large scale P2P network[19, 25]), this is not necessarily what users are concerned with. This chapter examines survey and passively<sup>2</sup> gathered data on antisocial behaviours online, and in P2P networks and discusses existing implementations designed to address these problems.

## 4.2 BEHAVIOURS

### 4.2.1 Free-riding

Empirical evidence gathered by two studies show that not sharing files, but downloading other peer's files (commonly called "free-riding", "free-loading" or "leeching") is widespread[19, 25]. Saroiu *et. al.* [19] report that 25% of Gnutella peers do not share any files, with 75% sharing 100 files or less. Additionally, the 7% of users that share more than 1000 files share more than the remaining 93% of users combined. Adar and Huberman [25] found even more extreme results - "70% of Gnutella users share no files, and nearly 50% of all responses are returned by the top 1% of sharing hosts". There is a large difference in these results, which can be attributed to the different methodologies employed during testing. Adar and Huberman gathered results for 24 hours, and found 33,335 unique peers whereas Saroiu *et. al.* collected data for eight days and found 1,180,205 unique peers. Even if the results of the Adar and Huberman study are

---

<sup>1</sup> See 5.2 for discussion on this point

<sup>2</sup> Data gathered passively while users used the system without users' knowledge.

disregarded, Saroiu *et. al.* presents a compelling case for the level of free-riding present in a network such as Gnutella. Adar and Huberman also performed a similar study in the now defunct Napster network, citing “40-60% of the peers share only 5-20% of the shared files” [19], which suggests free-riding is not a Gnutella-only phenomena. Levels of free-riding in other P2P networks, or types of online file sharing mediums, such as those which use IRC or FTP (for definitions on both, see Appendix B) is not readily available, but would make for a useful comparison.

The specific reasons for users choosing not to share files are unclear. One could assume it is done for selfish reasons; the less is shared the less other users will be downloading, thus conserving bandwidth and system resources for the selfish user. When peers download from the selfish user, it consumes CPU time, disk I/O bandwidth and outgoing network bandwidth which the selfish user could be otherwise using for themselves. Other plausible explanations are that the user does not know how to share anything; the user does not have anything to share or the user is unable to share due to technical reasons. As most P2P clients automatically share anything the user has downloaded, the first two points are weakened, and with modern clients featuring increasingly sophisticated firewall penetration capability, the third is also less likely. Thus, selfish behaviour seems the most likely explanation for the lack of file sharing found on many P2P networks.

#### **4.2.2 Behaviours Poll**

While free-riding is an easily measurable antisocial behaviour, there are other antisocial behaviours that can disrupt a user’s experience. In a study conducted for this thesis, 140 P2P users of varying networks were asked to rank seven behaviours on a 1-10 scale (10 being the most disliked). The results were (in descending order of rank averages, full results are in Appendix A.1):

1. Misrepresenting content i.e. naming a file to indicate it is something it is not (9)
2. Prematurely cancelled downloads (7.7)
3. Obnoxious chat messages (7)
4. No content shared (5.7)
5. Peers staying online long enough for only their own downloads to finish (5)
6. Peers that download several files at once from the one source (4.5)
7. Peers that download proportionally much more than they have shared (4.2)

These results indicate that the level of sharing other peers does not particularly bother the majority of P2P users sampled. It is worth noting that the top three ranked antisocial behaviours are those that have a more directly observable negative affect on the user, whereas the other antisocial behaviours are more likely to happen behind the scenes and have a neutral affect on the user. For example, seeing a file shared, downloading it and realising after downloading it was not what was advertised has an immediate negative affect on the user – they wasted time downloading something that was worthless to them – time that could have been better spent downloading the real file. When a user does not share any content, it is unlikely to have any direct negative consequence for others. The amount of resources per user in the network is lowered by this user's action, but as long as other users find what they are seeking it is unlikely to affect them. However, a user, Alice, on seeing Bob downloading several files from her at once may query what files Bob has shared (perhaps in an attempt to locate new files that someone with a similar taste has). If Bob has no files shared, Alice might feel slighted by Bob's behaviour.

Antisocial behaviour online is also found in other mediums such as online chat and e-mail. A series of studies have been done on children and teenagers' exposure to online antisocial behaviours such as harassment. A survey of children in the United Kingdom reports that 7% have been harassed in chat-rooms, and 4% via e-mail[26]. In a study of online teenage girls conducted by the Girl Scouts Research Institute, 30% of participants indicated they had been sexually harassed in a chat session[27]. A survey by Ipsos-Reid of 10,000 young (12-24) Internet users found that 70% use chat rooms, with 25% of female and 12% of male users reporting "negative experiences" whilst in a chat room[28].

In a different poll of 20 P2P users (full results are in Appendix A.2) this author has conducted, all respondents indicated they had encountered antisocial behaviour online, while 20% of respondents indicated they had never experienced antisocial behaviour within a P2P community. Of those that reported online antisocial behaviour, 45% stated that it didn't bother them. Interestingly, of those that experienced antisocial behaviour in a P2P community, only 6% stated it didn't bother them, with 37% stating it bothered them (a further 25% saying it bothered them a lot). So while antisocial behaviour appears to be less widespread in P2P, it seems to bother people more than antisocial behaviour in other online environments. When asked if they would leave a network because of the level of antisocial behaviour, 45% of respondents said yes, 40% said no, and 15% said they already had.

### 4.2.3 Consequences of antisocial behaviour

Given that many P2P networks contain thousands of users it could be argued that the large number of free-riders and spammers is unimportant. Similarly, given that many antisocial behaviours do not have a direct effect on individual users it may be that anti-social behaviour may not be an issue of concern to many users.

Free-riding, the act of downloading content from peers while you yourself have nothing shared can have negative impacts on the performance of the network. When the ratio of free-riders to the actively sharing peers is unbalanced, traffic of the network is disproportionately high for the sharers. In a network such as Gnutella, the sharing nodes have to answer more requests for searches - and actually send the file to requestors. As clients usually have a finite number of download 'slots' available, a sharer can quickly become tied up serving requests. It is the cooperators in the network who are disadvantaged by uncooperative free-riders. In a system designed to facilitate *peer-to-peer* file exchanging, having few sharers forces the network to behave like a client-server system.

The affects of other behaviours such as bullying via messages are akin to offline bullying behaviours, and there is a large body of work on the detrimental affects this can have on people, in particular children or teenagers (see [29-32], for example).

## 4.3 FREE-RIDING ANGER

In a study designed to discover how people react to free-riding, 140 peer-to-peer users were asked a series of four questions in sequential order, displayed one at a time. The participants did not know the purpose of the questionnaire, other than to gather general feedback about peer-to-peer communities. The first two questions asked the participant to rate his or her anger level (1-10 scale, 10 being the highest) toward a peer *C*, given a hypothetical sharing situation in a network consisting of only four members. The sharing situations presented to the participant are shown in Table 1.

<u>Person</u>	<u>Shared Content (MB)</u>	
	Question 1	Question 2
<i>You</i>	160	50

A	200	30
B	130	70
C	20	25
<hr/>		
<i>Avg. Result:</i>	<i>3</i>	<i>2</i>

*Table 1: Hypothetical sharing situations for questions one and two*

In question 1, a free-rider situation was presented, and participants responded with a very low level of anger toward free-riders. Question 2, the anger level dropped, indicating there is a link between the anger level and how much C deviated from the group norm. In the next two questions, the participant was asked the inverse – what anger level would *you* expect to receive from another peer in the same four-member network given a new set hypothetical sharing situations. These situations are shown in Table 2.

<u>Person</u>	<u>Shared Content (MB)</u>	
	<b>Question 3</b>	<b>Question 4</b>
You	20	200
A	140	300
B	160	500
C	180	700
<hr/>		
<i>Avg. Result:</i>	<i>5</i>	<i>3</i>

*Table 2: Hypothetical sharing situations for questions three and four*

With the ratios between sharing amounts almost identical as the first two questions, questions three and four indicate people expect much greater anger than they themselves are prepared to give. The differences in anger level between the two presented scenarios are also more marked, with a difference of two levels, instead of one (see 11.1 for full results in this study).

## 4.4 RELATED IMPLEMENTATIONS

There are many existing approaches to dealing with various types of online antisocial behaviour. To deal with unwanted messages in instant messaging systems, clients often have ‘ignore all messages from this user’, or ‘ignore all messages from those not on my contact list’ functions. With centralised chat systems, such as IRC (see Appendix A for a definition) or AOL® Instant Messenger[33], there is often recourse for users affected by

antisocial behaviours. In IRC, users can get kicked or banned from a channel (leaving them free to join other channels on the network) if the channel operator does not approve of their behaviour, or the matter can be escalated to server operators that can ban the user from the entire network. AOL® Instant Messenger allows users to send public or anonymous warnings to another user. Warnings are maintained on the server side, and persist from session to session, decaying over time. In each chat window, the public warning level of users is displayed, so a user can gauge the reputation of the person they are talking to. As the warnings accumulate, a user may be blocked from sending any messages, or even connecting[34]. For e-mail, there are sophisticated spam blockers available, and projects such as the Realtime Blackhole List[35] which can effectively block the source of bulk e-mail.

In centralised file sharing communities which have an inherent control hierarchy such as DirectConnect, IRC-type kick and ban commands can be issued against users. FTP and IRC-based client-server file sharing communities often use an accounts based system, where a user must have a valid account on the server before they can participate. To prevent free-riding they also often have a size or quantity-based ratio system for uploads and downloads. For example, a user must upload 1MB for every 10MB they wish to download. These types of mechanisms are easily implemented in a centralised system, and ensure that the users that download are also responsible for uploading new content onto the server for others. By making the upload/download relationship proportional, users on low bandwidth connections are not disadvantaged; and as such this server policy is generally accepted to be a fair practise within these communities. As these centralised groups are often very small (10-200 users) when compared to hundreds of thousands of users in Gnutella, the author has often seen them develop into niche, elite groups with private, invite-only membership. Boutique networks such as these have the intrinsic benefits of a close-knit community: everyone knows one another; content quality is higher and moderation requirements lower.

Existing peer-to-peer file sharing networks offer few new techniques to the user to counteract antisocial behaviours. Simple mechanisms such as ignoring messages or cancelling a download are available, but have limited use when the population of the network is so large – cancelling one free-riders' download only opens a slot for the next. MojoNation[36] (a now defunct system) sought to apply a micro-payment type mechanism to P2P. When a user sends a file (or part thereof) to someone else, they accrue credit – 'mojo' – which can be stored and later redeemed for priority downloads

from other peers. Payments are only required when a particular resource is overburdened. With MojoNation users who have been actively participating on the network get priority over those just recently joined, or users who are not exchanging files. This mechanism can act as an automatic global load balancing enforcer, and could help reduce the impact of free riders. Shirky[37] counters that using CPU and bandwidth resources as a currency in a P2P system is inherently flawed. Shirky notes that the “Tragedy of the Commons”<sup>1</sup> effect, which MojoNation has attempted to deal with does not apply to the digital world. If one copies a file from one user to another, nothing is lost. The only resources being consumed are those that are “infinite over time” - bandwidth can be saturated for five minutes for example, but after the transfer is finished it is available again, it cannot be saved up. If a user leaves their P2P client running overnight while they sleep, do they lose anything by allowing one hundred users to download from them? Even if their computer was being saturated with requests, using 100% CPU and 100% network bandwidth, as long as it is available when the user needed it in the morning, does it matter? Shirky argues that when a resource is so cheap from a user point of view it is not worth their extra time or cognitive load in monitoring usage of the resource (or how they are using other’s resources). While Shirky’s rebuttal holds true for unmetered network connections, when a user is connected via a capped (in both speed and/or download amount) Internet connection<sup>2</sup> these previously infinite resources start becoming valuable, and worth controlling.

## 4.5 CONCLUSION

While antisocial behaviours are prevalent in P2P communities, there are few implementations that adequately promote prosocial reform. Traditional methods (such as ignoring one particular user’s messages) transposed into a P2P domain does not scale to a system the size of a popular P2P network, with hundreds of thousands of users. Client-server architectures such as AOL® Instant Messenger allow users to warn others about a particular user. This method scales more successfully, as warnings are aggregated on the server-side. However, in a true decentralised P2P network, this type of system is difficult

---

<sup>1</sup> Example from [30]: If there is a shared pasture owned by a group of sheep farmers, farmers acting in the groups’ interest would try to keep herd sizes low, so not to overgraze field. It is however in each farmers’ self interest to maximise his herd to take as much as possible of the free resource.

<sup>2</sup> As is usually the case in Australia

to implement effectively. MojoNation attempted to effectively ‘tax’ transactions on the network, but failed as the resources being taxed were viewed by users to be so cheap, it wasn’t worth their effort to monitor them[37].

# 5 ALTRUISTIC PUNISHMENT

## 5.1 INTRODUCTION

There is a large body of research from the fields in economics and biology on the dynamics of group cooperation. How group cooperation comes about in groups of strangers, or groups where it is in each group member's interest to act selfishly is not something that can be easily explained by evolutionary models. A powerful motivator for increased cooperation has been shown to be punishment (reward, on the other hand has been shown to not work as well[38]). The chapter examines group cooperation, punishment and discusses the form punishment should take.

## 5.2 GROUP COOPERATION

Getting a group to cooperate together is never an easy task and when it is in each group member's interest to act selfishly, it is can be even harder. If the primary goal on becoming part of a file sharing community is to exchange files, preventing other people from accessing your files can be seen as being uncooperative or acting selfishly. Similarly, other forms of antisocial behaviour that have the effect of intimidating, annoying or driving away users could also be viewed as being uncooperative.

Group cooperation norms develop over time via pay-off biased (tendency to copy the most successful behaviours) and conformist (tendency to copy the most frequent behaviour) transmission[39]. When it is easier to not cooperate - for example, by not sharing files - it is the tendency of people to do so, and via norm transmission this behaviour spreads. Thus, with a diminishing cooperation level, mechanisms need to be introduced to stop the decline, or even turn negative cooperation growth into positive. Economists use game theory, and in particular Public Goods games to explore theories on group cooperation and selfishness [38, 40-42]. In one Public Goods game, four players are given a sum of money (say, \$20), which they then independently invest into a common pool. No players know how much anyone else has contributed. After each player has invested, the pool is doubled, and the money divided evenly amongst the four members. If each player invests their entire balance, cooperating fully, they all double their money. For every player that does not cooperate fully, the rate of return is much

lower. Indeed, if a player was entirely selfish, they could choose to not invest anything, as they still receive the division of the pool that others have invested into. Traditional economic models of peoples' behaviour would indicate that all players would behave exclusively selfishly, by investing nothing and thus getting no reward. Research shows however that people do invest quite heavily, but with each game play iteration (with same group composition), cooperation drops[42]. Fehr and Schmidt [43] surveyed 12 different Public Goods experiments without punishment and reported that in the first game rounds, the average investment of players was 40-60% of their total amount. By the time the final game round is played however, 73% of players were contributing nothing, or close to nothing. This cannot be explained by players learning that selfish behaviour is optimum as when players were reorganised into new groups, experimenters found that the same pattern re-emerged.

## 5.3 PUNISHMENT

In society, we have authoritative figures and institutions to keep 'uncooperative' behaviours in check. By way of group cooperation norm transmission, an entire populace can be kept in a relatively law-abiding state without everyone required to have direct contact with law keepers - the mere threat of punishment is enough to keep most of the population cooperative[39]. This point is further reinforced by the results discussed in 4.3. Users anticipate a higher degree of anger from other users the more they deviate from the group norm. Thus, if punishment was available, a user could expect to be punished in relation to how their cooperation level deviates from the norm – possibly reforming their ways in order to avoid potential punishment.

Altruistic punishment differs from normal punishment in that the punisher does not receive any direct benefits from punishing an individual. To prevent people from punishing others limitlessly, there is often a relative, associated cost for the punisher. For example, if Alice punishes Bob by 3, Alice must absorb an associated cost of 1. Fehr and Gächter [41] documents Public Goods experiments where an altruistic punishment option is made available to participants. In this game, players were made aware of how much other players had invested into the common pool. Players were punished by other players the more they deviated from the group norm. This behaviour is interesting because punishers were willing to absorb higher costs in an attempt to keep free-riders in

line with the group norm, even though punishment had no direct (and possibly no indirect) benefits for them.

Altruistic punishment has been shown to be highly effective in encouraging cooperation among groups. A study cited by Fehr and Schmidt [43] ran a Public Goods game with and without punishment. When punishment was not available, cooperation within the group decreased to zero as the game progressed. With punishment 80% of participants were cooperating fully by the last round. Other studies conducted also suggest that without punishment, cooperation converges to zero, but with punishment cooperation is either maintained at a higher level, or converges towards full cooperation[41]. If punishment was costly for the punisher, why would they still do it? [42] has shown that people value fairness and cooperation in other people over selfish, uncooperative behaviours. Fehr and Gächter [44] reviews the work of Offerman [45] and Charness and Rabin [46], summarising that the tendency to punish antisocial behaviour is stronger than the tendency to reward prosocial behaviour.

## 5.4 FORMS

How should punishment work? Who does the punishing? What happens when one gets punished? These questions need to be addressed for any punishment system to be effective. To determine the best way to influence a persons' behaviour, it could be helpful to determine why people are on a file sharing community in the first place and what they value most on said community. The poll discussed in 2.2 provided useful data in this regard. By removing, or limiting access to features that provide what users most value, it seems likely that punishment can be most effective. Whether there is a publicly accessible 'reputation' value available to peers (for example, the Instant Messenger client discussed in 4.4) is also a matter of consideration. Should punishment histories be available publicly? Henrich and Boyd [38] indicates that reputation is an essential requirement to encourage prosocial behaviour and in [47], Keller and Reeve demonstrate that familiarity is a major catalyst for cooperation. This familiarity can enable reciprocal behaviour amongst group members; several studies and shows that 40-60% of a pair wise Public Goods game players act reciprocally, with 20-30% making selfish decisions[44]. Reciprocal behaviour can be summarised as treating well those that treat you well, and treating poorly those that treat you poorly.

Cooperation between individuals that have previously had no contact (and thus cannot be explained by reciprocity) cannot be explained easily. Axelrod *et. al.* [48]

theorises that individuals can use ‘tags’, publicly observable characteristics (for example, feather colouring) which can alter the likelihood of another individual cooperating with it. Other individuals are biased to cooperate with those of a similar marking to their own. This behaviour can be observed in nature, and online in the form of nickname tags. In online gaming environments for example, it is highly common to see player names prefixed by a clan tag that signifies the player is a member of a certain clan. This can serve as a warning to other players that do not belong in the same clan, and provides a way for other clan members to support each other while gaming. This type of prefix has also been observed by the author in other online mediums such as IRC and DirectConnect.

To summarise, if a user can know about with whom they are interacting with, and how the other party has interacted with others could improve the prosocial-behaviour forming effect of punishment. Conversely, Axelrod *et. al.* [48] has shown that cooperation between individuals can occur without either party having any previous knowledge of each other. In these situations cooperation is thought to resort to public characteristics[48].

In much of the work discussed, ‘cost’ is often used in the monetary sense of the word (some studies were done for actual monetary reward), so does this equate to P2P? This thesis uses the words ‘moderation’ and ‘punishment’ interchangeably, with the more neutral phrasing of ‘moderation’ favoured in later chapters. In poll of 20 P2P users (full results in Appendix A.2), participants were asked if they’d moderate antisocial behaviour if given a chance, 70% said yes, with 42% still agreeing to moderate, even if they had to absorb a cost. While 20% of participants would not moderate at all, this figure jumped by 11% when told there would be a cost involved.

## 5.5 CONCLUSION

People are often inclined to follow the path of least resistance. If it is less costly for them to not cooperate, they will tend to do that on the other hand, if it is less costly to cooperate, they will tend to do that. By introducing punishment into a group, the perceived costs associated for non-cooperation can be increased, to produce higher overall cooperation.

# 6 IMPLEMENTATION

## 6.1 INTRODUCTION

While producing designs and algorithms for distributed systems is a complex process, implementation is usually more complicated - Naanou was developed over an eight month period, and contains over 35,000 lines of C# code. The end result is a fully-functioning, “from scratch” implementation of a peer-to-peer system. An overall schematic of the structure of Naanou is shown in Figure 2.

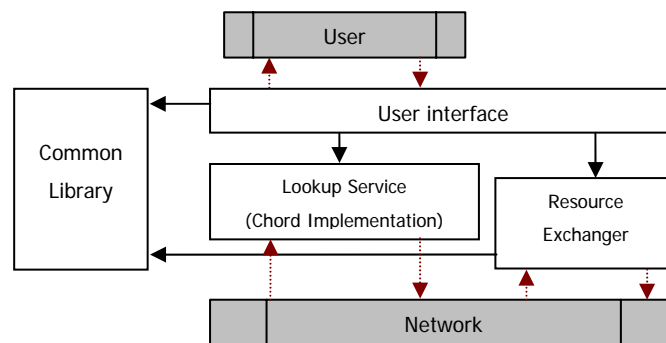


Figure 2: Schematic diagram of Naanou. Solid arrows indicate code linking relationships, dashed arrows indicate information flows.

This chapter details how the major components of Naanou were implemented, starting with the Chord implementation. 6.3 introduces the moderation implementation, followed by keys, downloading and user interface.

## 6.2 NETWORK

### 6.2.1 Introduction

Chord is a theoretically highly scalable, high-performance protocol, however current implementations of Chord exist only within testing (and often only simulated) environments, and have not been used extensively in the “real-world”. This is also

generally true of other distributed hashtable P2P implementations<sup>1</sup>. How well a distributed-hashtable based network would perform in a P2P environment is an open question.

### 6.2.2 Transport

For Naanou, network communication between nodes is done via a binary-formatted remote procedure call (RPC) mechanism using TCP/IP as the transport. The .NET framework that Naanou uses is particularly flexible in terms of how RPCs are transported, and can allow for any number of plug-in transports. There are transports also available for protocols such as HTTP<sup>2</sup> and SMTP (Simple Mail Transport Protocol), which can enhance the availability of Naanou by tunnelling through popular, pervasive protocols. By the use of plug-in RPC formatters (responsible for serialising and deserialising method calls and objects), RPC flexibility is further enhanced. The most common method for transports in existing P2P implementations is UDP or TCP/IP for control messages, and TCP/IP for data exchanges. A RPC mechanism was chosen for simple implementation, and the flexibility of plug-in transports and formatters. The simplified external RPC interface for major functionality is presented in Figure 3.

```
void FillFingers(string[] fingerLocations)
void FindSuccessorNodeAsync(string key, FindSuccessorResult callback)
void GetKeyAsync(string key, string inserter, GetKeyResult callback)
void PutKey(string key, KeyContainer key)
void NotifyPredecessor/Successor()
long Ping(long id)
void QueryAsync(SearchQuery query, QueryResult callback)
```

*Figure 3: Simplified RPC interface*

### 6.2.3 Peer Discovery

As the Chord protocol does not specify how peers are to initially locate another node to join the network, a peer discovery mechanism was required. Peer discovery is very difficult to implement in a completely distributed nature, and there are no current systems that approach a high level of decentralisation. In Chord, a client needs to only know of only one other active node (already connected) to join the network. After

---

<sup>1</sup> There are only two other distributed hashtable implementations that the author knows of that are in “real-world” use, see 2.3 for discussion.

<sup>2</sup> See Appendix B for a definition

surveying existing mechanisms available in current systems, a hybrid approach was taken for Naanou. Discovery can take place via broadcast, web bootstrap or a manually defined address (by default attempted in that order).

Broadcast discovery sends a small UDP packet to the broadcast address of each of the local Ethernet interfaces available on the local machine. Naanou nodes already running continually listen for such packets, and on receiving a request, reply with their own details. This method is highly decentralised, but requires another node to be running on the same network segment as the newly started node. While this works well in local area network (LAN) environments, it does not work on many commercially available Internet access methods.

Web bootstrap discovery accesses one or more well-known URLs (defined in a configuration file) and downloads their contents. The contents are parsed, looking for special tags. The tag format is: `[nn!NODE_ID!NODE_URI!NETWORK_ID!PRIORITY!nn]`, for example: `[nn!100!tcp://192.168.119.200:8000!200!0!nn]`. These tags can be present anywhere in the content of the document (including non-viewable areas such as HTML comments), and can be served from any type of web URL, for example a single-purpose PHP[49] script, static HTML[50] or a public forum entry. By making the format of tags simple, and without requiring HTTP header values to be set, any user that has access to the web can effectively help other Naanou users find the network – decentralising the discovery system. Another benefit of having plain-text tagging is that existing web services such as search engines and public web caches can be utilised to list tags for a new node. Plain-text tagging is different from other web-based bootstrap mechanisms which usually require the host to run a system of scripts and remain online on a semi-permanent basis to service peers.

The last resort for peer discovery is the manual method. A user can specify a list of space-delimited node addresses in their configuration file, and Naanou will try each in turn. This list could be inserted by a third-party program, for example it could be gathered from a list of user's on the same IRC channel as them, or a user's AOL® Instant Messenger contact list.

Once Naanou has collected a number of node addresses (the exact amount is determined by a configuration option, by default set to 1), it aborts the discovery process and attempts to join them. Each discovery method has priorities assigned to it, with web discovery supporting variable priorities for each discovery address. For the case where Naanou has several addresses it has gathered from different sources, priorities allow

Naanou to make more informed choices about which addresses to try first (on the basis of the higher likelihood they are online, for example). In the PHP scripts developed for web discovery, nodes are given higher priorities the longer they have been online (based on the work done by Saroiu *et. al.* [19]<sup>1</sup>). Broadcast discovery gives all discovered nodes a high priority (because it indicates that they are online and available at the time of discovery start), and manual specification gives nodes a medium-level priority. Naanou groups all discoveries by priority level, and then by the network id of each discovered address. Discoveries are sorted by priority level, and within priorities, by the size of each network (as represented by discovered addresses). As it is possible for multiple networks to exist simultaneously (although there are mechanisms in place to prevent it, and to heal partitioned rings), a starting node will join a larger network over a smaller network. Once a node selects and contacts an online node, it is quickly synchronised to the network and ready to lookup or store keys.

#### 6.2.4 Chord Improvements

The research describing Chord makes reference<sup>2</sup> to a number of improvements that could be made to Chord to improve the reliability, robustness and performance. The author has developed upon and implemented some of these improvements for Naanou. In particular, the network partition detection and healing, lower lookup hop-count, node caching and reverse fingers recommendations were used.

Partition detection is designed to detect multiple separated Naanou networks forming. If the Chord ring structure breaks (due to a large number of node failures, for instance), and heals again, two or more sections of the network have the possibility of forming individual networks. Lookups within these partitioned networks are only performed within the localised network, and thus is a situation that needs to be prevented. Naanou re-performs the node discovery process periodically, at random intervals in an attempt to locate peers that are on a different network to itself. If a network with more peers is detected, the client will switch networks. Other stabilisation techniques discussed in the Chord research are also implemented.

---

<sup>1</sup> See 3.2.4 for further discussion

<sup>2</sup> [5] page 42

## 6.3 KEYS

### 6.3.1 Introduction

As Naanou is built upon a DHT lookup/network service, it is useful to think of each object within the system as having a key-value pair associated with it. Once a key is known, the location (where the key is stored) can be established, and a value for that key retrieved or set. This is the basic primitive provided by the Chord implementation, with node lookup, file searching and file indexing capabilities built upon this. Each node in the network is responsible for a particular section of *keyspace*, a finite space (the bounds of which are determined by the bit length of the hashing algorithm used) based on their node id. Naanou uses 160-bit SHA-1[16] hashes. Nodes organise themselves into a ring-like structure, in order of their ids with lookups forwarded around the ring until they reach the desired node. Chord uses various means to improve upon this basic notion of a lookup to guarantee  $O(\log n)$  hops, even in the event of a large number of simultaneous node failures.

### 6.3.2 Replication

To ensure keys remain available in the network even after the node responsible for storing that key has failed, Naanou replicates stored keys to nearby nodes. If the primary node fails, the failed node's keyspace usually comes under the control of the node's predecessor, and as it has a replicated copy of all keys it can continue to provide access. By increasing the number of nodes Naanou replicates to, the availability of keys can be increased, at the expense of higher network traffic ( $O(n)$  only). As keys replicate outwards, an *authority* field on each key is increased, starting from 0 at the primary node. The authority field is used in an exponential decay function on the original expiry time of the key - the more hops away from the primary source a key is, the quicker it will expire. Replicated keys can also be used as a shortcut when searching. For example, when a query arcs in towards the best node for that lookup, it may travel through a node that has a replicated key that matches the query. The node can reply to the query, reducing the hop count of the search - the act of replication is analogous to a caching mechanism. As keys expire, the node that originally inserted them into the network automatically reinsert before they become unavailable on the network.

### 6.3.3 Node Ids

Each Naanou node has a unique id, which is the hash of its IP address. This id is used for message routing in the Chord ring, and determines the ‘position’ of the node within the key space. IP address based ids were selected because of the publicly observable and verifiable nature of an IP address. When a node joins the network, and notifies others what its’ id is, any other node can then verify this independently, by examining the socket address that the node is connecting from. To prevent IP address spoofing attacks, a node can use a ‘dial-back’ mechanism where after initial introductions, connections are dropped and reinitiated from the other side.

When starting up, each node also computes a unique *fingerprint* for itself, which should identify that user on that particular machine with a reasonably high degree of certainty. The primary design goal for the fingerprint is to prevent collisions with other nodes (as fingerprints are used in moderation), with the secondary goal being to make the fingerprint relatively stable over the course of time. The fingerprint is the hashed combined values of the physical hard drives serial numbers, CPU id, operating system serial number, storage volume signatures and user name. Thus a fingerprint could be assumed to remain relatively static, given the attributes that make up the fingerprint do not change frequently for the average computer user. This would mean that different users using the same computer would have different ids, but the same user would have different ids depending on what computer they were using. Fingerprint plaintext is sent in the header of all transmissions carried over the network. The benefit of using plaintext is explained later in 6.4.2, but there is an obvious privacy concern with sending around plaintext that uniquely identifies a user-machine combination.

### 6.3.4 Publishing

Key publishing is a simple process. First the node that is to store the key is looked up via the Chord primitive *FindSuccessor*. Once the node is located, the key to be stored (be it a moderation, a file or other type of key), it is serialised to binary blob, and sent to the storer. The storer reconstructs the key (all keys implement a common interface), and stores it in a hashtable. A key can contain opaque data, and as such, could be used for storing the actual file data as well, as per Freenet. When the storer stores a key, it sets a time for the key to expire at, based on the publisher’s recommended expiry length, and records which node stored the key. Keys in Naanou can only be removed by the node that inserted the original key. This is authenticated based on the node id and the socket address – if either of these does not match the original inserter, the delete request is

dropped. Replicated copies of keys are not deleted, and remain active until they expire. While deletion of keys is supported in the Naanou API, no part of Naanou currently uses this functionality, and there is an immutable bit that can be set on keys to prevent any deletion (other than by expiry).

### 6.3.5 Metadata

To make searching effective, resources are published on the network based on their metadata values. Naanou can natively read metadata from audio-video interleave (AVI) video files, MP3 and various image formats, and can support other easily. Where metadata is not available, or is incomplete, the indexer falls back to using the resource's filename to describe it. Describing a resource by its metadata is a powerful technique, for example it allows users to filter resources by certain attributes (e.g., hide all video files under one hour long), or just find out more information about a resource without having to download it (e.g., video resolution). Sample metadata that Naanou extracts is shown in Figure 4.

```
Video file
  Frame: 640x352
  Duration: 01:51:56
  Codec: DIV4
MP3 file
  Title: Presto – Ode to Joy
  Album: Symphony No. 9
  Artist: Beethoven
  Bitrate: 192
  Duration: 00:18:03
  Year: 2001
  Genre: Classical
JPEG file
  Frame: 2000x1312
  Artist: Bob West
  Camera: NIKON D1H
  Description: Africa dawn
```

*Figure 4: Extracted metadata from various resource types*

### 6.3.6 Searching

While distributed hashtable-based P2P protocols often feature high performance characteristics, their limitation of key to value lookups makes searching difficult. In a hybrid network such as Napster, queries are sent to the central server, which could easily do text matching on the query against its indexing using existing information retrieval

techniques. The Napster server was essentially an open database that could be modified and searched by external users. Unstructured networks such as Gnutella pass requests around the network, performing text matching against their local index and routing replies to the original requestor. DHT systems offer only a very simple search primitive. Given the key  $k$ , find the node on the network responsible for storing  $k$ . How can traditional text searches be mapped to this primitive? In most other systems, a user can search for *'radio'*, and retrieve results for *'radiohead'* or *'radioland'* because of the text-based matching. In a DHT however, keys are hashed to a number. The keys for *'radio'*, *'radiohead'* and *'radioland'* may be 100, 5 and 93256 respectively. How is the system to locate possibly related keys when a uniform hashing<sup>1</sup> algorithm is used?

To improve the searchability of resources within a DHT, the system needs to be able to map a user's query to a set of likely target keys. To address this, Naanou uses a combination of methods. When there is no metadata available for a resource, Naanou parses the filename, and converts the name into a series of words. Each word (up to a predefined maximum) is stored separately on the network, along with a series of *context* words. For example the words *'carmina burana primo vere'* get stored on the network as:

- |                  |                                  |
|------------------|----------------------------------|
| 1. Key: carmi na | Context: burana, primo, vere     |
| 2. Key: burana   | Context: carmi na, primo, vere   |
| 3. Key: primo    | Context: carmi na, burana, vere  |
| 4. Key: vere     | Context: carmi na, burana, primo |

Each key, along with its associated context gets stored independently on the network, and most probably on different nodes. When a user is searching for the same filename, or a shortened form, the same process is applied. For example, if the user was searching for *'primo vere'* after the resource was stored on the network, the following keys are queried:

1. Key: primo Context: vere
2. Key: vere Context: primo

Which would yield the following results from the storers of those keys:

1. Key: primo Context: carmi na, burana, vere
2. Key: vere Context: carmi na, burana, primo

Along with other fields associated with the returned keys, the searcher could then initiate a download from the source. Context values are used to calculate a relevancy value for a

---

<sup>1</sup> *Uniform hashing* is a type of hashing algorithm that spreads keys evenly over entire keyspace

search result. In the above case, our first query matches a result that contains keys for both our first query and second query, which would indicate the words are used in a similar context, and thus more relevant. If this splitting approach was not taken, the words ‘*carmina burana primo vere*’ would be stored at one location on the network, and would only be able to be located if the searcher used exactly the same text. To further increase the chance of finding results, all keys are converted to lower-case, and stop-words<sup>1</sup> removed, thus ‘*Tales from the Viennese Woods*’ would get converted to ‘*tales viennese woods*’. If the percentage of stop words removed passes a threshold (default is 70%), stop words are not removed, to prevent resources having empty descriptors.

A similar process takes place for metadata values, with metadata tuples stored individually, and the other tuples forming the context. Individual words are not split as in the filename search, however. Each type of metadata extractor in Naanou defines what metadata tuples are to be descriptors (and thus searchable, and which are to be just published. For example, an MP3 file may have artist, album and title fields as descriptors, while other fields such as track number or year are just published. When a tuple is a descriptor, it is stored on the network, along with context tuples. When a tuple is just published, users cannot search for it directly, but can view the data in a search result or filter results client-side using the value of the fields. For example, a resource may have the following metadata tuples available:

```

Artist: Orff
Album: Carmina Burana
Track: 01
Title: Primo Vere Uf Dem Anger
Year: 1924
Genre: Classical

```

As a user is unlikely to want to search by track number, year or genre (and as the fields are so generic, the results possibly returned by such searches would be huge), they are not published. Naanou publishes the resource as follows:

```

Key: artist: orff
    Context: album: carmina burana, title: primo vere uf dem anger
Key: album: carmina burana
    Context: artist: orff, title: primo vere uf dem anger
Key: title: primo vere uf dem anger
    Context: artist: orff, album: carmina burana

```

---

<sup>1</sup> A list of the most commonly-used 100 words in the English language

When a user now begins a search for one of three keys published, they will locate the correct resource. Note that in both metadata and filename-based key search, spelling is important. If a resource is published under *'awrff'*, a user searching for *'orff'* will not locate it. This is not a problem however, as if one user spells something wrong, it is possible that another user misspells the word the same way, and they can easily locate each other. For all file resources stored, each key published about the file contains a reference back to the original file, extra descriptor fields not published and information about how clients can download the file. This reference is the unique 160-bit hash of the file contents. As the file id is a separate entity from metadata or filename descriptors, resources that have the same content can be located, regardless of how they are spelt or described<sup>1</sup>.

These techniques combined allow the user to search effectively based on attributes they most likely want to use. While there is a higher overhead because of the multiple key stores per resource, there is no competing design that can do as flexible searches within a DHT environment<sup>2</sup>. Result quality could further be improved by applying a word stemming algorithm to metadata and filename words. Stemming algorithms can convert noun plurals into their singular equivalents, and other such transformations to improve the probability that the keys used by the publisher and the searcher are identical.

## 6.4 ALTRUISTIC MODERATION

### 6.4.1 Introduction

As discussed in 5.4, moderation should affect what the user values most in order to be most successful. Based on the poll discussed in 2.2, the implementation provides for the following ways to affect the user experience:

- Download speed can be throttled down to any arbitrary value,
- Search results returned by nodes can be shortened arbitrarily, and

---

<sup>1</sup> Naanou uses this file id internally, as will be explained in 6.5

<sup>2</sup> In the Freenet specification, there is a reference made to proposed 'indirect files' which would act in a similar way to Naanou's.

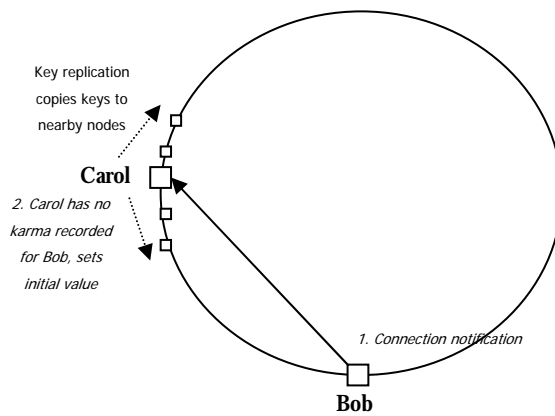
- Users can be ignored, and prevented from participating on the network in any fashion.

When a user first joins a network, an initial, median-level *karma* value is established for the user. Karma values for users are stored on the network itself like other keys, and persist from session to session. Moderation by a user decreases the punished user's karma, which slowly refills over time. When moderating, a user can choose the severity of the moderation, on a 1 to 10 scale. As moderation follows the research discussed in 5.3, the moderator also gets affected by his or her moderation act. In Naanou, a ratio of 3:1 is used; for every 3 moderation points inflicted on someone, the moderator is inflicted by 1. When a user's client interacts with another client, for example when a remote node is performing a search, or requesting a file for download, the local client can perform a lookup for the remote user's karma value. Once retrieved, the value can be used to modify the quality of service the remote user receives – the lower the karma, the lower the quality of service (QoS).

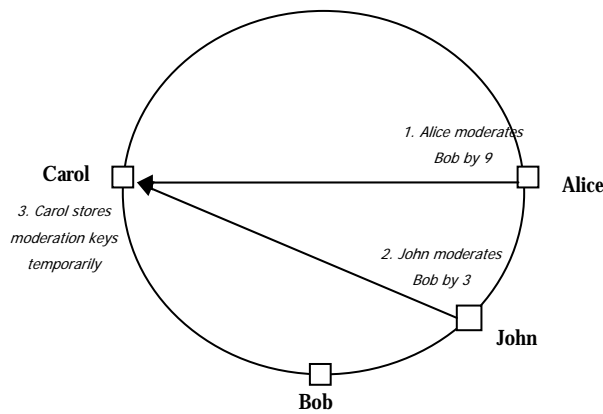
Karma values are stored on the network as per normal keys, but do not expire for a much longer period. Karma is keyed to a node's fingerprint which is embedded in all transmission headers, so once a node id is known, the associated karma can be located. When a user requests to moderate another user within Naanou, a moderation key is created, set to correct level, and stored at the karma key position on the network. Each moderation key has an individually determined expiry length, which is a function of the moderation severity (the more severe the punishment, the longer it lasts). When a user requests the karma level of another user, they use the karma key for that user to locate a set of all moderation keys. These keys are concatenated locally by the requestor to establish a karma value for a user. As each key has its own individual expiry time, karma eventually is replenished as keys expire.

'In-turn' moderation, where the moderator themselves get punished, also needs to take place. Say Alice wants to punish Bob by 9 points. Carol, as it turns out is responsible for storing moderation keys accrued against Bob (based on her node id, and Bob's fingerprint id). When Carol receives Alice's moderation key, she does not store it immediately. Instead, Carol needs to make sure she can do in-turn moderation on Alice first. Carol locates the node – Mary - storing Alice's karma and sends a message indicating she needs to moderate Alice by 3 (because of the 3:1 ratio), due to Alice's request for moderation against Bob. Mary, on receiving the in-turn moderation request

contacts Alice to confirm that she did indeed want to punish Bob by 9 points. If Alice denies knowledge of this, Mary sends a negative acknowledgment back to Carol who then discards Alice’s original moderation request. If Alice confirms this, Mary records the in-turn moderation key from Carol and sends a positive acknowledgment to Carol who then proceeds to record the moderation key from Alice. Thus, if Alice wants her moderation request to be honoured, she must acknowledge Mary’s query, and receive in-turn moderation. These processes are illustrated below in Figures 5a-d.



*Figure 5a: Bob’s first connection to the network. Carol is responsible for storing the keyspace for Bob’s karma.*



*Figure 5b: Alice and John moderate Bob*

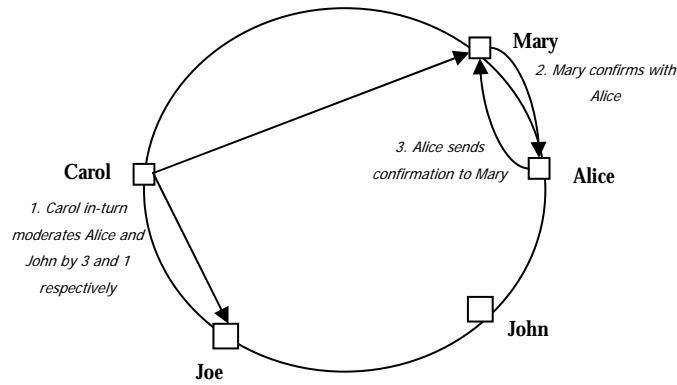


Figure 5c: Carol in-turn moderates Alice and John. Mary is the moderation-key storer for John, and Joe is the moderation-key storer for Alice. The interaction between Joe and John is not shown.

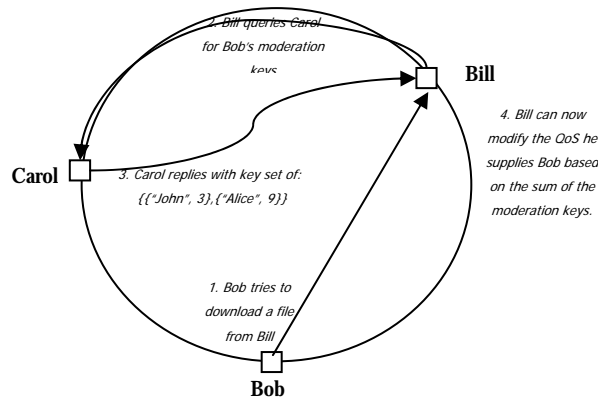


Figure 5d: Karma lookup

### 6.4.2 Security Implications

Storing karma on the network does make it susceptible to improper modification by a user. As karma directly impacts on a user’s experience, it is in their best interest to maintain their karma at the highest level possible. While needing to prevent a user from modifying their own karma level, the system also needs to allow any other user on the network to decrement the level. This section will examine several methods of attacking the karma implementation. It is assumed that the underlying transport is infallible, and communication between nodes is secure<sup>1</sup>. Several methods of attack rely on ids on the network not being selectable by a client. If a client could choose any node id it wishes, it can effectively position itself on the Chord ring where it can do most damage (i.e., by being the best node selection for message routing).

<sup>1</sup> Such security could be implemented via existing technologies such as a Secure Sockets Layer (SSL)

**Raising one's own karma** – Moderation keys record each moderation action taken by others thus, to raise karma these moderation keys would need to be removed prematurely. By setting an immutable bit on moderation keys, they cannot be removed by any delete request. Another way to force deletion would be to have keys expire earlier – for example in one second, rather than the usual range of minutes to hours. Moderation key expiry is set by the node that stores karma based on the moderation value (as opposed to regular keys, which have expiry lengths set by the key publisher). The only way for this attack to work would be by having the karma-storing node compromised. As the moderator has no choice in picking the node that stores karma, it cannot direct moderation requests to a node that is already compromised (or itself). There could be the case where a node is responsible for storing its own karma. This probability is directly related to the number of nodes present in the network, and would be unlikely in a reasonably sized network.

**Dropping moderation keys** – An attacker could accrue a higher karma by using a compromised karma storer to drop moderation requests by other nodes. This attack is identical to the previous attack, and suffers from the same difficulties.

**Preventing in-turn moderation** – Preventing in-turn moderation would be a successful method of keeping one's karma levels high - to moderate others as much as the attacker wanted without any consequences. The attacker's karma would never decrease because of in-turn moderation, and they could carry out moderation attacks against other users, preventing them from using the network. Because a successful moderation action is only committed when the karma storer successfully in-turn moderates the moderator, the karma storer for the original moderator would need to be compromised (to prevent it sending out an in-turn moderation request).

If an attacker was to send a moderation request marked 'in-turn', instead of a usual moderation request, could in-turn moderation be avoided? For example, say our attacker Alice wishes to punish Bob by 81 points, but wants to avoid the in-turn moderation of 27 points she would normally receive. Alice crafts a moderation request that appears to be an in-turn moderation request – that is, a message that would normally be sent if Alice was in-turn moderating someone else. For her plan to work, she would need to make it appear as though Bob has moderated someone by 243 points (81x3), and

she is performing an in-turn moderation against Bob (as per protocol) of 81 points. The only node that can store moderation requests against Bob is Carol. When Carol receives in-turn moderation, she contacts who she believes is the original requestor – Bob - to confirm whether they did indeed make the moderation request. This is where Alice's plans come unstuck, because the original requestor must be Alice's target – Bob. Bob of course has no knowledge of the moderation request, and denies it. Carol drops Alice's in-turn moderation request, and prevents the attack from taking place. Furthermore, in Alice's in-turn moderation request, the invented target for Bob's moderation must also be included. Carol could automatically drop any in-turn requests where the node id of the sender does not equal the node that should be storing the karma for the target, which would improve the efficiency of attack prevention.

**Muling** – 'Muling' is a term used in online role playing games where a single person maintains several online characters in order to exploit their combined strengths, or to circumvent per-character limitations imposed. For example, a character may be created – the mule - just to hold game items for the person's main character. Such behaviour is usually forbidden by game administrators. In Naanou, muling could be used by a person to avoid the bad karma penalties. If the person had access to multiple computers, or was using a modified Naanou client (that could generate different fingerprints each start-up), as soon as one fingerprint starting getting negative karma accrued against it, could change fingerprints – starting anew. This is a challenging problem for Naanou to address, and this implementation is not fully resistant to it. To compensate for damage that could be caused, every time a fingerprint is first seen on the network, it is given a medium-level karma level. Meaning, for every new mule created by a person, they would be receiving only half karma that they would have had, had they been online for longer, reducing the incentive for muling. The level of the starting karma could be lowered further to additionally reduce incentive. Further protection could be provided by requiring all clients to have an identity certificate, signed by a certification authority, such as Verisign.

**Creating fake moderation keys** – Each node on Naanou has a chance of being the karma storer for another Naanou node. In the present Naanou design, there is little protection against renegade karma storers that generate fake moderation keys and store them locally. The difficulty in performing this attack against a *designated* user is that the attacker would have to control the node responsible for storing the target's karma. This could be

done by attacking the current node storing the target's karma using out-of-bound techniques or by modifying their own node so they become the storer for the target's karma. The first technique is outside the scope of the design, but the second is a possibility. To carry out this attack, the attacker would have to make their node id closer to the target's karma key – so other nodes on the network will direct moderation request to it, rather than the true storer. As node ids are the hash of the node's IP address, the attacker would have to modify their IP address. In most mainstream Internet connection types, this is impossible and even in instances where it is possible, an attacker only has a limited range of IP addresses to choose. Further complicating the matter is that the attacker would have to decrypt the SHA-1 algorithm to find an IP address that would give them the correct id. Decrypting SHA-1 is considered a 'hard' problem, but given the limited range of IP address space, is something that could be brute-force attacked.

To summarise, a node *can* produce fake moderation keys unchecked, but will have difficulty in targeting a particular node. Security could further be enhanced by requiring that all moderation keys be signed by the moderator. A node, on looking up the karma level for a node could then verify each key's signature before forming a decision on the karma level. A simpler model may be to just use the current 'moderator' field on keys, and when checking a karma level, verify with each moderator that they did indeed set the key (of course, this would require that the moderator is still online). There would be a considerable extra overhead involved with these two checks (particularly in the latter method), but these methods could be included into Naanou if fake key creation became a problem.

**Karma stealing** – What is to stop an attacker from assuming the identity of another user, inheriting their good karma (or worse, once inheriting the karma, running it down to zero)? To perform this attack, an attacker would have to have their node use a fingerprint equal to another node. When remote nodes have dealings with the compromised node, they would in fact be looking up the karma for another existing node on the network. In the current implementation, there is no protection against a hacked Naanou client using invalid fingerprints, but there are several ways to address this issue. If moderation keys were associated with the node id of a client as well as the fingerprint, the attacker would also have to overtake the node id of the target (i.e. the same IP address), which is considered a difficult problem (see previous paragraph). With this method, there is a problem as node IP addresses are subject to change; how would the *real* node maintain a

link with their karma level? Another, possibly more resilient approach is to use signed certificates in conjunction with public key encryption to verify a node's identity.

### 6.4.3 Conclusion

Naanou's moderation implementation is an effective decentralised reputation mechanism in a distributed hashtable network. The security implications are not insurmountable, and many can be addressed through the use of a public key architecture for node identities. This would add effort (and possibly cost, as many certification authorities require payment for certification issuing) for the user, but would significantly increase the security of the system. Additionally, resilience against attacks can be achieved by not having a single node responsible for karma storing, rather a group of  $k$  nodes. Moderation keys can be individually stored at each node, thereby preventing a storer arbitrarily dropping moderation keys, and forcing an attacker to have to compromise  $k$  nodes instead of one node if they wished to alter their karma. When checking karma, a node would query all  $k$  nodes, and arrive at a conclusion using an existing polling algorithm (for further discussion, see [51]). In its current implementation, Naanou's moderation mechanism deals with most attacks well, with the exception of karma stealing.

## 6.5 RESOURCE EXCHANGING

### 6.5.1 Requirements

Obviously, in a file sharing P2P network, the ability to effectively exchange files is an important attribute. Users expect to be able to download at a fast rate, and when their client is sending to others, they would like to be inconvenienced as little as possible. With this in mind, the design goals for the resource exchanger component are:

- Maximise speed of downloading, minimise inconvenience of uploading
- Maximise availability of resources
- Cater for varying connection speeds of users
- Support both searching and browsing of files
- Ensure transmission is reliable and files are correctly sent

Several existing P2P clients address these points to varying degrees. An effective approach that is used in the more advanced clients is multi-source downloading, or '*swarming*'. This download technique tries to saturate an incoming link by simultaneously

downloading many parts of a file at once, usually from different sources. Using this method, users on slow modems can still be used to serve files, for example a single cable user could have their incoming link saturated by downloading from 20 56kbps modem users at once. To maximise the availability of resources, most clients automatically make the default download directory shared – so as soon as a file is downloaded it is available to others. More advanced clients make segments of a file available as soon as it is downloaded, further increasing availability. To cater for differing connection speeds, clients often have a configuration setting where a user can indicate what speed connection they have. This connection speed is advertised along with any resources a user has shared, and was intended so users of similarly classed Internet connections can seek each other out more easily. This system is often abused, usually by a user with a high-speed connection advertising a low connection speed, in an attempt to avoid people downloading from them. Most clients also have an automatic retry function, which automatically retries a download if it gets disconnected, so the user does not have to worry about it.

### 6.5.2 Design

Naanou borrows from many existing techniques, and introduces its own unique technique to meet the design goals. All resource exchanges in Naanou take place over HTTP<sup>1</sup>, in its' own implementation of a HTTP 1.1-conforming server. HTTP was used for its high performance attributes, along with being able to be tunnelled via existing web proxies. Naanou uses pluggable protocols, so a node could advertise a source for a resource as an FTP URL, for example. In the current implementation, only the HTTP protocol implementation supports most of the advanced features discussed here.

Firstly, Naanou treats all files as a series of blocks. When these blocks are reassembled in the correct order, they make up the original file – which can be verified as being exactly the same by computing the content hash of the file. Once a user has located a file to download via the search function, the client knows the unique content hash for that file. With this hash, the file can be identified even if a user changes the filename<sup>2</sup>, and once downloaded, can be verified that it is the same file the user searched for. Once the hash is known, the client needs to locate sources for the file. Often, if the file was found as a result of a search, it may already know of some sources (as search results

---

<sup>1</sup> See Appendix A for a definition

<sup>2</sup> Thus, helping to address one of the issues outlined in 2.2

contain references to a file source), but the only canonical way of locating a source is to query the node storing the file key. The file key is keyed by the file's content hash, and is stored like any other key on the network. Naanou first tries to use any sources it gathered via the search (to avoid the extra lookup to the file key), but if these fail, it will query the file key. File keys are published for every complete file by every source, so a client can collect a set of file keys from a node each of which contain a reference to a download location. Once the client has a list of likely sources for a file (remembering that a file key could exist after a file is physically deleted at the source), it interrogates each to identify what byte ranges of the file they have available. In this interrogation process, each source also reports back a list of the last ten users that downloaded this resource from them. Because a client does not publish a file (i.e. store the file key and metadata keys for a file so it can be searched for) until it is completely downloaded, these previous downloader lists help to identify other likely sources without any extra effort on their part. As far as the author knows, this feature is not in use elsewhere.

Now that a complete source list is available, along with data on what parts of the file each source contains, the client constructs a 'map' of the file, by overlaying available byte ranges and sources to identify what byte ranges are most available (that is, has the most sources), and those that have fewer sources - or none. Naanou then proceeds to download ranges in parallel, prioritising ranges that have fewer sources. Configuration options allow the user to set how many ranges to fetch simultaneously for a single resource, and per user (for example, setting the client to download a maximum of 20 ranges in parallel, but only ever 5 at a time from a single user). As each range is downloaded successfully, it is stored into a temporary block storage area, which is a staging area for later file reassembly. As soon as a block is saved into this area, it is available for other peers to download from, and Naanou uses its block storage area internally to cache file parts. The size of the block cache is kept to a user-defined maximum (and an enforced minimum of 20MB), and blocks are evicted on a least-recently-used policy.

Naanou adapts its download mechanism depending on who it is downloading from. Every source a user downloads from for a session is remembered, along with statistics about previous transfers. When the client needs to download a new block from a source, it first checks to see if it already has downloaded anything from it before, and if so – what the average speeds were. If no record of a previous transfer was found, Naanou gets a small block from them – 512KB (2<sup>9</sup>). If that block was downloaded within a

minimum amount of time, the next block size fetched from the host is a higher power of 2 (up to a maximum of 14 – 16MB). Block sizes grow or shrink depending on network throughput, and are remembered for subsequent transfers for that host. Sizes grow to increase throughput, by reducing the overhead incurred for each block transfer. Sizes are kept from growing too high so slower hosts can transfer complete blocks within a reasonable amount of time. With this unique, adaptive download mechanism modem users can contribute more effectively to others' downloads because their connection is not being tied up with large, lengthy uploads. By having an adaptive (and initially small) block size, unreliable hosts can still manage to send complete, usable blocks of a file. Additionally, transfers can be individually throttled to a set throughput, with this feature being used in conjunction with moderation.

Files shared by a user can also be browsed, via a dynamic HTML interface (shown in Figure 6f). Metadata attributes stored about resources are also shown, with some resource types given more descriptive names than their filename when metadata is available. These pages are generated dynamically by Naanou, as required.

## 6.6 USER INTERFACE

### 6.6.1 Requirements

A usable, well-designed interface is required to make a large, complex system appear simple. The major design goals for the Naanou interface were:

- Make common tasks as direct as possible
- Inform the user about system status and operation progress

### 6.6.2 Design

The design of Naanou's interface is best demonstrated by screen shots taken from a running client. These are shown in Figures 6a-g, along with short notes.



Figure 6a: The initial splash screen. This appears while Naanou is starting up and initialising, and until a connection to the Naanou network is established. A user is kept updated of progress via status text at the bottom of the box. The main window is kept hidden until Naanou is ready.

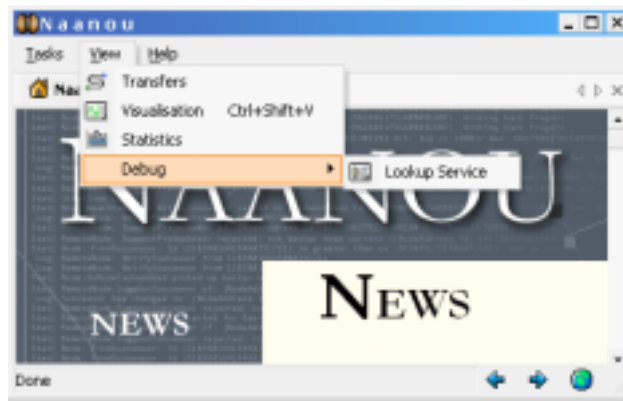


Figure 6b: The main window, where all major functions are accessed from. Naanou uses a multiple document interface methodology, with the default document the Naanou web site, viewed through an embedded web browser.



Figure 5c: Search window. A user can select what they wish to search by (for example filename or audio metadata), and the interface dynamically changes to accommodate. After filling out fields, the user clicks 'Search' and the search commences. Results are shown as they arrive in the table below. The columns used in the table depend on the type of data searched for. A user can also filter out unwanted results, either by user or by resource (e.g. excluding every thing shared by a certain user, or excluding all instances of a certain file). Results can be sorted by clicking column headers, and columns can be rearranged as the user desires.



Figure 5d: Exchange monitor. As resources are being sent or received, the monitor allows a user to track their progress. The window is split into incoming/outgoing panes (each resizable), with relevant status shown for each. Aggregate statistics are shown on the far right of each.

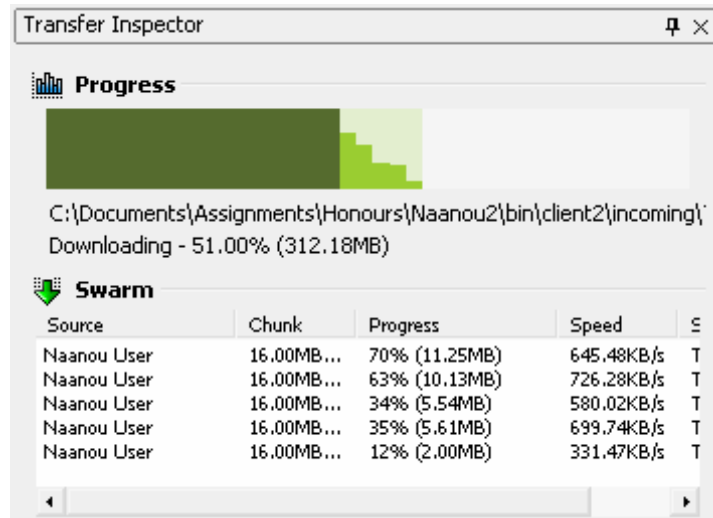


Figure 5e: Transfer inspector. When double-clicking on a transfer or using the context menu the inspector is displayed, showing status of the swarmed download graphically and via a table. Dark green represents completed blocks of a file, medium green is downloads in progress (bar height represents completion amount, bar width represents size of block), and white represents data still to fetch.

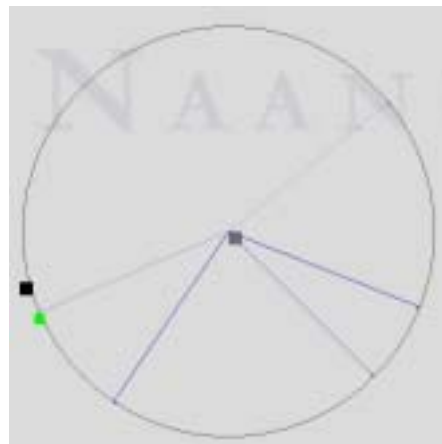


Figure 5f: Visualisation window. This window can indicate how busy the local node is, with all key lookups and stores and being animated on screen. Lines appear and fade out in a aesthetically designed manner



Figure 5g: Resources shared can also be browsed directly via a web interface. Clicking on a file link commences download.

## 6.7 CONCLUSION

This chapter introduced various innovative techniques for improving upon current P2P experiences. It also demonstrates methods allowing for flexible searches and maintaining a distributed reputation system within a distributed hashtable network. After releasing Naanou publicly for a period of two weeks, five random participants were asked (on a scale of 1-5, 5 being the highest), how they ranked Naanou against other P2P systems they have used. The users ranked Naanou well, giving it an average score of 3.6 overall and good scores on speed of searching, downloading and usability. The full results are shown in Appendix A.2.

# 7 CONCLUSION

## 7.1 FUTURE WORK

### 7.1.1 Summary

There are many components of the Naanou implementation that can be further improved. The public test of Naanou suffered from a small user base, with many users downloading the software, running it and then disconnecting after they had finishing trialling the software but before they could provide feedback<sup>1</sup>. Other users left as soon as they realised they couldn't find what they were after and did not leave the program running. Unfortunately, in the absence of the required number of regular, stable users a solid network could not be formed during the timeframe of the project. It is likely that with more time, better documentation and wider advertising, a larger user base could be established, which in turn would attract more users. This would allow more detailed exploration of the strengths and weaknesses of the program. Although Naanou incorporates a greater range of features than other available P2P systems, existing systems have a larger established user base, thus a probably larger content base, and this what users primarily desire<sup>2</sup>. In order to gain insight into the robustness of the system an internal test was conducted during which approximately 15 Naanou nodes were kept running continuously and users reported good results in using the system.

### 7.1.2 Network

By applying topological binning (discussed in section 3.2.5) to the Chord protocol, network latency between nodes can be improved, with a minimum of extra overhead. Performance could be improved by converting RPC calls into UDP messaging. Although extra overhead would be required to manage this, and would not be as flexible as it currently is, this conversion could prove to be generally beneficial. UDP-based messaging often excludes firewalled hosts from participating, so a hybrid solution might need to be sought.

---

<sup>1</sup> This was observed by reading activity logs

<sup>2</sup> As the poll cited in 2.2 demonstrated

### **7.1.3 Privacy & Security**

Techniques used in Freenet such as source hiding, indirection and signatures could be applied in a distributed hashtable network such as Naanou to guarantee privacy regarding who published what data, and who is searching for what data. This would be an interesting avenue to explore, and would take the best aspects of Freenet (privacy and security) and combine them with a high performance, scalable network layer. As discussed in 6.4.2, certifiable identities of users could significantly enhance the security of the moderation system, and could be adapted to be used in other aspects of Naanou. Freenet's anonymous content distribution system (in Freenet as files are requested they physically travel from node to node, where as Naanou employs direct node to node transmission) could also be adapted for Naanou, given that a high performance block-based transfer system is already in place.

## **7.2 SUMMARY**

This thesis presented the research and design behind a scalable, moderated P2P file sharing system, introducing many new concepts and technologies. Research in altruistic punishment from the biological and economic sciences is applied for the first time to this domain. A novel method for decentralised moderation which can support the altruistic punishment facility was also introduced. The implementation -Naanou - is one of only three other distributed hashtable systems, and the only application of the Chord protocol in a peer-to-peer file sharing system. Surveys and experiments were conducted in order to inform the design, and judge the success of the system.

Naanou represents an important attempt to incorporate novel ideas and techniques into the existing P2P paradigm. In its current form Naanou was well received by the majority of users who trialled the software and it is hoped that with further refinement it could be turned into a commercially viable product.

# ACKNOWLEDGEMENTS

The author would like to thank his supervisors Dr. Peter Sutton and Daniel Johnson for their valued assistance throughout the long period of this thesis. Dr. Tim Mansfield and Dr. Robert McArthur of the DSTC are also thanked for their support. The author would also wishes to thank his parents for their support, and the various online communities and groups that helped during testing and questionnaires. The work presented in this thesis has been funded in part by the Distributed Systems Technology Centre, a Cooperative Research Centre Programme through the Australian Government's Department of Industry, Science and Resources.

---

# REFERENCES

- [1] A. Frank, R. Beutler, and A. Markham, "The Copyright Crusade," Viant 2001.
- [2] Sandvine Inc., "Peer-to-Peer File Sharing: The effects of file sharing on a provider's network," [Online document], 2002, [cited 2002 Oct 19].  
Available at: <http://www.sandvine.com/register.asp?TID=1&ID=1>
- [3] Gnutella Developer Forum, "Gnutella Developer Forum," [Online document], 2002, [cited 2002 Oct 13].  
Available at: [http://groups.yahoo.com/group/the\\_gdf/](http://groups.yahoo.com/group/the_gdf/)
- [4] S. Fanning, "Napster." Los Angeles, CA: Napster, Inc., 2002.
- [5] N. S. Good and A. Krekelberg, "Usability and Privacy: a Study of Kazaa P2P File-sharing," HP Laboratories, Palo Alto 2002.
- [6] J. McCaleb, "Overnet." Brooklyn, NY, 2002.
- [7] P. Maymounkov, "VARVAR." New York, NY, 2002.
- [8] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," presented at SIGCOMM, San Diego, California, USA, 2001, pp. 149-160.
- [9] A. Rowstron and P. Drushel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems," presented at IFIP/ACM International Conference on Distributed Systems Platforms, Heidelberg, Germany, 2001, pp. 329-350.
- [10] I. Clarke, "Freenet: A Distributed Anonymous Information Storage and Retrieval System," presented at ICSI Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, 2000, pp. 46-66.
- [11] NeoModus, "Direct Connect," 1.9.1 ed. Berkeley, CA, USA: NeoModus Inc., 2002.
- [12] FastTrack, "FastTrack P2P Stack." Netherlands: FastTrack, 2002.
- [13] K. Truelove, "Gnutella Protocol Description," Clip2 Distributed Search Systems December 17th, 2000 2002.
- [14] K. Spripanidkulchai, "The Popularity of Gnutella Queries and its Implications on Scalability," Carnegie Mellon University 2002.
- [15] J. Ritter, "Why Gnutella Can't Scale," [Online document], 2001, [cited 10th May].

- Available at: <http://www.darkridge.com/~jpr5/doc/gnutella.html>
- [16] National Institute of Standards and Technology, "Secure Hash Standard," National Institute of Standards and Technology FIPS PUB 180-1, April 17 1995.
- [17] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-Area Cooperative Storage with CFS," presented at 18th ACM Symposium on Operating Systems Principles, Banff, Canada, 2001.
- [18] P. Maymounkov and D. Mazieres, "Kademlia: A Peer-to-peer Information System Based on the XOR Metric," presented at 1st International Workshop on Peer-to-Peer Systems, Cambridge, USA, 2002.
- [19] S. Saroiu, K. P. Gummadi, and S. D. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," presented at Proceedings of Multimedia Computing and Networking, San Jose, CA, USA, 2002.
- [20] D. Kato, "GISP: Global Information Sharing Protocol -- A Distributed Index for Peer-to-Peer Systems," presented at International Conference on Peer-to-Peer Computing, Sweden, 2002.
- [21] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An Infrastructure for Fault-Resilient Wide-Area Location and Routing," U.C., Berkeley, CA, USA, Technical Report UCB//CSD-01-1141, April 2001.
- [22] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A Scalable Content-Addressable Network," presented at ACM SIGCOMM, San Diego, CA, USA, 2001, pp. 161-172.
- [23] C. Plaxton, R. Rajaraman, and A. Richa, "Accessing nearby copies of replicated objects in a distributed environment," presented at ACM SPAA, Newport, Rhode Island, 1997.
- [24] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-Aware Overlay Construction and Server Selection," presented at IEEE INFOCOM'02, New York, NY, 2002.
- [25] E. Adar and B. Huberman, "Free Riding on Gnutella," *First Monday*, vol. 5(10), 2000.
- [26] National Children's Home, "Press Release," [Online document], 2002, [cited Oct 13 2002].  
Available at: <http://www.nchafc.org.uk/news/index1.asp?auto=194>

- [27] Girl Scouts Research Institute, "Teenage Girls and the Internet: Driving without a license on the Information Superhighway," [Online document], 2002, [cited 2002 Oct 13].  
Available at: [http://www.girlscouts.org/news/presrel/NetEffect\\_021302.pdf](http://www.girlscouts.org/news/presrel/NetEffect_021302.pdf)
- [28] Ipsos-Reid, "Uncomfortable Liaisons," [Online document], Ipsos-Reid, 2000, [cited 2002 Oct 13].  
Available at: [http://www.ipsos-reid.com/media/dsp\\_displaypr\\_cdn.cfm?id\\_to\\_view=1113](http://www.ipsos-reid.com/media/dsp_displaypr_cdn.cfm?id_to_view=1113)
- [29] E. Laukkanen, S. Shemeikka, I.-L. Notkola, H. Koivumaa-Honkanen, and A. Nissinen, "Externalizing and Internalizing Problems at School as Signs of Health-Damaging Behaviour and Incipient Marginalization," *Health Promotion International*, vol. 17(2), pp. 139-146, 2002.
- [30] N. Marr and T. Field, *Bullycide: Death at Playtime*. Didcot, Oxfordshire, UK: Success Unlimited, 2001.
- [31] A. Kaukiainen, C. Salmivalli, K. Lagerspetz, M. Tamminen, M. Vauras, H. Maeki, and E. Poskiparta, "Learning difficulties, social intelligence and self-concept: Connections to bully-victim problems," *Scandinavian Journal of Psychology*, vol. 43(3), pp. 269-278, 2002.
- [32] M. H. Vickers, "Bullying as Unacknowledged Organization Evil: A researcher's story," *Employee Responsibilities and Rights Journal*, vol. 13(4), pp. 205-217, 2002.
- [33] America Online Inc., "AOL Instant Messenger," [Online document], 2002, [cited 2002 Oct 13].  
Available at: <http://www.aim.com/>
- [34] AOL Inc., "Instant Messenger Warnings," [Online document], 2002, [cited 2002 Oct 13].  
Available at: <http://www.aim.aol.com/faq/warnfaq.html>
- [35] Mail Abuse Prevention System LLC, "Realtime Blackhole List," [Online document], 2002, [cited 2002 Oct 13].  
Available at: <http://mail-abuse.org/rbl/>
- [36] Mnet Development Team, "Mnet," 0.5.0.14 ed: SourceForge.net, 2002.
- [37] C. Shirky, "In Praise of Freeloaders," [Online document], O'Reilly OpenP2P, 2000, [cited 2002 Oct 13].  
Available at:  
[http://www.openp2p.com/pub/a/p2p/2000/12/01/shirky\\_freeloading.html](http://www.openp2p.com/pub/a/p2p/2000/12/01/shirky_freeloading.html)

- 
- [38] K. Sigmund, C. Hauert, and M. A. Nowak, "Reward and Punishment in Minigames," International Institute for Applied Systems Analysis, Laxenburg, Austria IR-01-031, September 2001.
- [39] J. Henrich and R. Boyd, "Why People Punish Defectors," *Journal of Theoretical Biology*, vol. 11(208), pp. 79-89, 2001.
- [40] E. Fehr and S. Gächter, "Altruistic Punishment in Humans," in *Nature*, vol. 415, 2002, pp. 137-140.
- [41] E. Fehr and S. Gächter, "Cooperation and Punishment in Public Goods Experiments," *American Economic Review*, vol. 90(4), pp. 980-994, 2000.
- [42] K. Sigmund, E. Fehr, and M. A. Nowak, "The Economics of Fair Play," in *Scientific American*, 2002, pp. 83-87.
- [43] E. Fehr and K. M. Schmidt, "A Theory of Fairness, Competition and Cooperation," *Quarterly Journal of Economics*, vol. 114(3), pp. 817-868, 1999.
- [44] E. Fehr and S. Gächter, "Fairness and Retaliation The Economics of Reciprocity," *Journal of Economic Perspectives*, vol. 14, pp. 159-181, 2000.
- [45] T. Offerman, "Helping hurts more than helping helps: the role of the self-serving bias," CREED, University of Amsterdam, Amsterdam 1999.
- [46] G. Charness and M. Rabin, "Social Preferences: Some Simple Tests and a New Model," University of Berkeley, Berkeley 2000.
- [47] L. Keller and K. H. Reeve, "Familiarity Breeds Cooperation," in *Nature*, vol. 394, 1998, pp. 121-122.
- [48] R. D. Axelrod, M. L. Cohen, and R. Riolo, "Evolution of Cooperation Without Reciprocity," in *Nature*, vol. 414, 2001, pp. 441-443.
- [49] The PHP Group, "PHP: Hypertext Preprocessor," [Online document], 2002, [cited 2002 Oct 18].  
Available at: <http://www.php.net/>
- [50] World Wide Web Consortium, "W3C HTML Home Page," [Online document], 2002, [cited 2002 Oct 15].  
Available at: <http://www.w3.org/MarkUp/>
- [51] A. S. Tanenbaum and M. van Steen, "Distributed Systems: Principles and Paradigms." New Jersey: Prentice Hall, 2002, pp. 371-375.
- [52] Distributed.net Inc., "Distributed.net Client." Redmond, WA, US, 2002.

# APPENDIX A - POLL RESULTS

## POLL 1

Participants did not know the intent of the questionnaire, other than to find out what they thought about current P2P systems. Users were presented one question at a time, each shown on a new screen. The hyperlink for the questionnaire was posted in various online forums, and was answered by 140 anonymous participants. The questions as they appeared on the questionnaire and the results are presented below:

### Emotions

#### Question 1

In this question, you are required to indicate your anger level towards another peer. The network consists of you and only the three other people, and the amount shared by each is listed in the table below.

Peer	Shared Content (MB)
<i>You</i>	160
A	200
B	130
C	20

Considering the share amounts listed above, what is your anger levels towards Peer C? Please rate between 1 and 10, with 10 being the very angry, and 1 being not angry at all.

*Average result: 3*

#### Question 2

In this question, you are required to indicate your anger level towards another peer. The network consists of you and only the three other people, and the amount shared by each is listed in the table below.

Peer	Shared Content (MB)
<i>You</i>	50
A	30
B	70
C	25

Considering the share amounts listed above, what are your anger levels towards Peer C? Please rate between 1 and 10, with 10 being the very angry, and 1 being not angry at all.

*Average result: 2*

#### Question 3

In this question, you are required to indicate the anger level you would expect to receive from another peer. Please give an anger feeling level between 1 and 10, with 10 being the most intense, and 1 being none.

Peer	Shared Content (MB)
A	140
B	160
C	180
<i>You</i>	20

Considering the amounts above, and that all peers know the amounts, what would you expect the anger level from one of the peers to be, should you unexpectedly meet them?

*Average result: 5*

**Question 4**

In this question, you are required to indicate the anger level you would expect to receive from another peer. Please give an anger feeling level between 1 and 10, with 10 being the most intense, and 1 being none.

Peer	Shared Content (MB)
A	300
B	500
C	700
<i>You</i>	200

Considering the amounts above, and that all peers know the amounts, what would you expect the anger level from one of the peers to be, should you unexpectedly meet them?

*Average result: 3*

**Behaviour in File Sharing Communities**

**Question 5**

This question is about finding out what behaviours people dislike the most in a file sharing community. Listed are seven possible behaviours, but please list and rate others that aren't present in the comments box. Please rate on a scale of 1-10.

People who down multiple data streams simultaneously from the one person

*Average: 4.5*

People who download proportionally much more than they have shared

*Average: 4.2*

People who prematurely cancel other people's downloads

*Average: 7.7*

People who send obnoxious chat messages

*Average: 7.0*

People who misrepresent content they have shared (e.g. naming a file to indicate it is something it is not)

*Average: 9.0*

People who stay online only long enough for their own downloads to finish

*Average: 5.0*

People who have no content shared

*Average: 5.7*

## **Community Values**

### **Question 6**

Please rate on a scale of 1-10 (10 being the highest) the value you place on the following attributes of a peer-to-peer filesharing community. Rate the attribute '1' if it holds no value for you.

Like minded, interesting people to chat with

*Average: 3.4*

High content quantity

*Average: 8.4*

High-speed content download

*Average: 8.2*

Anonymity

*Average: 6.9*

High content quality

*Average: 8.8*

Large network size

*Average: 7.4*

Presence of authority figures who moderate environment

*Average: 3.5*

## **POLL 2**

20 anonymous users of P2P programs were asked this series of questions in an online form. Five participants were current users of Naanou. The questions as they were on the questionnaire and the aggregate results are below:

### **Antisocial Behaviour Online**

Have you ever experienced antisocial behaviour from another person while **online**?

<b>Value</b>	<b>Freq</b>
Never	0
V. Rarely	4
Rarely	9
Often	6
Very Often	2

If so, did that bother you?

Value	Freq
NA	0
Didn't bother me	9
Bothered me a little	7
Bothered me	4
Bothered me a lot	0

### Antisocial Behaviour in File Sharing

Have you ever experienced antisocial behaviour from another person in an online **file sharing** system?

Value	Freq
Never	4
V. Rarely	3
Rarely	5
Often	3
V.Often	5

If so, did that bother you?

Value	Freq
NA	4
Didn't bother me	5
Bothered me a little	1
Bothered me	6
Bothered me a lot	4

Would you consider leaving a file sharing system because of the level of antisocial behaviour present?

Value	Freq
No	8
Yes	9
Yes, have done so	3

### Moderation

If you observed a person behaving antisocially, and you had the opportunity to moderate that behaviour, would you do so?

Value	Freq
No	4
Yes	14
Depends on situation (please specify)	2

If, when you moderated, you 'punished' the person by a factor of 3, and you were affected by a factor of 1 by moderating, would you moderate antisocial behaviour you observed?

<b>Value</b>	<b>Freq</b>
No	6
Yes	8
Depends on situation (please specify)	5

If moderation was available, which of the following punishments do you think would be most effective?

<b>Value</b>	<b>Freq</b>
Slowing of downloads	4
Limiting of search results	1
Exclusion	8
Other	6

\* 1 subject did not specify

### **Values**

When you are using a P2P network, which of the following do you rate as most valuable?

<b>Value</b>	<b>Freq</b>
My time and effort	4
How much enjoyment I get	1
How many files I can download	12
Other, please specify	3

### **Naanou**

Please answer this section only if you have used Naanou, otherwise, skip to the end. On a 1-5 scale, with 5 being highest, how would you rate Naanou (compared to other systems you have used) on the following attributes? (average results are shown)

Overall: 3.375; Speed of searching: 3.28; Speed of downloading: 3.28; Usability: 3.57

## APPENDIX B – DEFINITIONS

### *Peer-to-Peer*

In pure P2P every node is as equal as every other node, and when several of these nodes meet, they form the P2P network. P2P has many applications, including distributed computing efforts[52], chat and the most popular – file sharing. P2P is characterised by being decentralised and distributed – two features that lend themselves to making P2P difficult to control. This is the antithesis of the older network computing model – client-server, examples of which are File Transfer Protocol (FTP), HyperText Transfer Protocol (HTTP) and Internet Relay Chat (IRC). Client-server architecture, while often high performance, has a dependence on the server's ability to be online and available when a client has a request, whereas with P2P if one peer is down it has little bearing on the operation of the network as a whole. Existing P2P systems have varying degrees of decentralisation and distribution. Older 'P2P' technologies such as Napster were essentially client-server, while newer systems such as Gnutella are completely decentralised.

### *File Transfer Protocol (FTP)*

FTP is an early network protocol for transferring files between two machines. Usually the user connects to a server with an FTP client, and either retrieves files or sends files. This normally requires a user to have an account on the server, although FTP can also operate in anonymous mode. FTP is defined by RFC 959.

### *Internet Relay Chat (IRC)*

A distributed text-based chat system. Clients connect to a server, which is often affiliated with a particular IRC network. Every server in the network is linked together so a message from a client on one server can be routed to a client on a different server within the same network. Users can send private message to each other or participate in *channels*, which is a form of group chat, often centred around a particular topic. IRC has a hierarchy of administration mechanisms available, such as operators '*ircops*' (operators of the server) and channel operators '*chanops*', which 'own' a particular channel. Due to the large numbers of users on networks and channels, control is often delegated out among

trustworthy participants to share the administrative burden. IRC was originally defined by RFC 1459.

***HyperText Transfer Protocol (HTTP)***

This pervasive protocol carries the World-Wide-Web, defined in RFC 2616. Due to its wide availability – and especially within firewalls - HTTP has more recently being used to carry all manner of protocols, and is the basis for a new remote procedure call mechanism – Simple Object Access Protocol (SOAP<sup>1</sup>).

---

<sup>1</sup> <http://www.w3.org/TR/SOAP/>

# **APPENDIX C – SOURCE CODE**

Please refer to the separately bound document for source code portions.

# APPENDIX D – PROJECT

## SCHEDULE

### SEMESTER 1

Week	Item
1	Research, system design
2	Research, system design
3	Research, preliminary coding, systems design
4	Research, Chord implementation: design, coding
5	Research, Chord implementation: design, coding
6	Research, systems design, Chord implementation coding
7	Annotated bibliography, systems design, Chord implementation coding
8	Chord implementation coding
9	Research, resource exchange design, coding
10	Research, resource exchange design, coding
11	Progress paper, coding, design
12	Seminar preparation, coding, design
13	Seminar, coding, design

### SEMESTER 2

Week	Item
1	Coding – Resource exchange
2	Coding – Resource exchange
3	Coding – UI
4	Coding – UI
5	Testing, fixing bugs
6	Testing, fixing bugs
7	Release publicly, fix bugs
8	Maintain release
9	Moderation coding & design
10	Thesis Writing, Moderation Coding, Questionnaire, Poster
11	Thesis Writing
12	Thesis Writing
13	Innovation Expo