TinyP2P: A Decentralized User-Friendly Peer-to-Peer File-Sharing System

Submitted in partial fulfilment
of the requirements of the degree of

Bachelor of Science (Honours)

of Rhodes University

Nichola Reid

Grahamstown, South Africa
November 2015

Abstract

Peer-to-Peer is said to be the next era of the internet, enabling the development of highly efficient and scalable distributed applications. File sharing is a popular online activity carried out by millions of users every day, but file sharing applications face several challenges to uptake, including availability, freeloading and legality issues. This paper explores these challenges, and investigates the viability of a purely decentralized application based on a Distributed Hash Table. We look at Human Computer Interaction, user interface goals and end user perspectives in order to achieve usability. We conclude that it is viable to develop a lightweight application to facilitate file sharing and communication in a user-friendly way, without the restriction of connecting to a central server or registering an account.

Acknowledgements

I would like to thank my supervisor, Mr Yusuf Motara, for his guidance and assistance throughout the project. I would like to thank him, in addition to Mr Karl van der Schyff and Mr Christopher Morley, for the loan of two virtual machines to assist with developing and testing the application.

Thank you to Caro Watkins, the Departmental Manager at the Hamilton Computer Science and Information Systems Department at Rhodes, for assisting me with distributing my survey as well as being helpful throughout my degree.

I would also like to thank my parents for their financial and emotional support

I would like to acknowledge the financial and technical support of Telkom SA, Tellabs, Easttel, Bright Ideas 39, THRIP and NRF SA (UID 75107) through the Telkom Centre of Excellence in the Department of Computer Science at Rhodes University.

ACM Computing Classification System Classification

Software and its engineering → Peer-to-peer architectures

Human-centered computing → User interface design

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The process of file sharing is an important and widely used activity in computing. The exchange of data between computer users has many uses, with the power to promote and simplify development through the sharing of work information between colleagues, or to facilitate the sharing of multimedia and art between friends.

Many different file sharing protocols exist, with varying features, styles of user interface and levels of security. Peer-to-Peer (P2P) is said to be the next era of the internet, with its decentralized and highly scalable nature allowing for the growth of powerful distributed networking applications (Aberer & Despotovic, 2001). Online file sharing is an activity carried out by millions of users daily. The process of transferring a file from one person to another is inherently scalable, but much research has been done in the areas of network configuration and file searching.

1.1 Research Questions

This project was conducted to investigate challenges to uptake of file sharing systems, and to determine whether a user-friendly application could be developed that addresses these challenges. It explores Human Computer Interaction and user interface goals, as well as end user perspectives, to show that such an application could be used by people in a wide spectrum of computer science and networking experience and ability.

1.2 Paper Overview

The remainder of the paper is structured as follows. Chapter 2 reviews the literature surrounding the core concepts of the project. It looks at file sharing and challenges to uptake, Peer-to-Peer, Distributed Hash Tables (DHT), end user perspectives, security, anonymity, and Network Address Translation.

Chapter 3 explains the approach taken to developing the application. Section 3.1 presents a survey about file sharing application features, conducted for comparison and to provide a more contemporary view of the one conducted twelve years previously. Section 3.2 explains the approach of decentralization in order to address challenges to uptake. The remainder of Chapter 3 describes the methodology and the approach taken in implementing the application and user interface.

Chapter 4 presents a summary of what has been achieved, outlines the challenges faced in development, and proposes future work that can be done to enrich the resulting application.

The result of this project, TinyP2P, is a lightweight file sharing application which is completely decentralized and uses IP Mnemonics to abstract and simplify peer connections, making setting up an ad-hoc file sharing network easy to learn and remember for novice and experienced computer network users alike.

Chapter 2

Literature Review

2.1 Introduction

The two main functions of a file sharing system are the lookup mechanism and the actual file download. While the transfer of a file from one user to another is inherently scalable, the file lookup function is notably more complex and problematic (Ratnasamy *et al.*, 2001).

This chapter discusses how file sharing can facilitate collaboration and learning, and gives several mechanisms to illustrate how file searching can be optimized. We identify and explore availability, freeloading and legality issues as significant challenges to the successful uptake of file sharing systems.

We review literature surrounding the history and prevalence of P2P, and present Grid computing as an application of P2P to illustrate its benefits and potential. We describe DHT, with several routing algorithms and extensions, as the most scalable and robust P2P architecture.

We then investigate the end user perspectives of file sharing applications, and explore various Human Computer Interaction (HCI) goals.

Finally, we discuss security, anonymity, and Network Address Translation.

2.2 File Sharing and Searching

A distributed file system aims for consistency and synchronization between files in different physical locations. A file sharing collaboration tool synchronizes files in a distributed file system to facilitate group work and resource sharing among knowledge workers (Lee, 2003).

Eisenstadt *et al.* (2003) discuss how the real-time online presence of peer-group members in a long distance learning environment could improve students' emotional well-being and academic performance. Their application, BuddySpace, uses online presence, geolocation information and file sharing to facilitate conversations and improved learning among student peers. BuddySpace requires groups to register and connect to a server and is thus not decentralized.

A file searching mechanism should be efficient and effective, returning results accurately and quickly with minimal communication. Locality-aware searching algorithms take into account geographical distances between peers within a large-area P2P network, and greatly improve response time and reduce network bandwidth usage (Kwon & Ryu, 2003).

Iamnitchi *et al.* (2004) define "small-world" file sharing communities as spatially concentrated clusters of peers within P2P networks who share similar interests, like scientific communities and peers with interest in similar files. They propose exploiting this phenomenon in user behaviour to create more efficient P2P systems.

Le Fessant *et al.* (2005) measured the eDonkey network and showed that a search mechanism could be significantly improved by taking into account interest-based locality. Peers who have similar content stored in their caches form clusters and send requests directly to each other before consulting the server.

Sripanidkulchai *et al.* (2003) demonstrated that interest-based locality shortcuts reduced many queries to single-hops, and that total load in the system was lowered by a factor of between 3 and 7 when tested on five different content distribution applications, including Kazaa and Gnutella.

2.3 Challenges to Uptake of a File Sharing System

This section describes three significant challenges to uptake faced by file sharing systems: availability, freeloading and legality.

2.3.1 Availability

One problem previously faced by the BitTorrent protocol was the requirement to download file location information (.torrent) files from a server, also known as a *Tracker*. .torrent files contain information about how the file is divided into chunks, as well as directions for the torrent client to locate and download these chunks. This problem of availability meant that if the Tracker hosting the .torrent files was unreachable, file sharing could not take place (Pouwelse *et al.*, 2004).

To address this centralized method of file location, the use of DHT (explored further in Section 2.5), Peer Exchange (PeX) and Magnet links were introduced. A *swarm* is the group of peers sharing a particular torrent, including *seeds*. Seeds are peers who have all the pieces of the file and are uploading pieces to other peers (BitTorrent, 2014a). DHT effectively makes all peers in a swarm act like a Tracker. PeX is a protocol that allows peers in a swarm to directly update each other without having to rely on a Tracker. Magnet links are Uniform Resource Identifiers (URIs) containing the hash code and location information of the required file (BitTorrent, 2014b).

Magnet URI's allow for users to get the download information of the requested file directly from other peers, rather than having to download a .torrent file from a server (Pouwelse *et al.*, 2004). They do, however, also allow for the facilitation of a number of other methods of sharing, including Trackers and direct web downloads.

Figure 2.1 shows an example of a Magnet URI. It is essentially a list of parameters. The Exact Topic (xt) is the key used to locate and verify the file. The Exact Length (xl) is the size in bytes and the Display Name (dn) is the file name. The Tracker (tr) is a URL to a Tracker to use BitTorrent without DHT, as some applications do not support it. The Exact Source (xs) is P2P address, for example a DirectConnect hub, and may possibly include the file hash. The Acceptable Source (as) links to a direct download from a webserver, intended for use in the case of failure to obtain the file via P2P. These parameters may each appear zero or more times, and can appear in any order (Mohr, 2002).

```
magnet:?&xt=urn:btih:QHQXPSDJU8OKDWKP47RRVIV7VOUXFE5Q
&xl=5242880
&dn=Bobby.McFerrin.-.Dont.Worry.Be.Happy.mp3
&tr=udp://tracker.openbittorrent.com:80%2Fannounce
&xs=http://192.0.2.27:6346/uri-res/N2R?urn:sha1:FINYVGBZTICF
&as=http://download.last.fm/Bobby.McFerrin.-
.Dont.Worry.Be.Happy.mp3
```

Figure 2.1: A hypothetical Magnet URI

As of 2012, one of the most popular and resilient torrent websites, The Pirate Bay[1], completely shut down its Trackers and now only uses Magnet links. This saved a large amount of bandwidth and server space. Additionally, Magnet links are more difficult to block than .torrent downloads (Brian, 2012).

There is a cost associated with the increased decentralization of the BitTorrent protocol. *Poisoning* and *pollution* refer to the deliberate or accidental injection of fake or corrupted files into a file sharing system. Poisoning and pollution decrease the integrity of the files within a system (Christin *et al.*, 2005). Pouwelse *et al.* (2004) showed that the global components (for example, volunteer moderators) of BitTorrent lead to a high level of integrity for both content and meta-data, solving the problems of poisoning and pollution at the cost of system availability. They exhibit that decentralizing the BitTorrent protocol would solve the problem of availability but would not be able to ensure integrity of files, and so such a system could potentially be flooded with fake files.

2.3.2 Freeloading

An analysis of the Gnutella network showed that 70% of system users were "freeloaders" who downloaded from the P2P resource pool but did not upload or share any original content (Feldman & Chuang, 2005). Freeloading (or "leeching") is a significant challenge facing P2P file sharing systems.

A survey of end user perspectives on file sharing system features conducted by Lee (2003) (discussed further in Section 2.6.1) suggested that anti-freeloading mechanisms, such as upload requirements or point-based reward systems, would be detrimental to the overall

[1] https://thepiratebay.gd. (October 2015).

desirability of a system. An anti-freeloading mechanism could limit the user base and in turn the content base, which would greatly diminish the success of the system.

In keeping with the *laissez-faire* nature of P2P networking - that is without user fees, priority user groups or restrictions on use - P2P remains a powerful, accessible and democratic infrastructure. The requirement of a minimum number of shared files, coupled with the need for a central server, are reasons why well-known file sharing applications such as DirectConnect have relatively low popularity among similar systems (Pouwelse *et al.*, 2004).

A file sharing system may depend on popular content being available in order to sustain its user base. Without such content, it would tend to lose some of its user base; this, in turn, may cause the volume of popular files to continue to decrease, causing a downward spiral in the number of users as users lose interest and pull out. However, freeloaders can still provide common-service capacity while not contributing to file-serving capacity (Ge *et al.*, 2003). The load-balancing nature of DHT (see Section 2.5) makes use of the resources of even those users not contributing to the file base. Every node in the P2P network assists in the maintenance of the hash table and the routing of messages, and this impacts on scalability and performance (Feldman & Chuang, 2005).

The BitTorrent protocol addresses the problem of freeloading by making a node spend more of its upstream bandwidth on peers who provide it with a high downstream bandwidth. This means that seeds will receive faster downloads than leechers. The increased download speed serves as an incentive to seed, but since leeching can still take place with a sacrificed download speed, it is not too strong a disincentive as to cause these users to not use the system (Crosby & Wallach, 2007).

2.3.3 Legality

Online piracy is a significant hindrance to the commercial and legal use of P2P file sharing. The largest source of intellectual property violations within a P2P network is collusive piracy, in which colluders (paid clients) share copyrighted material with pirates (unpaid clients). (Lou & Hwang, 2009).

Lou & Hwang (2009) developed a content-poisoning mechanism to identify and target pirates and colluders to prevent illegal file sharing in systems like Gnutella, Kazaa and eDonkey. They found that their scheme proved less effective in the poison-resilient BitTorrent protocol. Dhungel *et al.* (2007) analyzed the resilience of the four main components

of BitTorrent, namely leechers, seeds, peer discovery and torrent discovery. They found that attempts to attack each of the four components were not significantly effective, in worst cases merely extending download time by 50%.

The resilient nature of the BitTorrent protocol means that illegal activity is difficult to prevent, while the system is still increasingly being used to distribute legal, commercial content (Blackburn & Christensen, 2009).

DirectConnect (also known as DC or DC++) is a free, open-source file sharing application that allows users to connect to one another via a central hub. Users can connect to a public hub that is published in a list viewable in-client, or to a private hub by entering its IP address or domain name. The problem occurs when the hub goes down and users no longer have a means by which to connect. Rhodes University once had a highly popular DC network, but IT administration stopped making the campus network available for the use of DC due to the number of students who were sharing unlicensed or otherwise inappropriate material (Barker, 2015). University and network administrators do not want to bear the legal and reputational consequences of the actions of the DC network users. Historically, piracy and high bandwidth usage led to many Universities and ISPs throttling or banning the Kazaa system (Wallach, 2003).

2.4 Peer-To-Peer

P2P is an IT architecture and a design philosophy that emphasizes decentralization (Schoder & Fischbach, 2003). Each node in a P2P network installs a single package which incorporates client-, server- and router-type code, so that each peer is able to issue queries, serve requests and perform routing (Taylor *et al.*, 2003). P2P is rapidly evolving the way networks are designed, as decentralization and resource pooling are increasingly being exploited to promote efficiency and scalability in networking applications (Foster & Iamnitchi, 2003). Le Fessant *et al.* (2005) named P2P traffic as the main component of internet traffic.

Kaune *et al.* (2008) estimated the fraction of P2P traffic within the internet to be at least 50%. Since most P2P applications now allow for the use of random ports, as well as increased usage of Virtual Private Networks (VPN), P2P traffic is difficult to detect and the actual figure is likely to be much higher while the measurable statistic appears to drop over time (Sen *et al.*, 2004).

P2P traffic detection allows Internet Service Providers (ISPs) to provide a degraded service or throttled bandwidth to reduce or prevent this kind of traffic. Detection would also enable copyright violators to be discovered and identified, and so it is sometimes in the best interest of the P2P user for traffic to be obfuscated (Spognardi *et al.*, 2005).

Karagiannis *et al.* (2004) measured the traffic of popular P2P protocols[2], and reverse-engineered them to ascertain payload characteristics, finding that any change in the proportion of P2P file sharing traffic was an increase, despite reported statistics.

One of the most significant advantages of a P2P system is its abstraction and encapsulation of client, server and router roles. P2P creates an overlay, or virtual network comprised of logical links, built on top of but not structurally related to the underlying physical network. P2P overlay networks are able to form and change dynamically and spontaneously (Vu *et al.*, 2009, p. 5). As functionality for client, server and router are contained in each peer node's implementation, P2P removes the need for a dedicated server (Ge *et al.*, 2003). A dedicated server is a single point of failure, and may require significant storage space. In distributed P2P systems, storage and overhead processing are divided between peers, making them highly scalable.

Grid computing is a form of parallel computing that allows for geographically dispersed computers to contribute their unused processing and memory resources to others, making use of heterogeneous resource capacities in various machines (Kaur & Rai, 2014). SETI@home (Searching for ExtraTerrestrial Intelligence at home)[3] is a project that allows volunteers to donate their unused processing power to analyze radio telescope data in the hope of discovering intelligent life in space (Anderson *et al.*, 2002).

Ganguly *et al.* (2006) propose enabling self-configuring virtual IP networks - similar to the internet - built on P2P overlays, allowing seamless access to Grid resources spanning multiple domains which are aggregated into a virtual IP network, isolated from the underlying physical network. Their technique, IPOP (IP over P2P,) utilizes P2P's self-configuring, scalable and fault-tolerant nature to achieve overlay routing without centralized administrative control and with acceptable latency overhead.

Hunt (2014) recently showed that it is viable to replicate social networking platforms like Facebook and Twitter in a P2P overlay using BitTorrent Sync, providing a scalable alternative to established client-server architectures with significantly decreased overhead costs.

[2]Namely eDonkey2000, FastTrack, BitTorrent, Gnutella, MP2P and DirectConnect.
[3]http://setiathome.ssl.berkeley.edu/

Ge *et al.* (2003) divide P2P architectures into 3 categories. The first, Centralized Indexing Architecture (CIA) depends on a central server to coordinate node participation and maintain the index of available files, and is typified by the Napster network. In the second, Distributed Indexing with Flooding Architecture (DIFA), each peer is responsible for the indices of only its own shared files. A query is flooded through the network, but their limited scope means that some queries may never be satisfied even if the requested files are available in the system. This type of system is typified by the Gnutella network. Finally, the Distributed Indexing with Hashing Architecture (DIHA) is typified by systems like BitTorrent which use DHT (explored in Section 2.5). Ge *et al.* (2003) conclude that the DIHA architecture scales better than CIA and DIFA, as it is not constrained by a central server and there are no query failures.

Ge *et al.* (2003) adjust their model to account for freeloaders, concluding that in DIFA, a high ratio of freeloaders significantly degrades performance, as the success probability of a query decreases. They note, however, that CIA and DIHA can support high ratios of freeloaders, taking advantage of their spare computing resources.

2.5 Distributed Hash Tables

A hash table is a data structure that maps keys to values in an associative array, using a hashing function to compute the indices where values in the table can be found in order to provide fast lookups. A DHT is a system that divides a hash table over a number of distributed nodes which are interconnected in a structured overlay network, so that each node stores a fraction of the entire hash table (Kurose & Ross, 2013, p. 177).

In DHT systems, files are associated with a key (produced, for example, by hashing the file name) and each node stores a certain range of keys. A lookup of the key returns the location of the node that currently stores the requested file. Routing algorithms vary among implementations and the scalability of a DHT is dependent on the efficiency of its routing algorithm (Ratnasamy *et al.*, 2001).

Section 2.5.1 describes the four original DHT protocols, which comprise the second generation of P2P: CHORD, CAN, Pastry and Tapestry. Section 2.5.2 examines Kademlia, the advanced DHT chosen to base this project on. These five protocols were inspired by systems like FreeNet and Gnutella, however, they differ in that they guarantee that queries will always be satisfied, while remaining scalable and self-organizing (Rowstron & Druschel, 2001). Section 2.5.3 lists several proposed extensions to improve upon DHT.

2.5.1 CHORD, CAN, Pastry and Tapestry

The CHORD protocol maps a key to a node using consistent hashing, which balances load, as nodes are probablistically assigned roughly the same number of keys. This also ensures that relatively little reorganization is needed when nodes leave or enter the network. Decentralization and load balancing make CHORD robust and suitable for large-scale P2P networks even in the face of concurrent node arrivals and departures (Stoica *et al.*, 2001).

Content-Addressable Network (CAN) uses a virtual multidimensional Cartesian coordinate space, dynamically partitioned among participating nodes in order to perform optimized routing. A node will discover and store the IP and virtual coordinate zone of all of its immediate neighbours. Routing is performed by following a straight line path through the Cartesian coordinate space, from the source to the destination. Each CAN message contains its destination coordinates, and so a message is routed by nodes forwarding it to the neighbour with the closest coordinates to that of the message destination (Ratnasamy *et al.*, 2001).

Figure 2.2 exhibits how routing is performed from Node 1 to coordinate (x, y) in the virtual coordinate space. In this example, Node 1 knows about the neighbouring nodes 2, 3, 4 and 5 (Ratnasamy *et al.*, 2001).

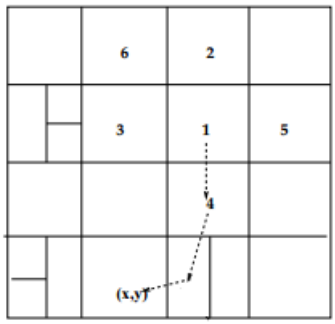


Figure 2.2: An example message being routed from Node 1 to coordinate (x, y) (Ratnasamy et al., 2001)

Pastry performs efficient routing of messages through locality-aware forwarding in wide-area P2P systems, with an expected number of $O(logN)$ hops in a network with n nodes. Pastry exploits locality by calculating distance between nodes using their IP addresses (Rowstron & Druschel, 2001). Tapestry is very similar to Pastry, but provides better

probabilistic bounds on routing distances and guarantees that an existing object is always reachable (Zhao *et al.*, 2001).

2.5.2 Kademlia

Kademlia is advanced in that it provides provable consistency, performance and minimal latency guarantees. It minimizes the number of configuration messages routed between nodes using parallel asynchronous queries to automatically spread configuration messages as a side effect of key lookups, thus avoiding time-out delays from failed nodes. The node IDs and keys are both 160-bit numbers. When a key-value pair is stored in the DHT, it is stored at the node that has an ID arithmetically closest to the key (Maymounkov & Mazieres, 2002).

Kademlia uses an Exclusive Or (XOR) metric on two peers' IDs to calculate the "distance" between them in order to perform routing (Maymounkov & Mazieres, 2002). This is a virtual distance calculated to structure the overlay network, and has no correlation to the physical proximity of nodes. XOR is a logical operation that evaluates two conditions and yields true if one and only one of them is true (Germundsson & Weisstein, 2002). The XOR operation is symmetric, meaning that the result can be obtained by XORing both keys, but either key can also be derived by XORing the result and the other key.

Table 2.1 uses simple four-bit numbers to illustrate that XOR decryption can be performed using either both keys, or one key and the result. Following the logic shown in the Table, performing the operation with Key_2 and the result would predictably yield Key_1.

Table 2.1: Symmetric XOR Operation to Obtain a Result or a Key

XOR on Two Keys			
Key_1	Key_2	=	Result
1	0		1
1	0		1
0	1		1
1	1		0

XOR on a Key and Result			
Key_1	Result	=	Key_2
1	1		0
1	1		0
0	1		1
1	0		1

In this way, Kademlia can determine the distance between two peers using both of their IDs, or determine the ID of a particular node using the ID of a given node and the distance between the two.

Kademlia treats nodes in a network like leaves in a binary tree. Every message sent by a node contains its node ID, and each node knows about at least one other node in its sub-tree. Lookup requests are sent to the closest node in the requestor's sub-tree, which in turn finds the closest node in its own sub-tree. The resulting node ID is sent back to the requestor, and the process continues recursively. Kademlia provably achieves a searching time that is always less than $log_2 n$ in a network with n nodes (Maymounkov & Mazieres, 2002).

Figure 2.3 shows how a node with the prefix 0011 finds the node with the prefix 1110 by contacting successively closer nodes until lookups finally converge at the target.

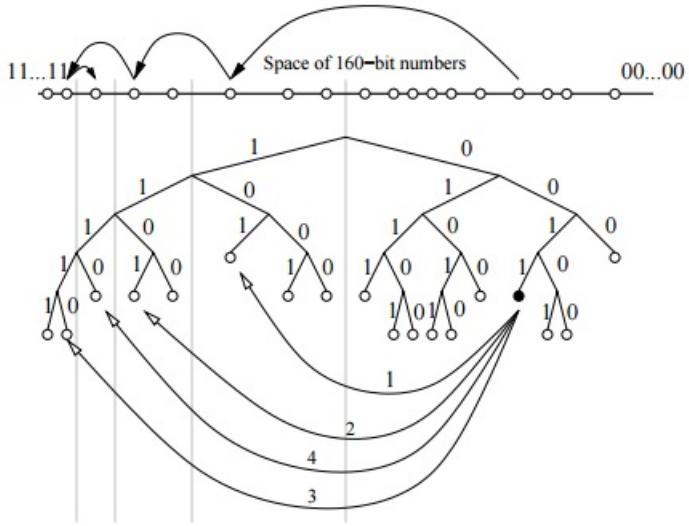


Figure 2.3: Locating a Node by its ID in a Kademlia Binary Tree (Maymounkov and Mazieres, 2002)

The dynamic joining and leaving of peers in a network is referred to as *churn*. CHORD, CAN, Pastry, Tapestry and Kademlia have demonstrated that in networks with moderate and even significant churn, DHT is highly scalable and is even capable of achieving one- or two- hop lookups with a modest sacrifice of bandwidth (Feldman & Chuang, 2005).

BitTorrent uses a Kademlia-based protocol for its DHT, essentially making each peer act like a Tracker (Loewenstern & Norberg, 2013).

In order for a lookup to be satisfied, the hashes must be exactly the same which means that the search term must exactly equal the file name. Kad is a file sharing system built on Kademlia, which facilitates keyword searches by tokenizing and hashing all the words

in a file name and uploading them to the DHT with the full file name and source as the value (Steiner *et al.*, 2007). This means that the file name "this.is-a TEST.txt" would be returned for a search of each of the words "this," "is," "a," "test" and "txt."

2.5.3 Extensions to DHT

A Hierarchical System is a partially centralized DHT in that it assigns certain nodes the role of *Supernode*, which acts as a proxy for multiple other nodes, known as *Leafnodes*. The Supernodes are in a DHT formation with each other, while the Leafnodes communicate with the rest of the network via their Supernode. Figure 2.4 shows a hierarchical P2P overlay network with Supernodes forming a Chord ring, and Leafnodes attached to them.

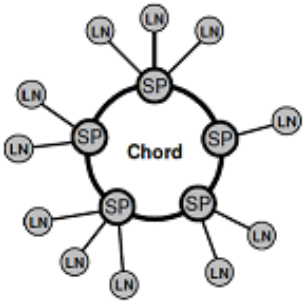


Figure 2.4: A hierarchical overlay network with Supernodes forming a Chord ring, with leafnodes (Zoels et al., 2006)

Zoels *et al.* (2006) showed that hierarchical systems decrease network costs in an environment where peers have heterogeneous resources, however the fact that centralization has the potential to overload peers means that system stability is compromised.

Kwon & Ryu (2003) propose an Asymmetric Hierarchical system to account for the varying resource capacities of different machines on the network. The P2P Asymmetric file Sharing System (PASS) assigns nodes with high available computing power the role of Supernode, which takes responsibility for all routing within the network. The file location information file is only replicated over three nodes, meaning that a file search can be completed within a maximum, constant number of hops which is independent of the number of participating nodes. The maximum number of search hops is four, and their system takes advantage of locality by first searching in the area where the searching node is, branching out to other areas if the local lookup fails.

PASS requires some memory to store routing tables at each Supernode (estimated to be at least 20mb in a network of 100 Supernodes, each owning 100 Leafnodes), but imposes less

computational overhead than DHT implementations where the routing table is distributed over all nodes. Storing the directory files in central Supernodes diverges somewhat from the server-less nature of P2P, but the advantage of this architecture is that discovery time is greatly reduced while there is still not a single point of failure. When a Supernode goes down, the network self-configures and creates another one. Kazaa and Morpheus are two proprietary systems that use this partially decentralized method of assigning Supernodes (Kwon & Ryu, 2003).

Wang *et al.* (2010) propose eKademlia, an extension of Kademlia which determines nodes' physical proximity to each other, and uses this locality information to optimally assign node IDs when forming the logical topology in order to improve lookup performance.

Aiello *et al.* (2008) propose Likir (Layered Identity-based Kademlia-like InfRastructure), which prevents routing and storage attacks by utilizing a certification service to provide a secure identity-based communication protocol. All content put to the DHT is associated with the owner's identity in a way that is not repudiable.

The efficiency of the protocols described in Section 2.5 demonstrate that DHT can be implemented in such a way that is scalable, robust, load-balancing, dynamically self-organizing, fault tolerant and extensible.

2.6 End User Perspectives

2.6.1 User Interface and Features

User Interface (UI) designers should be aware of the demographics of their intended users, taking into account their level of experience, environment, task characteristics and abilities. A *multi-modal* interface is one that adapts to different users and different contexts (Reeves *et al.*, 2004).

The survey of end user perspectives of P2P file sharing system requirements conducted by Lee (2003) asked what end-users felt were necessary or desirable features in a file sharing system. The result was a list of features ranked numerically from most to least important (see Table 2.2.) They indicated that the most important components are that it is free to use, is fast, stable and reliable, and can resume down-/uploading after application

Table 2.2: User Rated Features Of A File Sharing System (Lee, 2003)

	Features	Average Importance Score (1-7 increasing scale)
1	Charges no fee	6.50
2	Is fast	6.38
3	Is stable	6.34
4	Is reliable	6.33
5	Can resume loading	6.05
6	Has large file selection	5.78
7	Can exit nicely	5.75
8	Has large user base	5.56
9	Has good search features	5.53
10	Gives error messages	5.32
11	Can organize file as library	5.28
12	Can control spam	5.16
13	Provides server information	5.04
14	Can turn off ad	5.01
15	Has good security features	4.93
16	Supports passive search	4.36
17	Supports direct messaging	4.14
18	Can filter content	4.10
19	Support buddy list	3.99
20	Is open source	3.83
21	Can credit contributors	3.82
22	Has colourful interface	3.57
23	Supports chat	3.46
24	Has points for uploading	3.44
25	Supports only legal files	3.14
26	Has voice connection	2.93

restart. The importance of certain features differed significantly between experienced and less-experienced users, with features like chat support, buddy list support, passive search and a colourful interface proving more important to less experienced users. This suggests that these features are useful when learning and remembering how to use the system.

2.6.2 Human Computer Interaction

The quantitative nature of the requirements that UI design poses is critical and challenging. Human Computer Interaction (HCI) is described as the intersection of Computer Science with behavioural sciences, design and media studies among many other fields (Carlisle, 1976). It is involved in the planning, designing and studying of interfaces and interactions between computers and end users.

Rogers *et al.* (2011, p. 22) identify 6 main goals for a good user interface:

1. Effectiveness
2. Efficiency
3. Safety
4. Utility
5. Learnability
6. Memorability

The *effectiveness* and *efficiency* refer to the ability and degree to which the system aides the user in achieving their goals. A safe interface is so laid out that unsafe activities can't accidentally be carried out (see Section 2.7.1). *Utility* refers to overall satisfaction gained by the user from using the system. *Learnability* and *memorability* refer to the ease in which a user can master and remember how to use the application.

Brooke (1996) suggests that the usability of an system refers to its *appropriateness to use*. The usability of a system is apparent to and evaluated by the user long before they have committed to it, and so this property should be evident at the outset (Lee, 2003).

Evaluating appropriateness for use requires us to examine the *intended use* of the system, while evaluating ease of learning and remembering requires the examination of the *intended user base*.

Jef Raskin, Macintosh founder and HCI expert, opposes the use of the word "intuitive" with regards to HCI design, saying that something is only truly intuitive if it can be done without any training or rational thought, and is identical to something that the user has experience with. Raskin proposes that the word "intuitive" be replaced with "familiar" within HCI literature, and that intuitiveness might be the worst quality one could ask for in an interface. Interface designers are required to produce iteratively "better" interfaces; however with the expectation of increased intuitiveness and the restriction of an interface that is familiar to what the user is used to, potential improvements and modifications are limited and sometimes rejected outright. Raskin argues that the word "familiar" in place of "intuitive" will illustrate the boundaries that this requirement places on interface design and improvements, and give reasons for decision makers to be more flexible and open to new ideas (Raskin, 1994). Familiarity of a new interface may lead to a minimal initial learning time, but foregoing this and allowing for an initial learning phase for a product that is not entirely "intuitive" may lead to greater productivity and improved software after that time.

2.6.3 Sharing modality

One challenge facing the user interface design of a file sharing system is that users typically want different levels of sharing modality for different types of files and peers. For example, users may want certain files to be publicly available for any peer to download, or they may perhaps have a set list of friends with whom they would like to share their photos, or a single peer with which they want to share particular files.

Voida *et al.* (2006) address this by introducing a user interface component called a *sharing palette*, which incorporates aspects of push-orientated and pull-orientated sharing. Push-orientated sharing requires effort on the part of the sender, while pull-orientated sharing requires the receiver to search for and request the file.

The sharing palette uses different colours to identify different semantic groups with which files are shared, allowing the user to quickly and easily specify visibility and permissions for files without the need to maintain an access control list (Voida *et al.*, 2006).

2.7 Security

Milojicic *et al.* (2002) note that P2P exposes more security threats than client-server and

centralized models, and that some level of trust is necessary among peers.

Saroiu *et al.* (2001) showed that peers tend to misrepresent information if there is incentive in doing so. Processes such as the delegation of Supernode status requires for the system to either have an incentive for truthful information, or to be able to directly measure or verify the information regarding peers' resource capacities.

Wallach (2003) explores several methods to increase fairness, trust and security in P2P applications, namely cryptography, redundant routing, and economic methods. Secure routing can be provided via self-certifying data. Node ID attacks could be prevented using node certification models, however, in P2P systems where nodes are able to dynamically change their IP address these attacks may be unavoidable. These techniques, developed by Castro *et al.* (2002) allow a tolerance of up to 25% of malicious nodes maintaining good service and a minimized number of compromised nodes.

Wallach (2003) explained how malicious nodes in a Pastry network can virtually censor a certain document from the rest of the network, by cooperatively choosing node IDs that are clustered near the document's key. He notes that the ability to eject a malicious node from a P2P overlay is provable at the application layer, however it is not clear how such a process is provable at the routing layer.

Any kind of password based security in a DHT system would be ineffective if it was implemented at the application level, as the underlying DHT can still be exploited. It is thus necessary for this kind of security to be implemented at the DHT level (Bocek, 2015b).

P2P has the ability to resist Denial of Service attacks, as peers are aware of each others' existence and the network can recover even if a number of nodes have come under attack, by re-configuring itself around these "holes" (Maymounkov & Mazieres, 2002).

2.7.1 A Secure User Interface

Good & Krekelberg (2003) conducted a study on the usability and privacy of the Kazaa network, revealing that the ambiguity of the user interface commonly leads to users carrying out unsecure, undesirable actions such as sharing personal data. Their study shows that many users of Kazaa inadvertently share personal files like their emails, credit card information and tax reports. It also revealed that other users, whether malicious or

misinformed, take advantage of these mistakes, as dummy files with such names as "creditcards.xls" and "inbox.dbx" received download requests.

Good & Krekelberg (2003) outline four properties of a safe and usable P2P file sharing system:

1. Users are clearly made aware of what files are being offered for others to download
2. Users are able to determine how to successfully share and stop sharing files
3. Users do not make dangerous errors that can lead to unintentionally sharing private files, and
4. Users are sufficiently comfortable with what is being shared with others and confident that the system is handling this correctly

Only two out of a surveyed 12 users could accurately identify which files on their system were being shared. Good & Krekelberg (2003) conclude that the Kazaa interface makes too many assumptions regarding the users' knowledge of file sharing, and violates all four above-mentioned guidelines. They suggest that usability and security be of top priority when designing a file sharing application.

2.8 Anonymity

An IP address can identify users and their locations. Significant amount of research has been done in the area of *initiator* anonymity in P2P networks. This involves the spoofing of the IP address of the peer sending a request, by routing the request through various other participating peers so that the originating IP address is hidden. Scarlata *et al.* (2001) propose further measures to provide *responder* anonymity, building on existing initiator anonymity protocols using multi-cast routing and solving the problem of anonymity degradation over time.

Users in a purely serverless P2P system are required share their IP addresses to facilitate connection. This first step of connection requires some back-channel through which to communicate the address, which could potentially be insecure.

Users within a P2P network can see the IP addresses of other peers, however their identities and activities are hidden from the world outside of the overlay network in which

they are connected, as this traffic is difficult to detect (Sen *et al.*, 2004). A Friend-to-Friend network is one that consists of users who trust each other. This kind of network guarantees privacy, dependability and uncensorability by taking advantage of social trust (Sharma *et al.*, 2011).

2.9 Network Address Translation

Network Address Translation (NAT) is useful in small office and home communities, and is a means by which all users on a subnet share one globally routable IP address. To address the shortage of IPv4 addresses for each internet-capable device in the world, many routers are NAT routers which provide users with a unique private IP address. This poses some problems for certain protocols, specifically P2P, where multiple users may need to connect across NATs (Hu, 2005).

Universal Plug and Play (UPNP) and NAT Port Mapping Protocol (NATPMP) are protocols which allow an application to configure the NAT router or gateway in order to create port mappings which let the router know which packet is intended for which user in the network (Cheshire & Krochmal, 2013). Relaying allows for peers behind NATs to communicate with each other via another "relay" peer who is not constrained by a NAT.

Chapter 3

Usability Survey and the TinyP2P Application

The following section describes the approach taken in developing the application. Section 3.1 presents a survey carried out to rank the desirability of various features. Section 3.2 explains how decentralization can be applied to overcome the challenges to uptake identified in Section 2.3. Section 3.3 explains how Mnemonics are used to simplify IP connections. Section 3.4 describes the libraries used to develop the application, and Section 3.5 explains how the application functions. Finally, Section 3.6 discusses the security aspects of the application.

3.1 Usability Survey

To assist with this research, a survey was conducted to determine users' perspectives on file sharing systems. An online survey was distributed publicly on social media platforms as well as in an e-mail sent to Computer Science and Information Systems undergraduate students at Rhodes University. Ethics clearance to conduct the survey was granted by the Rhodes University Ethics Committee. A copy of the survey is included as Appendix A.

All participants were over the age of 18. Of the 96 participants, 22 have no formal academic background in Computer Science, 8 had studied IT at a school level and 66 had studied or were studying Computer Science or Information Systems at university. Participants who studied at both school and University were grouped into the latter group.

Participants were asked to select features that they thought were most important in a file sharing system. This question is a modified version of the study conducted by Lee in 2003, aiming to complement it and provide a more contemporary view. Speed, stability and reliability were grouped into one feature for the current study, as they appear to be highly interrelated. Chat, buddy lists and direct messaging were also combined to form one feature. Several of the features included in Lee's study were not included in the recent one, in order to make the survey shorter and less inundating for the participants.

The results are shown graphically in Figure 3.1. Participants were asked to choose as many features as they thought were of utmost importance in a file sharing system, without which they would not make use of such a system. A table of the results can be seen in Appendix B.

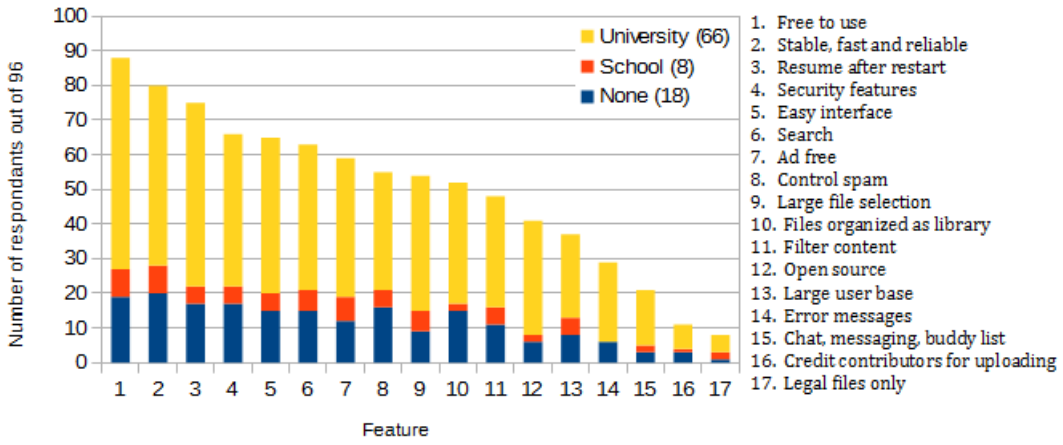


Figure 3.1: Number of votes for each feature by 96 participants of varied education level

The most important features align with those of Lee's study, namely that a system should be free to use, fast, stable, reliable, and that it can resume downloading after restart. Good security features ranked higher in the recent survey than in Lee's, while a large user base ranked lower. Methods to reduce freeloading, such as crediting contributors or awarding points for uploading, again received a low rating while support for only legal files ranked lowest.

When asked whether they participated in the downloading or sharing of illegal material, 76% (73 out of 96) of the participants responded that they did.

3.2 Decentralization to Overcome Challenges

TinyP2P creates ad-hoc Friend-to-Friend (F2F) networks called *TinyNets*. An ad-hoc network as one that is created on-demand and for a specific purpose. A F2F network is a type of P2P network in which the users know or trust each other. F2F networks consist of only the participating peers and are essentially non-existent to anyone outside of the network.

TinyP2P uses decentralization to try to overcome the three major challenges to successful uptake discussed in Section 2.3.

The first challenge is availability, and the need for some level of central coordination to facilitate peer connection and file searching. TinyP2P exploits decentralization to avoid the need for a central server by using IP addresses or Mnemonics (see Section 3.3) to facilitate direct connection of peers in a DHT. TinyNets are created when needed and destroyed when all participating peers have disconnected.

TinyP2P reduces the detrimental effects that can be caused by freeloaders, as the network doesn't serve as a persistent public repository, and each peer connected to a TinyNet most likely has a reason to be there. The scalability of the chosen DHT, as well as the purpose-specific nature of the networks created, mean that freeloading is not detrimental. All peers, even those who don't share files, assist in routing and topology maintenance, and are unlikely to detract utility from the other, contributing members.

Legality issues introduce a disparity between the interests of the server hosts and the clients. This means that certain systems, such as Kazaa and DC, are not a viable option for network users like students on a University campus, as discussed in Section 2.3.3. TinyP2P aims to overcome the problem of host-client interest disparity with its purely decentralized manner of setting up ad-hoc interest-orientated sharing networks. The traffic within the overlay network is difficult to detect to people who not connected to it, and the consequences incurred by sharing illegal material falls on the users.

3.3 IP Mnemonics

A TinyNet relies solely on the participating peers, and so a node needs to know the IP address of any node that is currently connected to a TinyNet in order to join. TinyP2P simplifies the process of communicating IP addresses by using a system of Mnemonics.

Mnemonics make learning and remembering easier by mapping short and familiar phrases to longer, more complex objects (Oxford Dictionaries Online, 2015). The system of IP Mnemonics maps a three-letter English word to each of the numbers from 0 to 255, making a unique four-word, four-syllable phrase translation of any IP address. The list is a modified version of the one found at `http://gurno.com/adam/mne/`. Although two-letter words like "ni" and "oz" might be shorter and still allow a mapping of the entire IPv4 address space, three-letter words like "cat" and "zip" are English words which are easy to pronounce and may aid the user in remembering the phrase. For example, the phrase "rat out tip key" might be easier to verbalize and visualize than "nz if do to" which is in turn easier than "192.168.231.123". The goal is that the phonetic key phrase be more natural to relay to peers than its numerical counterpart.

In the TinyP2P interface, a user may enter the Mnemonic or literal IP address of the peer that they want to connect to. The goal of the Mnemonic system is to simplify the process of communicating IP addresses, but we avoid appearing condescending to the user who would prefer to use an IP address. The Mnemonic system is simplistic, but mimics the decentralized method of sharing a short string in order to facilitate connection and sharing as implemented in Magnet URIs.

The current Mnemonic system works for all addresses within the IPv4 address space, but will not be applicable with the introduction of IPv6 addresses. A larger mapping system will need to be implemented to handle these.

3.4 Libraries

Hive2Hive[1] is an open-source Java library that supports secure, distributed P2P file synchronization and sharing, but is largely unfinished. Started in 2013 by post-graduate students at the University of Zurich, Hive2Hive grew rapidly and seemed very promising. However, due to the fact that most of the core developers have now graduated, the project has seen no progress since March 2015.

TinyP2P was initially going to be built on top of Hive2Hive, creating a user-friendly graphical interface making use of Hive2Hive's network management, user management, file management and file versioning functionality. However, given the time frame in which TinyP2P was developed, and the under-developed state of the Hive2Hive library, it became

[1] http://hive2hive.com/

apparent that basing the project on Hive2Hive would not be feasible. Instead, TinyP2P uses Hive2Hive's functionality for network management and has its own methods for user management and file management.

Further development of Hive2Hive will rely on pull requests from interested parties, and if this ever happens then TinyP2P can be extended to make use of the secure file synchronization and versioning tools offered by Hive2Hive.

TomP2P, also created by academics from the University of Zurich, is an extended DHT Java library which allows for multiple values to be stored for a key. It uses the Java NIO communication framework to handle concurrent connections, and uses the Kademlia routing algorithm (Bocek, 2015a). TinyP2P uses TomP2P version 5.0-Beta6.

TomP2P is under development and regularly being improved upon and extended. The incorporation of exploiting heterogeneous resources, locality awareness and certification are future possibilities for TomP2P and thus TinyP2P itself.

The GUI of TinyP2P was built using the Java Swing Toolkit and Netbeans IDE.

3.5 TinyP2P

TinyP2P takes up less than 6.5MB on disk, and requires no installation. As it runs on the Java Virtual Machine (JVM), it requires the Java Runtime Environment (JRE). It is lightweight and portable, and can be run from a flash drive. It provides an interface for simple distributed file searching, sharing, directory browsing and chat.

The cat icon in the top right corner of each window opens the Help window (see Screenshot 5), which displays the user's Mnemonics and IP Addresses, as well as a link to the User Manual which is included in Appendix C.

3.5.1 Putting and Getting from the DHT

Some technical clarification between the terms *peer* and *node* is needed. The *peer* is the client/server module that implements the application layer protocol and listens on a TCP port. The *node* is the client/server module that implements the DHT protocol, and listens on a UDP port (BitTorrent, 2014a). The DHT is made up of the nodes, which keep the

location information of the peers. A node can be thought of as the device on which the application is running, and the peer as an instance of the application.

Keys and peer IDs in the TomP2P DHT are both 160-bit numbers, a type called `Number160`. A `Number160` is obtained by hashing a string using the function `createHash()`. When data is put to the DHT, it is stored at the node whose peer ID is numerically closest to the key.

Results from asynchronous computations, like distributed `put` and `get` operations, are represented by a `Future`. `Futures` have methods to check if a computation is finished, to wait for completion, and to retrieve results of the computation (Java Documentation, 2014). `FuturePut` and `FutureGet` handle putting and getting within the non-blocking framework. This means that multiple `put` and `get` operations can be performed concurrently by different peers.

To create a key from the string "`my key`," the following code is used:

```
Number160 key = Number160.createHash("my key");
```

To put data to the DHT, for example the integer `value`, the following code is used:

```
int value = 1337;

FuturePut futurePut = node.getPeer().put(key).data(new Data(value)).start();

futurePut.awaitUninterruptibly();
```

The function `getPeer()` retrieves all the peer information relating to the `node` in the DHT. The code `put(key).data(new Data(value))` serializes the data and inserts the key-value pair into the DHT. This command is executed as a thread.

To get data from the DHT, the following code is used:

```
FutureGet futureGet = node.getPeer().get(key).start();

futureGet.awaitUninterruptibly();

Object result = futureGet.data().object();

int value = (Integer) result;
```

This code retrieves the value, in this example 1337, as an `Object`. This operation is also executed as a thread. The returned `Object` is cast to its expected data type, in this example, to an integer variable called `value`, on the node which performed the lookup.

Figure 3.2 is a simplified illustration of the `put` and `get` operations. Three nodes with 4-bit IDs are shown. Node 001 performs the put operation, storing the value at node 1010 whose ID is closest to the key (1011). Node 1110 performs the get operation, and receives the resulting value. For simplicity, this diagram does not include the details of routing as is shown in Figure 2.3.

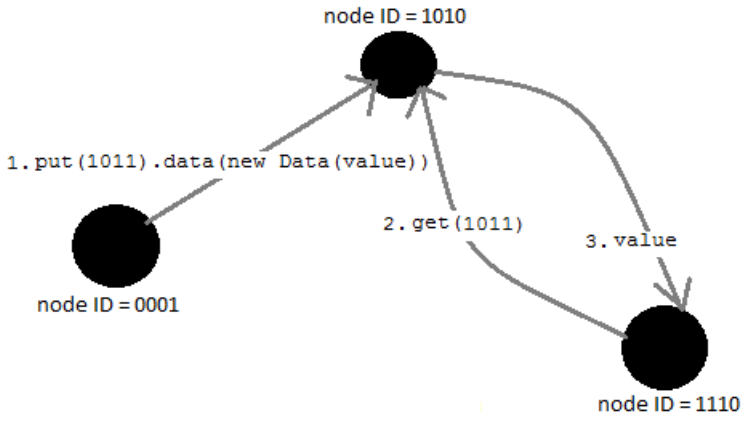
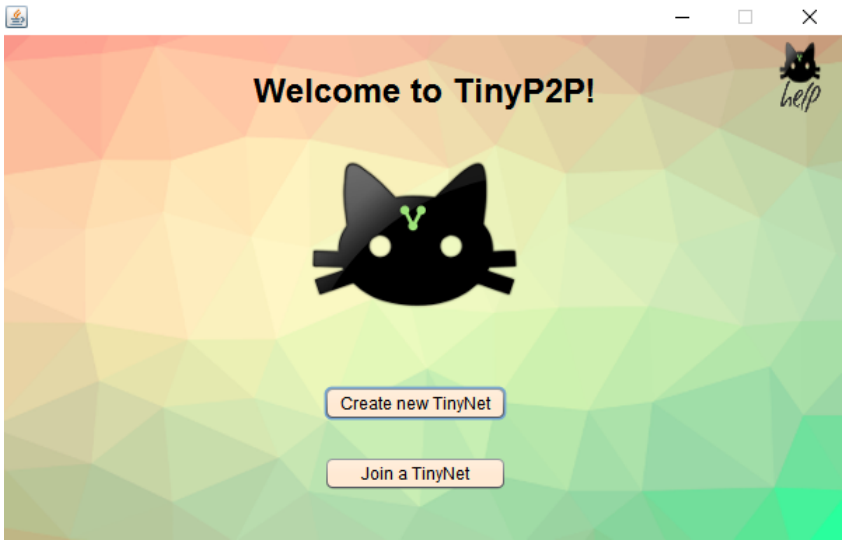


Figure 3.2: Simplified representation of `put` *and* `get` *operations in TomP2P*

3.5.2 Setting up a TinyNet

Upon start-up of the application, the user is presented with the introduction screen (Screenshot 1). We will refer to the first user to start up a new TinyNet as the *initial peer*, and any user subsequently joining the TinyNet as a *joining peer*.



Screenshot 1: TinyP2P Intro screen

When the initial peer clicks the "Create new TinyNet" button, a new `node` is created. Node creation is carried out by the `buildNode()` function, and is performed the same way regardless if the user is the initial peer or a joining peer. Each peer is assigned a randomly generated peer ID of type UUID[2] to uniquely identify them.

If the user is joining, they enter a bootstrap Mnemonic or address in the screen shown in Screenshot 2. The bootstrap address is the IP of any connected peer. The next step is to connect the new node to the DHT via the TinyP2P function `connectNode(networkConfig)`. The initial peer gives the `createInitial(peerID)` function as the `networkConfig` parameter, while a joining peer gives the `create(peerID, bootstrapAddress)` function as the `networkConfig` parameter.

Screenshot 2: Join a TinyNet

Once the user has reached the main menu, TinyP2P starts a TCP server thread that listens for requests and messages. When a download request or a chat message needs to be sent, a new TCP client is created that sends the message to the server running on the recipient's computer.

3.5.3 NAT

TomP2P automatically sets up NAT traversal. If a peer is not reachable by its external IP address, it will attempt to set up port forwarding using UPNP and NATPMP. Failing this, TomP2P allows for the setting up of distributed relaying (Bocek, 2015a).

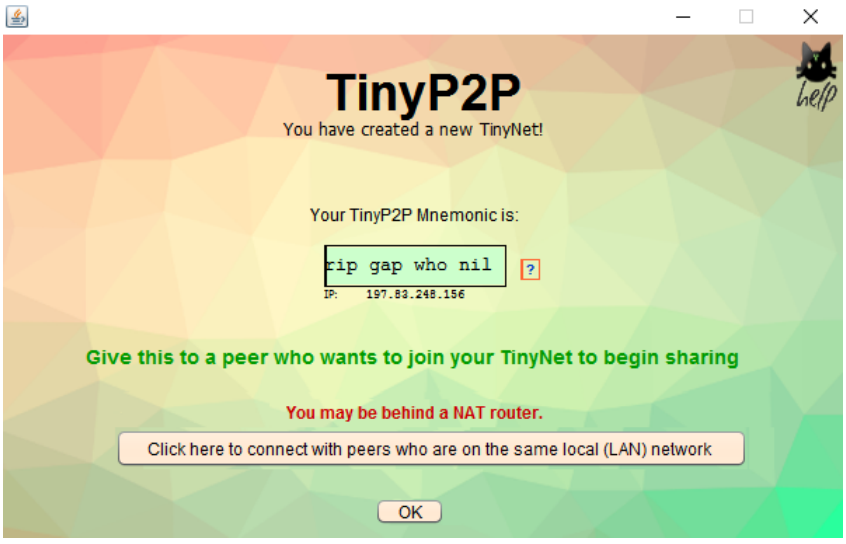
[2]A Universally Unique Identifier (UUID) is a 128-bit value which enables distributed systems to generate unique identifiers across nodes with minimal central coordination.

Dealing with NAT in a User Friendly Manner

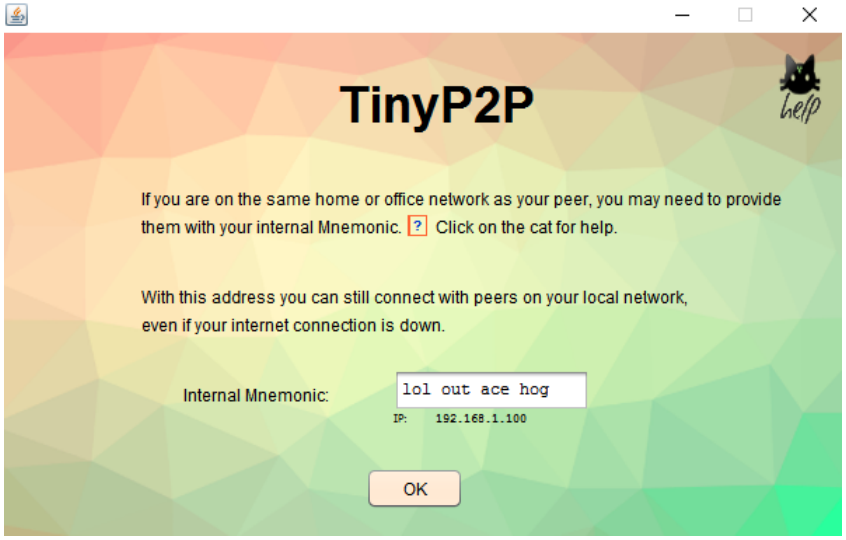
The challenge lies where users behind NATs need to know which of their two Mnemonics to share. To connect with peers on the same subnet would require them to share their internal Mnemonic, while users outside of the subnet would require the external Mnemonic. This needs to be communicated with the user without inundating them with a tutorial about how NAT routers work, but in a way that is informative and not condescending.

After a user has selected to create or join a TinyNet, TinyP2P checks for their external IP address by querying one of three websites. It firsts checks `http://bot.whatismyipaddress.com`, and in the case of failure to connect to the webserver, queries `http://checkip.amazonaws.com` and `http://myexternalip.com/raw` in turn. If connection fails to all three websites, it is assumed that there is a problem with internet connectivity.

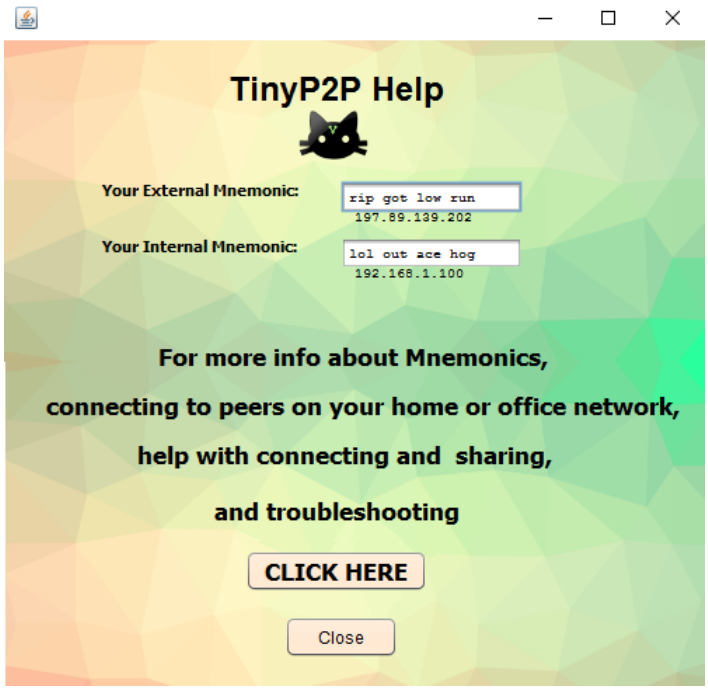
If the returned external IP address differs from the internal IP address, which is discovered via the Java code `InetAddress.getLocalHost()`, it is assumed that the user is behind a NAT router. The user is presented with their external IP Mnemonic, but a red line of text is shown informing them that they may need to provide a different Mnemonic to users who are on the same Local Area Network (LAN) (see Screenshot 3). When the user clicks on the button, they are presented with their internal Mnemonic and a brief explanation (See Screenshot 4). The user is informed to click on the cat icon in order to consult the User Manual for more information (see Screenshot 5).



Screenshot 3: Connection Information after creating new TinyNet



Screenshot 4: Information for the NATted user



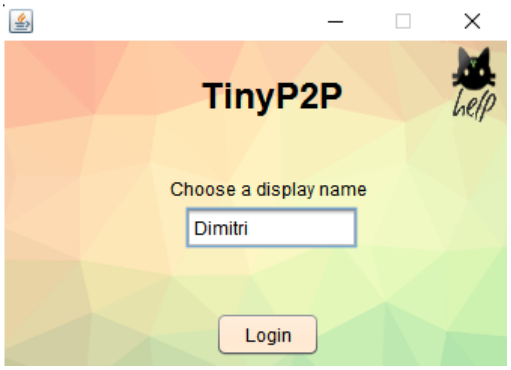
Screenshot 5: The Help page

In the case of internet connection failure, which is detected by failure to retrieve an external IP, the user is informed of this and of the fact that they can still connect to users on their LAN using their internal IP Mnemonic. If the detected external address is the same as the internal address, the user is not behind a NAT and the connection proceeds as normal.

3.5.4 Usernames

TinyP2P allows a user to choose a custom username, by mapping it to the user's peer ID and creating an entry in the DHT.

A TomP2P peer can obtain a list which contains the peer IDs of all peers in the DHT, known as the peer map. When a user enters their chosen username (see Screenshot 6), a DHT entry is created mapping it to their peer ID. Usernames are retrievable from the DHT by the `get` function, with the peer ID as the key.

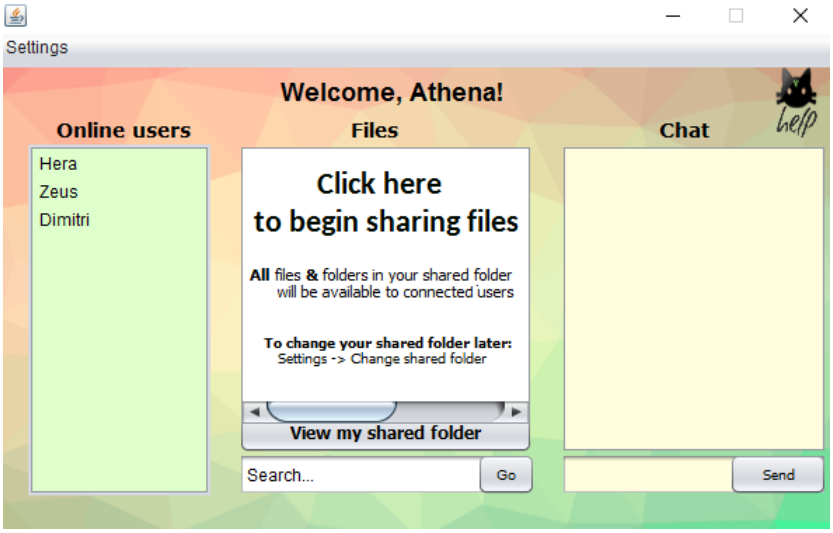


Screenshot 6: Choosing a username

Every ten seconds, the peer map is queried for changes in online users, and the usernames for all connected peers are retrieved from the DHT and displayed in the Online Users panel. TomP2P includes a `PeerMapChangeListener` that listens for peer arrivals and departures. Implementation of this would allow for the Online Users list to only be updated when changes in the peer map occur. However, the method in its current stage appears unstable and falsely notifies that peers have joined or left more often than once a second. For this reason, the method of polling the map every ten seconds for changes was chosen.

3.5.5 Shared Directories

Once the user has chosen their username, they arrive at the main TinyP2P window. The main window has three panels: Online Users, Files and Chat. The Files panel gives brief instructions on how to begin sharing (see Screenshot 7). Clicking on this panel opens the directory chooser dialog.



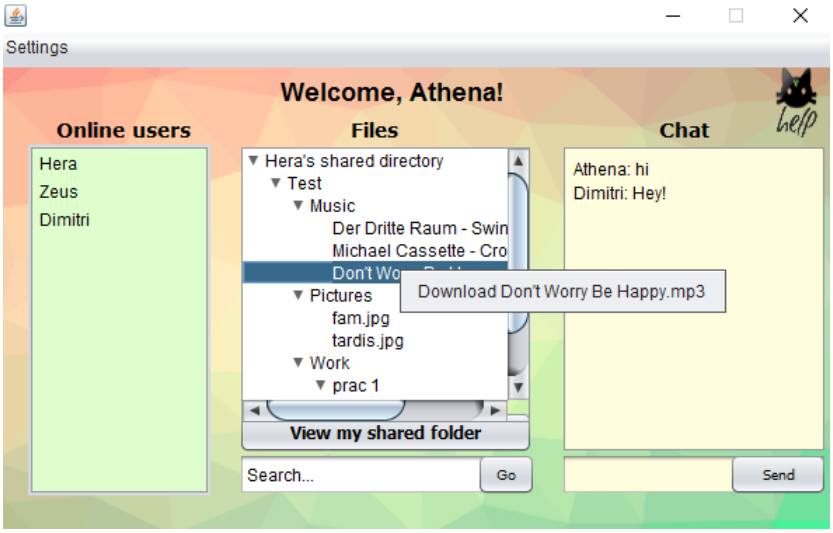
Screenshot 7: The main TinyP2P window upon start up

A user selects a directory which they would like to be available for other users on their TinyNet to see and download. TinyP2P creates a Directory List file, which lists all of the files and sub-folders in the selected directory. This file is then uploaded to the DHT, so that a virtual replication of the file system tree can be constructed for other users. A string containing the user's name and the word "dirlist" is hashed to create the key, and the directory list is the associated value.

When a user clicks on a peer's username in Online Users list, the selected username is hashed and concatenated with the string "dirlist" in order to perform the DHT lookup function which will return the selected user's shared directory list. A tree is then built from the list, emulating a file system tree (see Screenshot 8).

3.5.6 File Searching

TinyP2P uses the same keyword searching mechanism as Kad (see Section 2.5.2). Each file in a shared directory and its sub-directories is made available for searching. When a file is uploaded, the file name is tokenized into keywords which are then hashed and put to the DHT for file location. When a user searches a keyword, TinyP2P hashes the search string and searches for any matching keys in the DHT. If one is found, the shared directory tree of the user who stores the file is returned.



Screenshot 8: The main TinyP2P window, browsing a peer's directory, after double- or right- clicking on a file

3.5.7 Sharing Files

TinyP2P allows for push- and pull- oriented sharing. Users can request downloads from other users, or send their own files to other users. Each peer contains code for a client as well as a server, which listens persistently for download requests, send requests and chat messages.

Requesting a Download

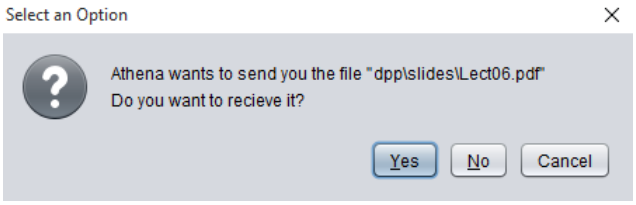
When a user is viewing a peer's shared directory, double-clicking on a file invokes the download process. Right-clicking on a file will bring up an option to download it, as shown in Screenshot 8. This step ensures that the user doesn't accidentally download a file unknowingly by clicking on it while browsing the directory.

When a user selects a file to download, TinyP2P creates a new instance of a download client, giving as parameters the sender's IP, the port and the file name. The client creates a socket and sends a request to the server, including the path and file name of the requested file relative to the sender's shared directory. An empty file with the requested file's name and extension is created in a directory called TinyP2P Downloads, which is a folder the same directory in which TinyP2P is executed. This means that if TinyP2P is run off of an external hard drive or flash drive, the downloads will be stored there. A menu item in the Settings menu opens the TinyP2P Downloads folder.

When the server receives a download request, it locates the requested file and opens a buffered file input stream. It then serializes the file and writes the bits into an output stream which is received at the client side and written to the empty file in the TinyP2P Downloads folder.

Sending a File

If a user is browsing their own shared directory, right- or double-clicking on a file will invoke a process to send the file to a peer. The sender chooses the receiver from a list of online users. The receiver is prompted for confirmation that they want to receive the file (see Screenshot 9).



Screenshot 9: Confirmation to receive file

When the receiver has sent confirmation, their TinyP2P client creates a download request and the process continues as if the receiver had requested the file.

3.5.8 Chat

When a user sends a chat message via the text input box, TinyP2P queries the list of IPs of all online peers and creates a new TCP client with each of them in turn. Chat messages are suffixed with the string "CHTMSG" to identify these messages among download requests that are also sent on this channel. When each of the peers' server modules receive a message ending with the chat message flag, they update their chat window with the contents of the message (see Screenshot 8). It is possible for a file to be created which acts as a chat message, if the file extension ended with the string "CHTMSG," but this is unlikely to be carried out accidentally.

3.6 Security

TomP2P has built-in signature-based security. Data and messages can be signed with a public key via the code `signAndSetPublicKey(keyPair)`, and a peer stores all public keys that it receives on first contact. This allows the recipient to verify the source of data or a message by checking it against its stored public keys. This enables entries in the DHT to be protected, meaning that no one else can overwrite them.

Encryption-based security is not built-in, but users can encrypt their own data.

Security and trust remain open issues in P2P networking, and trust and reputation are sometimes critical to the success of a system. As the intended user base of this system includes novice users, security features are important.

TinyP2P aims to promote some level of security via an interface which is cautionary and informative.

3.6.1 Security in the User Interface

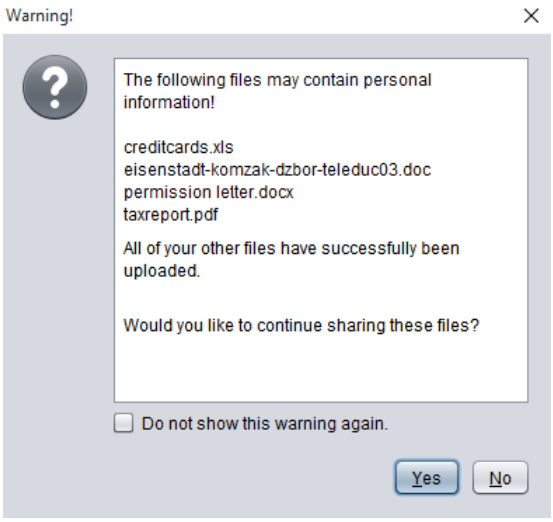
TinyP2P promotes security through the user interface by aligning with the four properties outlined by Good & Krekelberg (2003) in Section 2.6.2.

Firstly, users are aware which of their files are available for viewing and download by other users. A user can choose to view their shared directory, which will appear in the Files panel. All of the files listed in this tree are available for download. The directory tree is rendered with all branches expanded, showing all of the files.

Secondly, users can change which files and folders are being shared. An option in the Settings menu, Change Shared Directory, brings up the file chooser. This will replace whatever folder was previously selected, and when the user views their shared directory it will contain the updated information.

The inclusion of a directory selector with check-boxes to facilitate the sharing of multiple folders was partially implemented, but was abandoned in favour of having a single shared directory to further ensure that users are fully aware of exactly which files are being shared. Additionally, it is simpler to implement a single shared folder because all requested files are located in the same root directory, and the application needs to record less information regarding the user's file tree system further up than the selected directory.

Thirdly, when a user uploads their shared directory and file keywords to the DHT, TinyP2P keeps track of files that may potentially contain sensitive or personal information. The application identifies documents, spreadsheets and email inbox files which have extensions like .doc, .xls, .pdf and .dbx (and more[3]) and may contain personal information as outlined by Good & Krekelberg (2003). Once the directory has been processed, the user is prompted with a list of the identified sensitive files asking if they're sure that they want to proceed to share them. If the user selects "no", the files are not uploaded and if the user selects "yes" they are added to the DHT (see Screenshot 10).



Screenshot 10: Verifying sensitive information upload

On the download side, when a user tries to download a file that has a potentially dangerous extension, for example .exe, .reg, or .cmd, a warning message pops up warning them that this type of file may potentially be malicious or harmful. The user can select whether they wish to continue with the download, and a check-box allows them to disable this error message in future.

With these features, TinyP2P aims to ensure that the user is aware and comfortable with what files are being shared.

[3]docs, docx, dotx, docm, dotm, xla, xlsm, xlsx, xltm, xlam, pptm, ppt, pptm, pptx, potm, ppam, pps and sldm.

Chapter 4

Summary, Conclusions, Challenges and Future Work

4.1 Summary

Section 2.6.2 proposed that *appropriateness to purpose* and *relative ease in learning and remembering to use* require the examination of the intended use and intended user base of a system, respectively.

TinyP2P is intended to facilitate sharing files among task- or interest- related organizations or groups of people with varying levels of computing and networking experience.

It creates dynamic ad-hoc file sharing networks with minimized setup overhead, that first-time users should have little difficulty in joining, using and setting up.

The IP Mnemonic system and succinct on-screen instructions aim to make the system easy to set up and master. TinyP2P provides a simple but informative user interface to enable distributed file sharing, searching, directory browsing and chat between users.

TinyP2P is lightweight and portable, taking up less than 6.5MB of disk space, and can be run from a flash drive without installation.

The application automatically deals with NATs and can be used to connect peers over a LAN in the absence of an internet connection. TinyP2P is unmoderated and free to use, satisfying the most important end-user requirement identified by both ours and Lee's study.

TinyP2P uses decentralization to overcome several significant challenges to uptake faced by file sharing systems, namely availability, freeloading and illegality. TinyP2P requires no registration or connection to a server, creating structured overlay networks to facilitate ad hoc, user friendly file sharing. The user base and file base of each TinyNet is purpose-specific, addressing the problem of freeloading and taking advantage of the phenomenon of small-world file sharing communities.

P2P traffic within TinyNet overlays networks is difficult to detect, and the ad hoc nature in which they are created means that they are not prone to being shut down by an outside party.

4.2 Conclusions

This research has shown the viability of a purely decentralized file sharing application that requires no registration or central server coordination. It investigated the issues of availability, freeloading and legality in order to develop an application that addressed these issues using IP Mnemonics. It has reviewed aspects of HCI and UI in attempt to build a user friendly and familiar interface for use by a wide spectrum of users.

4.3 Challenges

The first challenge to development was the realization that progress on the Hive2Hive library had been suspended, shortly after this project was started. Hive2Hive has not been receiving attention as most of its core developers, students at the University of Zurich in Switzerland, have graduated (Rutishauser, 2015). More focus was initially intended to be placed on the usability, HCI and user interface components of the project, building on Hive2Hive's sharing, synchronization, versioning and user management components. It was realized that much of this functionality would have to be implemented before usability and UI components could be incorporated.

Another challenge was the poor documentation of both TomP2P and Hive2Hive libraries. They do not include extensive Java Documentation (JavaDocs) and sometimes do not conform to popular Java coding standards, and so a lot of time was dedicated to perusing and learning how the libraries operate

A third challenge was the necessity of using Java and the Swing GUI builder, due to Hive2Hive and TomP2P both being written in Java. Swing applications are slow to build and difficult to maintain, and a notably large amount of time was spent on each small component of the UI.

4.4 Future Work

TinyP2P was the result of a first attempt at a working distributed application and GUI, and was completed over the course of nine months. TinyP2P facilitates directory browsing, file sharing and text chat, but shows a basis on which a richer application with features like video chat, geolocation status and social network features could be developed.

The capability of resuming down- and up-loading after restarting the application is, as shown by both the study conducted by Lee (2003) as well as the survey conducted in parallel with this research, a highly desirable feature. Methods to recording the details of current down-/uploads in a unique, persistent manner may assist in the development of loading resumption.

Hive2Hive contains libraries that facilitate file synchronization and versioning. Upon further refinement of the Hive2Hive project, these features could be included in TinyP2P.

The possibility of adding a network password for TinyNets depends on the extension of the underlying DHT implementation, TomP2P, since any functionality of this sort added to the application level can easily be undermined by exploiting the DHT. Personal communication with the TomP2P developer revealed that this is under development and would greatly improve security (Bocek, 2015b).

A mobile implementation would require re-evaluation of the user interface design. The application is built on the Java Virtual Machine (JVM) and is thus compatible with a multitude of platforms. One challenge that will be faced by a mobile implementation is the fact that the user will have to reconfigure TinyP2P every time their IP address is reallocated. This means that the user may not be able to roam between different access points while using the application.

The current Mnemonic system works for all addresses within the IPv4 address space, but will not be applicable with the introduction of IPv6 addresses. A larger mapping system will need to be implemented to handle these.

A sharing palette GUI component, as suggested by Voida *et al.* (2006) (see Section 2.6.3) could improve the user experience of TinyP2P.

References

Aberer, K, & Despotovic, Z. 2001. Managing trust in a peer-2-peer information system. *Pages 310–317 of: Proceedings of the Tenth International Conference on Information and Knowledge Management*. ACM.

Aiello, L. M, Milanesio, M, Ruffo, G, & Schifanella, R. 2008. Tempering Kademlia with a robust identity based system. *Pages 30–39 of: The Eighth International Conference on Peer-to-Peer Computing*. IEEE.

Anderson, D. P, Cobb, J, Korpela, E, Lebofsky, M, & Werthimer, D. 2002. SETI@ home: an experiment in public-resource computing. *Communications of the ACM*, **45**(11), 56–61.

Barker, G. 2015 (7 October). *Student Disciplinary Prosecutor at Rhodes University*. Personal communication.

BitTorrent. 2014a (May). *The Basics of BitTorrent*. Online. Available from: `http://help.bittorrent.com/customer/en/portal/articles/178790`. Accessed: 10 October 2015.

BitTorrent. 2014b (July). *Client Features*. Online. Available from: `http://help.bittorrent.com/customer/en/portal/articles/163493`. Accessed: 10 October 2015.

Blackburn, J, & Christensen, K. 2009 (June). A simulation study of a new green BitTorrent. *Pages 1–6 of: The International Conference on Communications Workshops*. IEEE.

Bocek, T. 2015a. *The TomP2P Website*. Online. Available from: `http://tomp2p.net/`. Accessed: 23 May 2015.

Bocek, T. 2015b (15 April). *TomP2P developer*. Personal communication.

Brian, M. 2012 (January). *The Pirate Bay says it will no longer serve Torrents, shifts to Magnet links.* The Next Web. Online. Available from: `http://thenextweb.com/insider/2012/01/13/the-pirate-bay-says-it-will-no-longer-serve-torrents-shifts-to-magnet-links`. Accessed: 10 August 2015.

Brooke, J. 1996. SUS: A quick and dirty usability scale. *Usability evaluation in industry*, **189**(194), 4–7. London.

Carlisle, J. H. 1976 (June). Evaluating the impact of office automation on top management communication. *Pages 611–616 of: Proceedings of The National Computer Conference and Exposition.* ACM.

Castro, M, Druschel, P, Ganesh, A, Rowstron, A, & Wallach, D. S. 2002. Secure routing for structured peer-to-peer overlay networks. *ACM SIGOPS Operating Systems Review*, **36**(SI), 299–314. ACM.

Cheshire, S, & Krochmal, M. 2013 (April). *NAT Port Mapping Protocol (NAT-PMP).* RFC 6886.

Christin, N, Weigend, A. S, & Chuang, J. 2005. Content availability, pollution and poisoning in file sharing peer-to-peer networks. *Pages 68–77 of: Proceedings of the 6th ACM Conference on Electronic Commerce.* ACM.

Crosby, S. A, & Wallach, D. S. 2007. *An analysis of BitTorrent's two Kademlia-based DHTs.* Technical report. TR07-04, Rice University.

Dhungel, P, Wu, D, Hei, X, Schonhorst, B, & Ross, K. W. 2007. *Is BitTorrent Unstoppable?* Online. Available from: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.414.971&rep=rep1&type=pdf`. Accessed: 10 August 2015.

Eisenstadt, M, Komzak, J, & Dzbor, M. 2003. Instant messaging + maps = powerful collaboration tools for distance learning. *In: Proceedings of TelEduc03.* TelEduc.

Feldman, M, & Chuang, J. 2005. Overcoming free-riding behavior in peer-to-peer systems. *ACM SIGecom Exchanges*, **5**(4), 41–50.

Foster, I, & Iamnitchi, A. 2003. On death, taxes, and the convergence of peer-to-peer and grid computing. *Pages 118–128 of: Peer-to-Peer Systems II.* Springer.

Ganguly, A, Agrawal, A, Boykin, P. O, & Figueiredo, R. 2006. IP over P2P: enabling self-configuring virtual IP networks for grid computing. *Page 10 of: The 20th International Parallel and Distributed Processing Symposium.* IEEE.

Ge, Z, Figueiredo, D. R, Jaiswal, S, Kurose, J, & Towsley, D. 2003. Modeling peer-peer file sharing systems. *Pages 2188–2198 of: The 22nd Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3. IEEE.

Germundsson, R, & Weisstein, E. W. 2002. *XOR*. MathWorld – A Wolfram Web Resource. Online. Available from: `http://mathworld.wolfram.com/XOR.html`. Accessed: 20 September 2015.

Good, N. S, & Krekelberg, A. 2003. Usability and privacy: a study of Kazaa P2P file-sharing. *Pages 137–144 of: Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM.

Hu, Z. 2005. NAT traversal techniques and peer-to-peer applications. *Pages 4–26 of: HUT T-110.551 Seminar on Internetworking*. Helsinki University of Technology.

Hunt, K. R. 2014. *Investigating the Viability of Peer-to-Peer Applications using BitTorrent Sync*. Online. Available from: `http://www.cs.ru.ac.za/research/g11h3779/kieran-thesis/kieran-thesis.html`. Accessed: 19 September 2015.

Iamnitchi, A, Ripeanu, M, & Foster, I. 2004. Small-world file-sharing communities. *Pages 952–963 of: The 23rd Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 2. IEEE.

Java Documentation. 2014. *Interface Future<V>*. Online. Available from: `https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Future.html`. Accessed: 27 October 2015.

Karagiannis, T, Broido, A, Brownlee, N, Claffy, K, & Faloutsos, M. 2004. Is P2P dying or just hiding? *Pages 1532–1538 of: Global Telecommunications Conference*, vol. 3. IEEE.

Kaune, S, Lauinger, T, Kovačević, A, & Pussep, K. 2008. Embracing the peer next door: Proximity in Kademlia. *Pages 343–350 of: Eighth International Conference on Peer-to-Peer Computing*. IEEE.

Kaur, K, & Rai, A. K. 2014. A Comparative Analysis: Grid, Cluster and Cloud Computing. *International Journal of Advanced Research in Computer and Communication Engineering*, **3**(3), 1.

Kurose, J. F, & Ross, K. W. 2013. *Computer Networking: A Top-Down Approach*. 6th edn. Pearson.

Kwon, G, & Ryu, K. D. 2003. An efficient peer-to-peer file sharing exploiting hierarchy and asymmetry. *Pages 226–233 of: Symposium on Applications and the Internet Proceedings*. IEEE.

Le Fessant, F, Handurukande, S, Kermarrec, A, & Massoulié, L. 2005. Clustering in peer-to-peer file sharing workloads. *Pages 217–226 of: Peer-to-Peer Systems III*. Springer.

Lee, J. 2003. An end-user perspective on file-sharing systems. *Communications of the ACM*, **46**(2), 49–53.

Loewenstern, A, & Norberg, A. 2013 (March). *BitTorrent Enhancement Proposals*. Online. Available from: `http://www.bittorrent.org/beps/bep_0005.html`. Accessed 21 October 2015.

Lou, X, & Hwang, K. 2009. Collusive piracy prevention in P2P content delivery networks. *IEEE Transactions on Computers*, **58**(7), 970–983.

Maymounkov, P, & Mazieres, D. 2002. Kademlia: A peer-to-peer information system based on the XOR metric. *Pages 53–65 of: Peer-to-Peer Systems*. Springer.

Milojicic, D. S, Kalogeraki, V, Lukose, R, Nagaraja, K, Pruyne, J, Richard, B, Rollins, S, & Xu, Z. 2002. *Peer-to-peer computing*. Technical report. HPL-2002-57, HP Labs.

Mohr, G. 2002 (June). *Magnet v0.1*. Online. Available from: `http://magnet-uri.sourceforge.net/magnet-draft-overview.txt`. Accessed: 15 September 2015.

Oxford Dictionaries Online. 2015. *Mnemonic*. Online. Available from: `http://www.oxforddictionaries.com/definition/english/mnemonic`. Accessed 14 September 2015.

Pouwelse, J. A, Garbacki, P, Epema, D, & Sips, H. J. 2004. *A measurement study of the BitTorrent peer-to-peer file-sharing system*. Technical report. PDS-2004-003, Delft University of Technology, The Netherlands.

Raskin, J. 1994. Intuitive Equals Familiar. *Communications of the ACM*, **37**(9), 17.

Ratnasamy, S, Francis, P, Handley, M, Karp, R, & Shenker, S. 2001. A Scalable content-addressable network. *Pages 161–172 of: The 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, vol. 4. ACM.

Reeves, L. M, Lai, J, Larson, J. A, Oviatt, S, Balaji, T, Buisine, S, Collings, P, Cohen, P, Kraal, B, & Martin, J.-C. 2004. Guidelines for multimodal user interface design. *Communications of the ACM*, **47**(1), 57–59.

Rogers, Y, Sharp, H, & Preece, J. 2011. *Interaction design: beyond human-computer interaction*. John Wiley & Sons.

Rowstron, A, & Druschel, P. 2001. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Pages 329–350 of: Middleware 2001*. Springer.

Rutishauser, N. 2015 (6 September). *Hive2Hive developer*. Personal communication.

Saroiu, S, Gummadi, P. K, & Gribble, S. D. 2001. Measurement study of peer-to-peer file sharing systems. *Pages 156–170 of: Electronic Imaging 2002*. International Society for Optics and Photonics.

Scarlata, V, Levine, B. N, & Shields, C. 2001. Responder anonymity and anonymous peer-to-peer file sharing. *Pages 272–280 of: The Ninth International Conference on Network Protocols*. IEEE.

Schoder, D, & Fischbach, K. 2003. Peer-to-peer prospects. *Communications of the ACM*, **46**(2), 27–29.

Sen, S, Spatscheck, O, & Wang, D. 2004. Accurate, scalable in-network identification of P2P traffic using application signatures. *Pages 512–521 of: Proceedings of the 13th international conference on World Wide Web*. ACM.

Sharma, R, Datta, A, Amico, M. D, & Michiardi, P. 2011. An empirical study of availability in friend-to-friend storage systems. *Pages 348–351 of: The International Conference on Peer-to-Peer Computing*. IEEE.

Spognardi, A, Lucarelli, A, & Pietro, R. D. 2005. A methodology for P2P file-sharing traffic detection. *Pages 52–61 of: The Second International Workshop on Hot Topics in Peer-to-Peer Systems*. IEEE.

Sripanidkulchai, K, Maggs, B, & Zhang, H. 2003. Efficient content location using interest-based locality in peer-to-peer systems. *Pages 2166–2176 of: The 22nd Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3. IEEE.

Steiner, M, Effelsberg, W, En-Najjary, T, & Biersack, E. W. 2007. Load reduction in the Kad peer-to-peer system. *In: The Fifth International Workshop on Databases, Information Systems and Peer-to-Peer Computing*. University of Crete.

Stoica, I, Morris, R, Karger, D, Kaashoek, M. F, & Balakrishnan, H. 2001. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, **31**(4), 149–160.

Taylor, I, Shields, M, Wang, I, & Philp, R. 2003. Distributed P2P computing within Triana: A galaxy visualization test case. *Page 8 of: Proceedings of the International Parallel and Distributed Processing Symposium*. IEEE.

Voida, S, Edwards, W. K, Newman, M. W, Grinter, R. E, & Ducheneaut, N. 2006. Share and share alike: exploring the user interface affordances of file sharing. *Pages 221–230 of: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM.

Vu, Q. H, Lupu, M, & Ooi, B. C. 2009. *Peer-to-peer computing: Principles and applications*. Springer Science & Business Media.

Wallach, D. S. 2003. A survey of peer-to-peer security issues. *Pages 42–57 of: Software Security: Theories and Systems*. Springer.

Wang, C, Yang, N, & Chen, H. 2010. Improving lookup performance based on Kademlia. *Pages 446–449 of: The Second International Conference on Networks, Security, Wireless Communications and Trusted Computing*, vol. 1. IEEE.

Zhao, B. Y, Kubiatowicz, J, & Joseph, A. D. 2001. *Tapestry: An infrastructure for fault-tolerant wide-area location and routing*. Online. Available from: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.24.7439&rep=rep1&type=pdf`. Accessed: 29 May 2015.

Zoels, S, Despotovic, Z, & Kellerer, W. 2006. Cost-based analysis of hierarchical DHT design. *Pages 233–239 of: The Sixth International Conference on Peer-to-Peer Computing*. IEEE.

Appendix A

Distributed Survey

File Sharing Application Perspectives

File sharing plays a significant role in our online lives. A file sharing application is anything that you use to send AND receive files from another system. I'd like to know your perspectives on existing applications and what you think would make a great file sharing app. Your answers will be used to aid the design of a purely decentralized peer-to-peer file sharing application called TinyP2P.

Your responses are anonymous unless you choose to include your email address, and all questions are optional. You must be 18 years or older to participate.

* Required

*

- [ ] I am 18 years or older, and I consent to my information being used for research purposes

Do you have any formal academic background in Computer Science or Information Systems?
Please select which of the following apply to you

- [ ] I studied IT in school
- [ ] I studied/am studying Computer Science or Information Systems at university
- [ ] I have no formal academic background in Computer Science

Are you a Rhodes University student or staff member?

- ○ Student
- ○ Staff
- ○ I am not from Rhodes

How often do you use file sharing applications?
How much does file sharing factor into your daily activities?

- ○ Never
- ○ Occasionally
- ○ Frequently

What do you use file sharing for?

Check all the activities that you frequently take part in

- [ ] Downloading movies, music and TV shows
- [ ] Sharing files between work colleagues
- [ ] Collaborating with others on projects
- [ ] Sharing photos
- [ ] Downloading Open Source software
- [ ] In a learning environment (ie school or university)
- [ ] Other:

Do you download and/or share illegal material?

Do you use file sharing to access copyrighted movies, music, TV shows or software? This information is for research purposes only and your response will be kept anonymous.

- ( ) Yes
- ( ) No

Which of these file sharing applications have you used?

Please select all the applications that you have used

- [ ] BitTorrent (uTorrent, etc)
- [ ] DirectConnect (DC++)
- [ ] DropBox
- [ ] Google Drive
- [ ] LimeWire
- [ ] Windows HomeGroups
- [ ] GitHub

Which other file sharing applications do you use?

List some applications not listed above that you use

Which of these features are most important to you?

Select which features of a file sharing application you think are of utmost importance. Select the features without which a file sharing application would be insufficient.

- [ ] Free to use
- [ ] Has an interface that is easy to use
- [ ] Gives good error messages
- [ ] Is stable, fast and reliable
- [ ] Good security features
- [ ] Is ad free
- [ ] Has good search features
- [ ] Can resume downloading and uploading after restart
- [ ] Supports chat, direct messaging, buddy list
- [ ] Can control spam
- [ ] Can organize files as a library
- [ ] Can credit contributors, has points for uploading
- [ ] Can filter content
- [ ] Is open source
- [ ] Only supports legal files
- [ ] Has a large user base
- [ ] Has a large selection of files

What else?

What features or properties aren't listed above that you feel are important or desirable in a file sharing application?

Would you be willing to participate in usability testing for TinyP2P?

For science!

- ( ) Yes
- ( ) No
- ( ) Maybe

If so, please provide your email address so I can contact you closer to testing time

Appendix B

Tabulated results of survey

Table B.1: Number of Votes per Feature Out of 96 Respondents (2015)

	Features	Number of votes
1	Free to use	88
2	Is stable, fast and reliable	80
3	Can resume downloading and uploading after restart	75
4	Good security features	67
5	Has an interface that is easy to use	65
6	Has good search features	63
7	Is ad free	59
8	Can control spam	55
9	Has a large selection of files	54
10	Can organize files as a library	52
11	Can filter content	48
12	Is open source	41
13	Has a large user base	37
14	Gives good error messages	29
15	Supports chat, direct messaging, buddy list	21
16	Can credit contributors, has points for uploading	11
17	Only supports legal files	8

Appendix C

User Manual

TinyP2P User Manual

Contents

What is TinyP2P?
TinyP2P is an app that lets you share files and chat with friends.

What do I need to run TinyP2P?
TinyP2P doesn't require installation or account registration.
All you need to run TinyP2P is the Java Runtime Environment
and connection to a network.

How do I connect to a TinyNet?
If you are the first person to start the TinyNet, select the Create New TinyNet button.
If one of your peers has already started a TinyNet, select the Join TinyNet button.
Click here if you are having problems connecting.

I have created a new TinyNet, what now?
When you have created your TinyNet, you will see your TinyP2P Mnemonic and IP Address.
A Mnemonic is a phrase made up of four words, that translates to your IP Address.
Give either this phrase or your IP Address to a peer who wants to connect to your TinyNet.
Once your peers have joined your TinyNet, it's time to start sharing!

How do I join a TinyNet?
To join your peer's TinyNet, you need their Mnemonic or IP Address. Your peer can check their Mnemonic by clicking on the cat icon in the top right corner of the TinyP2P window.
You can enter the Mnemonic of any peer who is connected to a TinyNet. It does not have to be the Mnemonic of the peer who created the TinyNet.

If you are on the same home or office network as your peers, and are having trouble connecting, click here.

What is a Mnemonic?
TinyP2P connects you to your peers using IP Addresses.
A Mnemonic is just a four-word alternative to your IP Address.
Mnemonics let you communicate your IP Address to your peers more easily.
Instead of telling them your IP, which may look something like 192.168.1.254,
you can tell them your Mnemonic which may look something like "tip cup key dog."
You can always use a literal IP Address instead of a Mnemonic if you like.

How do I check my Mnemonic?
Clicking on the cat icon in the top right corner of the window will bring up your Mnemonic.

Your shared folder
The first step is to select a folder that you would like to share with your peers.
All the files and folders within this folder will be available to other users to see and download.
Make sure that there are no personal files inside the folder you select to share,
as well as all the folders inside the shared folder.
You can change your shared folder at any time by selecting "Settings -> Change shared folder."
You can view which files and folders are being shared by clicking on the "View my shared folder" button.

Downloading files
Your online peers will appear on the left of the screen, with their chosen TinyP2P usernames.
Click on a peer's username to view their shared folder.
Double-click or right-click on a file that you want, to begin downloading it.

Sending files
If you are viewing your own shared folder and want to send a file to a peer,
double-click or right-click on the file that you want to send and select "send file."
Enter the username of the user that you want to send the file to.
The user will be prompted to accept or reject your file.

I can't connect to a TinyNet!
1. Your firewall may be blocking connections
   Make sure that your firewall is not blocking Java applications.

2. You may be behind a Network Address Translator router
   If you are on the same home or office network as your peers, your router may be configured
   to use two different IP Addresses: One for peers on your network, and one for people over the internet.
   TinyP2P gives you both your **internal** and external Mnemonics.
   Use your **internal** Mnemonic to connect to peers **on your home or office network**.
   Use your **external** Mnemonic to connect to peers **over the internet**.
   To view both of your Mnemonics, click on the cat icon in the top right corner of the window.
   If your router isn't a NAT router, your internal and external Mnemonics will be the same.

3. Check your internet connection
   TinyP2P lets you connect to peers who are on the same Local network (LAN) as you,
   even when the internet is down. You need to provide your **internet Mnemonic** to connect to peers
   over LAN when there is no internet connection.