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

Nichola Reid[1], Yusuf Motara[2]
Department of Computer Science
Rhodes University, P. O. Box 94,
Grahamstown 6140
[1] nicky.reid92@gmail.com
[2] y.motara@ru.ac.za

ABSTRACT

Peer-to-Peer is said to be the next era of the Internet, enabling the development of efficient and scalable distributed applications. File sharing is a popular online activity carried out by millions of users daily, 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.

Categories and Subject Descriptors

Software and its engineering → Peer-to-peer architectures Human-centered computing → User interface design

General Terms

Design, Human Factors.

Keywords

Distributed Hash Tables, File Sharing, Peer-to-Peer, Usability

1. INTRODUCTION

Many different file sharing protocols exist, with varying features, 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 [1]. 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.

This project was conducted to investigate challenges to uptake of file sharing systems, 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.

2. Related Work

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 [2].

2.1 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 [3].

The real-time online presence of peer-group members in a long distance learning environment could improve students' emotional well-being and academic performance [4]. The 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 [5].

Interest-based locality means that peers who have similar content stored in their caches form clusters and send requests directly to each other before consulting the server [6]. Interest-based locality shortcuts can reduce many queries to single-hops, and total load can be lowered by a factor of between 3 and 7 when tested on five different content distribution applications, including Kazaa and Gnutella [7].

2.2 Peer-To-Peer

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 [8]. 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 [9].

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 [10].

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 [11].

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 [12]. In distributed P2P systems, storage and overhead processing are divided between peers, making them highly scalable.

[13] 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.

They 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. The also conclude 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.

2.3 Challenges to Uptake of a File Sharing System

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 [14].

To address this centralized method of file location, the use of DHT, Peer Exchange (PeX) and Magnet links were introduced. 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.

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 [14]. They do, however, also allow for the facilitation of a number of other methods of sharing, including Trackers and direct web downloads.

As of 2012, one of the most popular and resilient torrent websites, The Pirate Bay, 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 [17].

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 [16]. 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 [3] suggested that anti-freeloading mechanisms, such as upload requirements or point-based reward systems, would be detrimental to the overall 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-aller* 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 [14].

The load-balancing nature of DHT 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 [16].

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 [23].

2.3.3 Legality

Online piracy is a significant hindrance to the commercial and legal use of P2P file sharing.

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 [18]. University and network administrators do not want to bear the legal and reputational consequences of the actions of the DC network users.

2.4 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 [24].

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 [2].

2.4.1 *CHORD, CAN, Pastry and Tapestry*

The CHORD protocol maps a key to a node using consistent hashing, which balances load, as nodes are probabilistically assigned roughly the same number of keys. Decentralization and load balancing make CHORD robust and suitable for large-scale P2P networks even in the face of concurrent node arrivals and departures [25].

Content-Addressable Network (CAN) uses a virtual multidimensional Cartesian coordinate space, dynamically partitioned among participating nodes in order to perform optimized routing [2].

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. [26]. Tapestry is similar to Pastry, but provides better probabilistic bounds on routing distances and guarantees that an existing object is always reachable [27].

2.4.2 *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 [21].

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 [15].

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 [28].

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

2.5 End User Perspectives

2.5.1 *User Interface and Features*

The survey of end user perspectives of P2P file sharing system requirements conducted by [3] 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. 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 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.

2.5.2 *Human Computer Interaction*

The qualitative 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.

[29] 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. 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.

[31] Suggests that the usability of a 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 [3].

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.

2.6 Security

[20] Note that P2P exposes more security threats than client-server and centralized models, and that some level of trust is necessary among peers.

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 [33].

P2P has the ability to resist Denial of Service attacks, as peers are aware of each other's' existence and the network can recover even if a number of nodes have come under attack, by re-configuring itself around these "holes" [21].

2.6.1 *A Secure User Interface*

A study conducted by [22] 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.

[22] 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. [22] 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.

2.7 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.

Users in a purely serverless P2P system are required to 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 [10]. A Friend-to-Friend network is one that consists of users who trust each other.

2.8 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 [19].

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 [30]. Relaying allows for peers behind NATs to communicate with each other via another "relay" peer who is not constrained by a NAT.

3. Usability Survey and the TinyP2P 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.

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 [3], 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.

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.

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.

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 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. 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. The system of IP Mnemonics maps a three-letter English word to one of the numbers from 0 to 255, making a unique four-word, four-syllable phrase translation of any IP address. 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.

3.4 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.

A cat icon in the top right corner of each window opens the Help window, which displays the user's Mnemonics and IP Addresses, as well as a link to the User Manual.

3.4.1 Setting up a TinyNet

Upon start-up of the application, the user is presented with the introduction screen. 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.

The initial peer creates a new TinyNet. If the user is joining, they enter a bootstrap Mnemonic or address before connecting. The bootstrap address is the IP of any connected peer.

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.

3.4.2 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 [34].

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 three websites 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). When the user clicks on the button, they are presented with their internal Mnemonic and a brief explanation.

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.

3.4.3 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, 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.

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.

3.4.4 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. Clicking on this panel opens the directory chooser dialog.

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.

3.4.5 File Searching

TinyP2P uses the same keyword searching mechanism as Kad. 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.

3.4.6 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.

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.

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.

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.

3.4.7 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. 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.

3.5 Security

TomP2P has built-in signature-based security. 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.

3.5.1 3.6.1 Security in the User Interface

TinyP2P promotes security through the user interface by aligning with the four properties outlined by [22].

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.

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 may contain personal information as outlined by [22]. 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.

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.

4. Summary

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. 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 any outside party.

5. 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.

6. 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 [32]. 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.

7. 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 [3] 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.

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 [33].

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.

8. ACKNOWLEDGMENTS

I would like to thank my supervisor, Mr Yusuf Motara, for his guidance and assistance throughout the project. 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.

9. REFERENCES

[1] Aberer, Karl and Despotovic, Zoran, "Managing trust in a peer-2-peer information system", in Proceedings of the Tenth International Conference on Information and Knowledge Management (2001), pp. 310--317.

[2] Ratnasamy, Sylvia and Francis, Paul and Handley, Mark and Karp, Richard and Shenker, Scott, "A Scalable content-addressable network", in The 2001 conference on Applications,

technologies, architectures, and protocols for computer communications vol. 4, no. 31 (2001), pp. 161-172.

[3] Lee, Jintae, "An end-user perspective on file-sharing systems", Communications of the ACM 46, 2 (2003), pp. 49--53.

[4] Eisenstadt, Marc and Komzak, Jiri and Dzbor, Martin, "Instant messaging + maps = powerful collaboration tools for distance learning", in Proceedings of TelEduc03 (2003).

[5] Kwon, Gisik and Ryu, Kyung Dong, "An efficient peer-to-peer file sharing exploiting hierarchy and asymmetry", in Symposium on Applications and the Internet Proceedings (2003), pp. 226--233.

[6] Le Fessant, Fabrice and Handurukande, Sidath and Kermarrec, AM, and Massoulié, Laurent, "Clustering in peer-to-peer file sharing workloads", in Peer-to-Peer Systems III (Springer, 2005), pp. 217--226.

[7] Sripanidkulchai, Kunwadee and Maggs, Bruce and Zhang, Hui, "Efficient content location using interest-based locality in peer-to-peer systems", in The 22nd Annual Joint Conference of the IEEE Computer and Communications Societies vol. 3, (2003), pp. 2166--2176.

[8] Taylor, Ian and Shields, Matthew and Wang, Ian and Philp, Roger, "Distributed P2P computing within Triana: A galaxy visualization test case", in Proceedings of the International Parallel and Distributed Processing Symposium (2003), pp. 8.

[9] Foster, Ian and Iamnitchi, Adriana, "On death, taxes, and the convergence of peer-to-peer and grid computing", in Peer-to-Peer Systems II (Springer, 2003), pp. 118--128.

[10] Sen, Subhabrata and Spatscheck, Oliver and Wang, Dongmei, "Accurate, scalable in-network identification of P2P traffic using application signatures", in Proceedings of the 13th international conference on World Wide Web (2004), pp. 512--521.

[11] Spognardi, Angelo and Lucarelli, Alessandro and Pietro, Roberto Di, "A methodology for P2P file-sharing traffic detection", in The Second International Workshop on Hot Topics in Peer-to-Peer Systems (2005), pp. 52--61.

[12] Vu, Quang Hieu and Lupu, Mihai and Ooi, Beng Chin, Peer-to-peer computing: Principles and applications (Springer Science & Business Media, 2009).

[13] Ge, Zihui and Figueiredo, Daniel R. and Jaiswal, Sharad and Kurose, Jim and Towsley, Don, "Modeling peer-peer file sharing systems", in The 22nd Annual Joint Conference of the IEEE Computer and Communications Societies vol. 3, (2003), pp. 2188--2198.

[14] Pouwelse, Johan A. and Garbacki, Pawe and Epema, Dick and Sips, Henk J., "A measurement study of the BitTorrent peer-to-peer file-sharing system", PDS-2004-003, Delft University of Technology, The Netherlands (2004).

[15] Christin, Nicolas and Weigend, Andreas S. and Chuang, John, "Content availability, pollution and poisoning in file sharing peer-to-peer networks", in Proceedings of the 6th ACM Conference on Electronic Commerce (2005), pp. 68--77.

[16] Feldman, Michal and Chuang, John, "Overcoming free-riding behavior in peer-to-peer systems", ACM SIGecom Exchanges 5, 4 (2005), pp. 41--50.

[17] Matt Brian, "The Pirate Bay says it will no longer serve Torrents, shifts to Magnet links" (2012).

[18] Gordon Barker, "Student Disciplinary Prosecutor at Rhodes University" (2015).
[19] Hu, Zhou, "NAT traversal techniques and peer-to-peer applications", in HUT T-110.551 Seminar on Internetworking (2005), pp. 4-26.
[20] Milojicic, Dejan S. and Kalogeraki, Vana and Lukose, Rajan and Nagaraja, Kiran and Pruyne, Jim and Richard, Bruno and Rollins..., "Peer-to-peer computing", HPL-2002-57, HP Labs (2002).

[21] Maymounkov, Petar and Mazieres, David, "Kademlia: A peer-to-peer information system based on the XOR metric", in Peer-to-Peer Systems (Springer, 2002), pp. 53--65.

[22] Good, Nathaniel S. and Krekelberg, Aaron, "Usability and privacy: a study of Kazaa P2P file-sharing", in Proceedings of the SIGCHI conference on Human factors in computing systems (2003), pp. 137--144.

[23] Crosby, Scott A. and Wallach, Dan S., "An analysis of BitTorrent's two Kademlia-based DHTs", TR07-04, Rice University (2007).

[24] Kurose, James F. and Ross, Keith W., Computer Networking: A Top-Down Approach 6th (Pearson, 2013).

[25] Stoica, Ion and Morris, Robert and Karger, David and Kaashoek, M. Frans and Balakrishnan, Hari, "Chord: A scalable peer-to-peer lookup service for internet applications", ACM SIGCOMM Computer Communication Review 31, 4 (2001), pp. 149--160.

[26] Rowstron, Antony and Druschel, Peter, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems", in Middleware 2001 (2001), pp. 329--350.

[27] Zhao, Ben Yanbin and Kubiatowicz, John and Joseph, Anthony D., "Tapestry: An infrastructure for fault-tolerant wide-area location and routing" (2001).

[28] Steiner, Moritz and Effelsberg, Wolfgang and En-Najjary, Taoufik and Biersack, Ernst W., "Load reduction in the Kad peer-to-peer system", in The Fifth International Workshop on Databases, Information Systems and Peer-to-Peer Computing (2007).

[29] Rogers, Yvonne and Sharp, Helen and Preece, Jenny, Interaction design: beyond human-computer interaction (John Wiley & Sons, 2011).

[30] Cheshire, Stuart and Krochmal, Marc, "NAT Port Mapping Protocol (NAT-PMP)" (2013).

[31] Brooke, John, "SUS: A quick and dirty usability scale", Usability evaluation in industry 189, 194 (1996), pp. 4--7. London.

[32] Rutishauser, Nico. 2015 (6 September). Hive2Hive developer. Personal communication.

[33] Bocek, Thomas. 2015 (15 April). TomP2P developer. Personal communication.

[34] Bocek, Thomas. 2015. The TomP2P Website. Online. Available from: http://tomp2p.net/. Accessed: 23 May 2015.