1 Abstract

Sharing files among machines in a heterogeneous environment is, unfortunately, rarely trivial. It is unusual to find two different machines, be them university provided or student owned, that have the same file sharing products installed and configured to make file sharing simple and easy. Instead students often resort to sending files via email, but this is far from optimal.

It is not a difficult task to set up Samba or AppleTalk to send files among machines, but these protocols have maintenance overheads. Each user who wishes to share files with a specific machine shall need to have a user account and password for that machine to have at least a basic level of privacy. Synchronising user accounts and remembering passwords for many different machines is less than desirable.

This project attempts to solve the problem of moving files quickly and easily from one machine to another. It is the author's opinion that there is a gap in the market for a product that can solve all the pitfalls found in existing attempts to solve this problem.

This document serves to outline the Really Simple Document Sharing System project. It describes progress that has been made to the design and implementation of the product between the start of the project and its completion in May 2008. This report includes relevant research, design documents, test plans, and conclusions drawn from the project. Further ideas for the extension of the project are also proposed at the end of this report.

2 Acknowledgements

The author would like to acknowledge the following people for their assistance during this project.

- **Trolltech** (with special thanks to **Thiago Macieira**)
  Trolltech are the company that produce the Qt framework used extensively in this project. Thiago personally fixed several bugs discovered in the Qt framework that were holding up the development of the project.
- **David Sansome**, Electronics and Computer Science, University of Southampton, UK.
  Dave’s help was invaluable in debugging errors through the unfamiliar Qt library.

3 Contents

4 Chapters

4.1 Introduction

In the age of ubiquitous computing where many people have at least one if not several computing devices, it's a common problem to want to move files from one device to another. Over the years there have been many different products and solutions to cater for this. Some of them are designed to export an entire file system from one machine to another so may be used seamlessly as if it were local. Some have been developed as additional features to other applications, such as instant messaging. These products all try to fill a need in a certain situation; the distribution of a root file system to diskless workstations, or the ability to send a push a file to an instant messaging contact. A scenario that isn't well supported by the existing file transfer solutions however, is the secure exchange of files among users who are communicating face-to-face, especially where the pool of users is constantly changing.

These situations are not uncommon, for example a board room meeting, an open lab environment or a coffee-shop over lunch. The users may be communicating with lifelong friends, or new contacts for the first time and the devices and platforms involved may be quite different. This can make finding a quick and simple means of sharing files and documents rather awkward.

Existing solutions, such as the file sharing systems built into operating systems, tend to require user accounts to be created and maintained on each device for each user that wants access. This does not scale at all. If a person wants to trade files with many different machines, he or she will need an account on each of them. Knowing which machines have accounts and which do not, keeping passwords up to date, and removing obsolete accounts becomes a tedious maintenance overhead. Not to mention giving out local system accounts to everyone a user knows may be a significant security risk. Files could be openly shared by these means, without using accounts or passwords but this removes any level of security from the system. There would be no way to restrict access to shared files or account for transfers that have already taken place. In some situations this may be acceptable but this will not always be the case.

Conversely, systems such as instant messaging applications tend to use centralised user accounts so that once signed in, a user has an identity that can be recognised by any other user in the network. These systems require a permanent infrastructure, and an active connection to that infrastructure at all times to function. In an ad-hoc networking scenario, such a system would be completely unusable. There should also be consideration to the fact that file transfers may be routed through the central infrastructure as well. When two users are in the same geographic location, sending files through the Internet only to have the data return back to the same local network is a complete waste of bandwidth.

From a usability perspective, many users do not want to deal with configuring complicated file sharing products, creating accounts for all their contacts or registering for yet more Internet based services.

There is a need for a new product that is capable of operating without either local or centralised user accounts; one that can be used to securely share files, and restrict access to specific individuals with a minimum of fuss. A system that is capable of operating directly over any local network, especially when the group of peers that a user might want to trade with is constantly in flux.

It is the author's belief that all the components that would be needed to make such a system already exist, but they have not been brought together to solve this problem. This project is an attempt to do just that; to design and implement a piece of software aimed at simplifying the sharing of documents and other files among users.

4.2 Background Research

4.2.1 Existing File Transfer Systems

There are several existing choices for getting files from one client to another, but none of them are flawless. The following is a comparison of the advantages and disadvantages of various methods of file transfer; from the physical exchange of media to fully integrated network file sharing systems.

4.2.1.1 Physical Media

The simplest way of getting a file from point to point is by copying it onto some removable media, and taking it to the destination. This solution has merit, provided at least one of the parties has a device available and with enough free space for the file. This could be done using a USB flash drive, rewritable optical media, or worse, write-once media. Personal experience has taught that this is not always the case.

Another constraint is that both participants would have to be in the same physical location, thus files can't quickly be transferred between clients on different floors or the same building for example.

Finally, the size of the removable media is finite and often considerably smaller than the user's main data drive. This greatly restricts the number of files that can be shared at any one time. There is overhead in duplicating the requested files onto the removable media when requested, and then keeping those duplicate copies synchronised with the originals.

4.2.1.2 Email Attachments

Email has the advantage of ubiquity; everyone has at least one email address, and there's a plethora of clients available for every operating system. It is no surprise that email is one of the most favoured methods to get a file from one place to another. However, using email is the least direct of all the solutions compared here and there is the potential for significant delays receiving the file.

Unless both parties use email accounts on a local mail server, the file may have to be routed through the Internet to the email providers' servers and back again. This is a waste of Internet bandwidth since the file shouldn't need to leave the local network at all. Since the file might be passing through multiple servers en route, it will also be affected by several different policies about what may or may not be transmitted. For example, most mail servers have restrictions on the maximum size and type of attachments. The users will be held to the most restrictive policies of all the servers the message must pass through, and unfortunately for the user, the most restrictive server may not be the first one – the user may fix the problem and resend only to find it rejected somewhere else for another reason.

A less common, but potentially troublesome issue is grey-listing[1]. This is a spam prevention measure in which the mail server only handles email between local or known address pairs. Any other items will be rejected with a message to the sending server to try again after a time delay. The rationale is that spamming mail clients won't understand the request or bother to resend after the time period has elapsed, and given that the delay only occurs once, it isn't too

[1] (Harris, 2003)

much of an inconvenience to the users. Often, these delays cause annoyance for the users however.

4.2.1.3 Network File System (NFS)

NFS provides a major advantage over the previous solutions in that it leverages the existing local network connecting clients to send files directly from one to the other. The host exports one or more directories, which the client can then mount into their own file systems. This provides a straightforward means to access files and applications running on the client machine don't need to even be aware that the files being accessed aren't located on the local machine.

NFS was originally designed to provide file systems for diskless workstations – a client-server scenario. It is given a list of trusted hosts, and will accept all sessions from those hosts. No server-side checking is done to authorise actions requested by the user; this is left up to the applications running on the client machine. As such, NFS is not suitable for exporting file systems to untrusted nodes as would be found in a peer-to-peer file sharing system.

NFS itself has no means of standardising users across multiple hosts. Principally being a means to share UNIX file systems, permissions are stored in the host file system itself as a numeric uid/gid pair. These numbers are not automatically consistent across multiple clients unless the user database is synchronised using some other means. NFS is often deployed in conjunction with NIS for this very purpose[2]. NIS is again highly server-client in architecture, and does not map well onto a peer to peer environment.

The combination of the previous two points means that a file system shared over a peer to peer network couldn't easily be protected by either host-based or user-based permission systems. NFSv4 attempts to fix both of these issues[3], but as yet there is little support for the full NFSv4 protocol in Linux, and almost none in other operating systems. Windows support even for earlier versions of NFS is still far from ideal.

4.2.1.4 Server Message Block/Common Internet File System (SMB/CIFS)

Windows uses the SMB and the more recent CIFS protocols to share files and other local devices. It has been built in functionality in every copy of windows for many years, which makes it readily available for sharing files. As with NFS and unlike the other previous examples, SMB shares make files available for other clients to take rather than explicitly giving files out. This adds a security consideration which is dealt with by requiring clients authenticate themselves, unless of course it's intended for guests to have access to all shared files. A list of authorised users needs to be kept either on each and every host, or on a central "domain server" of which all the clients are members.

In a fluid environment, neither option is especially appealing. A centralised server would impose a hierarchy onto what should be a flat peer to peer system and would be difficult to maintain. An alternative for this, under development for Microsoft's newest version of windows but unfortunately dropped, would have been ad hoc networks called Castles[4]. These

[2] (Westphal, 2001)
[3] (Institute for Advanced Professional Studies, 2007)
[4] (Extreme Tech, 2004)

were to be peer-to-peer Active Directory systems, which would allow for a decentralised user database – exactly what is needed for a document sharing system. Credentials could be passed from machine to machine to allow every node in the network to know about all the other users.

Having separate user databases is the only viable solution, but requires each host to create accounts on their machines for every user he or she wishes to share with. Each client would also need to remember credentials for every machine he or she had accounts on.

There are a couple of other problems with the usability of SMB file sharing. First, browsing SMB file shares over highly latent[5] or low bandwidth links is usually very painful or even impossible and unfortunately this is regularly the case on congested wireless links. Second, sharing individual files, rather than an entire directory is not possible. Files would need to be moved or copied into another directory which could then be shared instead.

4.2.1.5 AppleTalk Filing Protocol (AFP)

AFP is a mature file sharing solution having been developed over many years, and has evolved from a part of the AppleTalk suite into its own fully fledged system.

The Apple ethos has always been to keep things simple, and this is clearly visible in the file-sharing systems it provides. Clients on the network will automatically find each other using a multicast-DNS protocol called Bonjour[6]. Other clients are added to the user's Finder application, so they are easily accessible to the user. Configuring the file sharing is as simple as ticking the relevant checkbox.

Other clients connect using a set of credentials from the host machine, which follows the same pattern as SMB shares (and consequently the same problems).

4.2.1.6 Virtual File Systems

The simplest way to expose files to a user is with a virtual file system. While it's perfectly possible to write a tool that fully understands the network protocols and is able to directly copy files onto the users real file systems, such systems are more awkward to use and as such people prefer not to use them. Allowing the user to open files directly in their applications even when they exist on remote machines is a considerable boost to usability.

There are two basic ways to implement a file system: write a kernel file system module, which is how all concrete file systems such as NTFS or Ext2 are implemented. An in-kernel module has the advantage of high performance but at a steep price. Implementing a kernel module requires intimate knowledge of the inner workings of the operating system, interacting directly with the core APIs. This is difficult and time consuming work, and worse still, the API is subject to change between versions of the operating system. The driver would require constant maintenance to work with newer systems.

[5] (Chen, 2006)
[6] (Apple Inc)

The alternative is to write the file system as a user application using a simplified API. Performance will never be as high as with a file system driver, but such applications are considerably easier to develop and are much more forgiving to changes in the API.

On Linux based systems, including Mac OSX, there is a VFS called "FUSE"[7] which is capable of most if not all the functions of normal file systems. Unfortunately FUSE is not currently supported in Windows platforms. The alternative is to use a Shell Namespace Extension[8], the technology currently used for windows tools including SMB network browser. Namespace extensions are not fully capable of all the low level file system drivers, and may not be accessible to older or non standard applications; however newer applications and the explorer should be able to read files from it without a problem.

4.2.1.7 Conclusions

Each means of file transfer analysed here has its own strengths, but all of them have weaknesses that prevent them from being ideal to use in dynamic environments. The physical media exchange and email solutions suffer from inconvenience; a transfer may not be immediately possible, and may take several attempts. All of the network file-system options require hosts to maintain user accounts for each user they wish to share with, or make shares "world-readable".

Writing a low level file system driver in a project of this size would be completely out of the question; it would take far too much time to produce and would require even more maintenance after the project is over. Writing a VFS in user-space is the logical choice. Even though the Windows VFS is not as powerful as the Linux and Mac option it should still be usable for most applications which is acceptable. If FUSE is ever ported over to Windows based systems, then this could replace the namespace extension, brining the product back to a single code base in the process.

There is a gap in the market for a product that could take the best features of each: the ease of use of AFP, the distributed user database that would have appeared in Longhorn/CIFS, and the complete abstraction of the file system available in NFS. The feature comparison chart, listed in Section 4.4.1, shows these advantages and disadvantages of each described system. The table then shows which features should be supported by the developed product to help ease the file sharing problems.

4.2.2 Development Frameworks

To be useful in the many proposed environments, this piece of software would have to be capable of running on multiple different platforms. Windows still has the lion's share of the market[9] but it is anticipated that a reasonably sized proportion of this software's target audience would be the kind of people who run Linux. Apple Mac based computing devices are also rapidly gaining in popularity. Unfortunately all three of these platforms are vastly different and software designed for one will not run on either or the others without careful consideration during the design and implementation.

[7] (FUSE)
[8] (Rensin, 2004)
[9] See Appendix 6.1

Fortunately there are languages and frameworks in which code can be written to run on a variety of platforms without a need to write separate code bases for each. Java is one option, but the author would prefer to use a C/C++ implementation for performance reasons. There are two main choices for writing cross platform software in C/C++, namely the Gimp Tool Kit (GTK+) and Qt. Both are open source application frameworks born from the Linux world but can be used to write software that runs on any recent versions of Windows, Linux or Mac.

4.2.2.1 GTK+

The GTK+ toolkit[10] was originally designed for the GNU Image Manipulation Program but has since developed into a fully featured cross platform development framework. It provides the necessary constructs to rapidly develop a variety of applications in a selection of different languages. Originally a C library, new language bindings have since been added to support Python and C#. GTK+ has been developed over many years and is a mature product, used in several large Linux applications and Desktop Environments.

One of the downsides to GTK+ applications is that they don't look quite like native applications on each of the operating systems, and feel a little out of place. The Mac port of GTK requires the X11 libraries from Linux to be installed to function. The language API also appears somewhat inconsistent (some methods use underscores and some use mixed case as a naming scheme) and awkward to work with.

4.2.2.2 Qt

Qt by Trolltech[11] is a development platform for C++, supporting all of the major operating systems this project wishes to target. It provides support for low level functionality, such as multithreading, network communication and database interaction which should help for the rapid development of this project. The Qt framework is also supported under Java and on embedded Linux devices such as certain mobile phones and tablets. Unlike GTK+, Qt uses native user interface components where possible, which means applications look and feel like any other written for the respective operating systems.

Qt is entirely object orientated with a clean and consistent API, as well as organised, thorough documentation. Qt is under active development by a commercial entity, which licenses it for both commercial and open-source uses (for which there is also a large community to support development).

Several big name companies are using Qt for their product offerings, including Last.fm's radio client and Google's Google Earth.

4.2.2.3 Conclusions

The use of one of these development frameworks will be essential to avoid writing and maintaining different code bases for each of the platforms to be supported by this project. Qt seems to be an obvious choice owing to its pleasant C++ API, native appearance, and strong community support.

[10] (Mattis)
[11] (Trolltech)

4.2.3 Trust in Decentralised Networks

Without a centralised architecture there's no single authority that can provide information about a peer's identity to anyone it attempts to communicate with. In a decentralised environment, the peers must obtain the information needed to authenticate the identity of others either from those it's trying to communicate with, or from its neighbours.

The Decentralised Trust Management paper by Repantis and Kalogeraki[12] lists some of the issues relating to the different places this information could be stored. Distributed data structures, with as Distributed Hash Tables "[generate] a large amount of network traffic and delay," and while this may be acceptable for some applications, in this situation the overhead may end up of the same order of magnitude as the document transfers themselves; especially if the documents shared are small, and retrieved infrequently. However, storing the information with the peer whose identity is being verified "requires complex operations" to ensure the information is not modified by the peer. The authors propose a compromise, which would be to store the information about a specific peer with a group of other peers[13].

With this project however, users may equally be connected to private ad-hoc networks or open environments with many other users. There may or may not be any other peers to assist in the verification process and so the information will have to be stored with the peer whose identity needs to be verified.

4.2.3.1 Pretty Good Privacy

Pretty Good Privacy[14] (PGP for short) is a suite of cryptographic applications that uses public key encryption algorithms to provide secure communication and authentication. With the PGP system, every user has a public/private key pair of which the public key is distributed to anyone who wishes to communicate with that user. With a centralised approach, this would be via some form of key server, but in a distributed environment, something called a Web of Trust is used.

The PGP model provides a means for storing trust in user identities in a distributed fashion, in a way that precludes tampering which will be very useful for this project. A more detailed description of the PGP Web-of-Trust and its operation is included in Appendix 6.2.

4.2.3.2 Qt Cryptographic Architecture

The security demands for this project will require the use of a library which supports operations for dealing with RSA keys and SSL certificates. Qt uses openssl internally for making SSL but doesn't provide the means for creating new keys and certificates. The Qt Cryptographic Architecture[15] (QCA), an entry into a Qt development competition[16], picks up where Qt left off, and provides a comprehensive API for many cryptographic operations including certificate and key generation. It does this following the Qt style as closely as possible such that it should not be noticeably apparent while using the library that it isn't a

[12] (Repantis & Kalogeraki, 2006, pp. 1-2)
[13] (Kamvar, Schlosser, & Garcia-Molina, 2003)
[14] (Zimmermann P. , 2002)
[15] (Karneges, Bar-Lev, & Hards, Qt Cryptographic Architecture (QCA), 2007)
[16] (Qt Centre, 2007)

core part of Qt. Distribution of the required libraries should also be no more complex than distributing the core Qt libraries.

4.2.4 Peer Discovery

To save the user from having to discover and type in the address of a peer they wish to trade files with, this software must be capable of discovering other peers automatically. In a centralised design there might be a registry that all peers inform of their presence and which lookup queries can be directed. In a decentralised environment peers must search the network themselves for others to trade with. Essentially this means broadcasting either an advertisement of the peer's presence or a query for other peers to respond to.

4.2.4.1 Multicast DNS and Service Discovery

There are two fundamental problems with broadcasting on the local network to find other peers: The first is the level of network traffic generated by lookups and the inherent processing load that brings. The other is that new protocols have to be defined to discover new services. To combat these, Multicast DNS[17] is a protocol being designed by two ITEF working groups (zeroconf and dnsext) to bring DNS-like abilities to a network without a central Name server.

The advantage of using multicast to distribute information among peers is that the traffic is seen by all the peers on the network, but is not processed by any machine that is not interested in it. MDNS clients periodically advertise themselves on the network, and these advertisements are seen by all other peers. Alternatively a peer can make an explicit request to which one or more other peers can respond.

Multicast DNS is the a transport layer, with an interface similar to that of DNS; service discovery comes from another protocol, which is layered on top of MDNS, called DNS Service Discovery. DNS-SD specifies how and when certain DNS queries can be used to leverage a DNS system such as MDNS, to achieve Service Discovery. One type of query can be used to enumerate all instances of a service on a network, and another type can be used to look up the connectivity information for a single instance of that service.

JDNS[18] is an implementation of Multicast DNS written in C and supplied with languages bindings for Qt/C++. The Qt wrapper, QJDns provides a minimal implementation of the Multicast DNS protocol with a familiar API to the rest of the Qt platform. Using the methods put forward by DNS-SD, QJDns can be used to implement Peer Discovery for this project.

[17] (Cheshire & Krochmal, 2006)
[18] (Karneges, JDNS, 2007)

4.3 Project Management

Unlike a more open-ended research project, this one has a fixed goal: to produce a piece of software to accomplish a fixed task. As such it lends itself to a formal approach to project management. It is the intention that by putting effort into the careful design of the application before starting the implementation, problems that would otherwise only be discovered much later can be avoided.

Feature driven development will be used to design and implement this project in separate stages. As described in The Practical Guide[19], this software development process calls for five main stages of development:

- Research and requirements gathering.
- Identification of required features.
- Architectural overview of feature interaction.
- Design of individual features.
- Implementation and testing of individual features.

The research stage has already been completed, and has been included in section 4.2 of this report. Since this methodology doesn't account for testing beyond unit testing of individual features, there will also be some end user testing at completion of the implementation. This will be used to guide the success of the project.

4.3.1 Time Management

In order to run the project successfully, effective time management is of the utmost importance. The first thing done at the start of this project was to identify the main tasks that would need to be completed, and estimate the amount of time required for each one. With the help of a Gantt chart[20] (see Appendix 6.3, Figure 3), the design tasks were each allocated a block of time to be done during the first term.

It is expected that a detailed breakdown of tasks during the implementation stage will be difficult to achieve until the design work has been completed. Therefore a block of time has been allocated for generic implementation at this stage, and once the design has been completed a new time plan will be generated for the remainder of the project.

4.3.2 Prototyping

Throughout the entire design process, and during the early implementation, various features will be prototyped to ascertain feasibility. By trying out different things in advance, the design process can be more informed about how different components can work together. Prototypes can also be used to learn new frameworks and architectures before having to use them as a part of the project implementation. One thing that will be prototyped during the development will be Graphical User Interfaces, using the Qt framework.

[19] (Palmer & Felsing, 2001)
[20] (Gantt, 1910)

4.3.3 Version Control System (VCS)

A very important tool for software development is an adequate version control system which can be used to track changes to documents and code. From the offset of this project, Subversion[21] will be used to store both source code and design documents.

The use of Subversion provides a regular backup of code and allows a developer to quickly revert back to previous version to undo mistakes, or create branches to test out new and dramatic changes without fear of breaking the software.

[21] (CollabNet, Inc., 2006)

4.4 Design

The following section of this report documents the stages of the design process, and the decisions made at each stage.

4.4.1 Feature Comparison

Following on from the background research into existing file transfer systems, the following table was produced. It compares the features available from each of the different reviewed systems and those that should be added to this product.

Colour has been used to help easily identify the results in the table below. Green indicates a desirable outcome, red for an undesirable one and orange for a compromise.

Table 1 - Comparison of features available in existing file transfer systems and proposed features for the new system.

Feature	Flash	Email	NFS	Samba	Apple-Talk	RS-DSS
General						
Availability	near line	Online	Online	Online	Online	Online
Transfer speed limited by	Physical movement	Internet bandwidth, delays	Network	Network	Network	Network
Maximum share size	Flash size (~5gb)	~20mb	No limit	No limit	No limit	No limit
Authentication						
Can make use of existing user accounts?	n/a	Yes	No	No	No	No
Decentralised user accounts	n/a	Yes	No	No	No	Yes
Consistent user accounts across the network	n/a	No	Not by default	Yes	Yes	Yes
Assurance of host identity	Yes	No	No	Yes	No?	Yes
Security						
Optional traffic Encryption	Driver dependant	Yes	Third party	No	No	Yes
Access Control Lists	Physical	n/a	Basic	Yes	Basic?	Yes
Ease of use						
Auto-discoverability	n/a	No	No	Yes	Yes	Yes
File system interface	Yes	No	Yes	Partial	Yes	Yes
User account creation	n/a	Manual	Manual	Manual	Manual	Auto
Share individual files	Yes	Yes	No	No	No	Yes
Share entire directories	Yes	No	Yes	Yes	Yes	Yes
Usable on all major Operating systems	Yes	Yes	Third party	Built-in	Third party	Third party

4.4.2 Specification

The specification, developed from the features comparison chart describes the required features that the software developed as part of this project must have. The specification forms the basis for the Feature Driven Development process and from it, the list of features can be deduced.

The specification can be found in Appendix 6.4.

4.4.3 Software Components

The software will consist of a single server application which runs all the time in the background, and several different frontends which can be used to access the network via the server.

4.4.3.1 Server Components

The server application will be formed of several different components. These will be initialised and connected together by a singleton application class, representing the core of the application. The rest of this section describes these components and what they do. Appendix 6.5 (Architectural Overview) shows how each one will link together in a graphical form.

4.4.3.1.1 Database/Configuration

The application will need to store some information persistently across sessions. This includes configuration settings, but also a list of all the peers the user has ever encountered, and access restrictions for their shared resources. As part of the framework, Qt provides an abstract interface to SQL databases and specifically a SQLite[22] implementation. Given the way in which this application will need to extract data from persistent storage, cross-referencing information from different lists, an in-memory SQL database is an optimal solution.

The following tables will be included in the database:

- Configuration.
  This table will store miscellaneous bits of information, such as the user's chosen name, their keys, and which port the server will listen on.
- Shares.
  This table will store the paths shared by the user, and the names under which they have been shared. This list will be displayed when another user attempts to enumerate the shared resources.
- Peers.
  This table will store the names and keys of all the other users that are ever encountered. The public key will be used to identify a connection from another peer and can be used to lookup access control information or verify introductions. Each user will have a trust metric associated with them which defines whether the user has had the peer's identity has been accepted and whether the peer may vouch for others.
- Access Control Lists.
  This table will store the access restrictions for each share which define whether

[22] (Hipp, 2008)

another peer may have access to a shared resource and if so, what actions may be taken (e.g. list contents, create new files, etc).

- Introductions.
  This table will store a list of the signatures created by other peers and sent to the user to vouch for his or her identity.

Each of the database tables are represented by a Singleton class which can be used to store and extract information in a useful format throughout the rest of the application. For example, the "configuration" class (the Settings Manager) will deal with Variant data types while the Known Peers class which wraps the "peers" database table will operate directly with Public Key objects. This will reduce the amount of manual conversions needed throughout the code and also abstracts the means by which data is stored in the database.

The full database schema, including type definitions and relationships between tables has been included as Appendix 6.6.

4.4.3.1.2 HTTP Server

Communication between frontends and the server component as well as between the server components running on different machines will be done using the Http transport protocol. This is a relatively straightforward to implement text based protocol which supports transporting almost any type of data.

The server will be custom built following just enough of the HTTP specification in order to achieve what is needed for this project. (A full http implementation would be too much work for a project this size, and would offer features that are in no way needed here). The server will be aware of the content it is serving and look for specific URLs and process them internally. For example, POST requests sent to the path "`/rpc`" will be regarded as being Remote Procedure Calls, and will trigger special handling.

In order to handle many requests simultaneously, each connection to the server will be handled in one of several pooled request handlers. Qt provides a set of classes to operate a thread pool which should make this easy to implement.

The use of a standard protocol also opens up opportunities for distributing the software. Implementing an HTTP server means that another user who doesn't have the software should be able to point their web browser at any device already running the software and download a copy. An internet connection would then not be required to acquire the software, making it much more reachable to first time users.

4.4.3.1.3 Remote Procedure Call

The mechanism by which clients and servers communicated will be via a remote procedure call system. A client application will construct a message that requests the server component execute a method, such as listing all of the peers it can see on the network, and return the results. The server component does all the communication across the network between peers, so the client application will send all its requests to the server component, which decides whether it should be processed locally (e.g. a peer scan) or proxied to the intended peer (e.g. a file read operation).

The RPC requests and their subsequent responses will be marshalled into messages using the XML-RPC protocol. These will then be sent across the network using the HTTP protocol and processed by the HTTP server described above. The XML-RPC protocol is advantageous in that it uses data types that closely mirror the internal types used by Qt including scalar types but also arrays and maps. Qt also provides built in XML support, so a translation between native objects and messages should be fairly trivial.

Remote method calls will be specified by a procedure object name, and a method name. There will be one procedure object for each category of methods that can be called, such as “File” operations or “Security” operations. These will be singletons and register themselves with a known name when they are created. The logical grouping acts to keep related code together but also provides a means for security checking: a peer whose identity has not been verified may be prevented from executing methods on any object other than the Security object, which is responsible for validating and providing introductions. When a request is received, the correct procedure object will be looked up using the name in the request, and the method will be invoked using Qt’s meta-object mechanisms for dynamic method invocation.

The request object will take a list of parameters, and the response object will also be able to return a list of values to maximise the usability of the system.

4.4.3.1.4 Service Discovery

The Service Discovery will be done using the QJDns library, and open-source, minimal implementation of the Multicast DNS protocol. This component will initialise the MDNS library, and then begin advertising itself to other peers on the network. The component will be responsible for discovering which IP addresses are available on the local machine, and identifying which address to advertise as the most-reachable address, such that the largest number of peers will be able to connect to it. A detailed explanation of how this process will work is included in Appendix 6.7.

Any part of the application will then be able to use this component to lookup the names of other peers on the network, or look up the connection information for any individual peer. Once the connection information has been queried for another peer on the network, this component will then be able to select the best means of communicating with the peer, be that over ipv6 or ipv4.

4.4.3.1.5 Certification/Security

An implementation of the PGP Web-of-Trust (as described in Appendix 6.2) will be used to identify peers attached to the file sharing network. Each user on the network will have a unique identity formed of a public/private key pair and a name. These will be used with an HTTPS implementation in the HTTP server component to ensure encrypted communication, and trust in the identity of the parties involved in communication. The public key and peer name will be wrapped in a self-signed SSL certificate which will be sent as part of the SSL handshake during connections. At the other end of the connection, the remote peer will be able to see the name and public key of the user that it is connecting to. The remote peer will be able to check the public key against its database to determine the level of trust it has in the owner of that key.

The user will be able to elect other peers it knows about as trusted introducers. When a connection is established with an untrusted peer, an exchange of introductions will take place and if the peer can present any introductions made by a trusted introducer then their identity will be accepted. Once the identity of t new peer has been established, it can be marked in the local database as trusted or possibly even as a new trusted introducer and communication can continue.

If no trust can be established with another peer because they cannot present an introduction from a trusted introducer, the user will be given the option to introduce this peer themselves. A new introduction will be created and sent to the peer that it can use to vouch for its identity at a later time.

Due to the complexity of these operations, flow charts mapping out the behaviour for each possible path have been included as Appendices 6.8 and 6.9, which show the peer verification processes carried out on the host and client sides of the connection respectively.

The key generation and signing operations will all be done using QCA, but the secure socket implementation is a part of Qt. Unfortunately both of these use different objects to wrap keys and certificates, so the code will have to contain lots of conversions between the types used by each library.

4.4.3.1.6 User Interface

The server application will be responsible for providing the user interfaces necessary for the user to configure the application to their own desires. This includes which paths are to be shared, and who the resources are to be accessible to.

When the user first launches the application, a wizard will be presented which requests enough information for the user to start using the software; namely a recognisable username for themselves, and one or more directories to share. This wizard will also be responsible for initialising the user's identity on the network by creating the necessary keys.

Once the application is running, there will be a system tray icon through which the user will be able to exit the application, or reach the other two main parts to the GUI. The first of these will be a configuration dialog, which gives the user complete control to customise the software. This dialog will allow the user to add and remove shared resources, look at the lists of known peers and received introductions, apply security restrictions to shares, and configure miscellaneous settings relating to the application. The other main GUI component will be a simple log viewer, which will display information about what the server application is doing.

Finally, there are occasions where the server application may need to popup prompts to the user to request an action to take. This may be needed when a peer is encountered whose identity cannot be verified, or when a peer attempts to communicate with a changed identity (which may happen if the peer has reinstalled the software, or someone attempts to impersonate another user).

4.4.3.2 Client Components

Different platforms have different user interfaces and platform APIs, so different client applications will need to be written for each of these. To avoid duplicating code between the different frontends, all the common code will be made part of a shared library which will be linked in to all client applications.

4.4.3.2.1 Shared Library

The shared library will be linked the client applications, but also the server application. The library will include the following components:

- Logging.
  A generic log for the applications to report their progress, and associated viewers.
- RPC Messaging.
  The code needed to convert internal data types to XML-RPC messages and back again.
- Network Client.
  A central place for clients to make requests to the server component running on the local machine and for server components to make requests to each other.
- Download Manager.
  A set of classes for multi-threaded and recursive downloads of shared resources.
- Shared Resource Navigation.
  A set of classes which implement a browser history, and provide a way to tell if a client can move forwards or backwards through the history, or up to a parent directory from the current location.

The recursive download feature will be non-trivial to implement, so a flow diagram mapping the behaviour of the download manager has been drawn up to help during the implementation stage. This has been included in Appendix 6.10 for completeness.

4.4.3.2.2 File system Frontends

There are no virtual file system frameworks common to all three major operating systems, so multiple frontends will need to be developed. For Linux and Mac OS X systems, FUSE will be used to implement an actual file system driver. Under Windows platforms, the virtual file system will be implemented using the Shell Namespace.

Both implementations will expose a directory hierarchy which the user can navigate through. The low level file operations such as open, close, and read will also be implemented so that applications can read or copy files onto the local file system.

The frontends will be responsible for translating calls from the operating system to interact with the file system into RPC calls to be sent to the server application. The responses to these calls will be used to render the current view of the virtual file system using the VFS APIs.

4.4.4 Class Diagrams

For the server and shared library, full class diagrams have been produced which show in detail how the components have been divided up into individual units, and how these units will work together to accomplish the task of each component.

The server application class diagram is included in Appendix 6.11 and the shared library class diagram is included in Appendix6.12.

4.4.5 Updated Time Management

With the design fleshed out the time management for the rest of the project has been updated. A second Gantt char has been included in Appendix 6.3 (Figure 4) depicting the amount of time allocated to each feature of the design.

4.5 Design Revisions

During the course of the implementation, parts of the design had to be revisited after unanticipated difficulties with the original design plan.

4.5.1 Threading Issues

While developing the network server to utilise a shared thread pool to respond to incoming requests, issues with the threading support in the application became apparent. If the user attempted to close the application while there were still active connections open, the worker threads would not properly complete and the application would not terminate. In order to save on the performance overheads of creating a new connection for every single request, the Http server component and network clients both support persistent connections that can be reused for multiple requests. This means that there will often be running worker threads when the user attempts to quit the server.

After some research on the subject of the Qt networking system, it was discovered that due the asynchronous nature, the request handling did not need to be threaded at all. The event loop system is capable of handling as many requests concurrently as there are resources on the machine to handle them without the need for additional threading.

In light of this, the multithreaded handling of network requests was removed completely.

4.5.2 Complexity of the Virtual File Systems

The shell namespace extension was prototyped using an example from the Windows SDK before embarking on the full design for this feature. After a week of experimenting with the API and making no progress toward a viable implementation for this project it was decided to abandon this feature entirely.

The complexity of the Windows Shell Namespace Extension API is so great, that there was no way it could have been used to implement a file system implementation, even just offering basic functionality, in the short time period allotted. A risk assessment was carried out to determine whether any additional time could be dedicated to this feature, but it was determined that doing so could jeopardise the rest of the project.

Based on the experience with the Windows VFS API, and given that FUSE is not cross-platform and a separate client application for Windows would have to be written regardless, it was also decided that FUSE support should be dropped as well. Instead effort would be focused on client applications which could be written to run on all platforms.

4.5.3 Client Applications

Two client applications were designed as replacement features for the Virtual File System frontends: a graphical client and a command line client. Each of them would be capable of the following:

- List other peers on the network.
- List a peer’s shared resources.
- Download shared resources recursively from a peer into a directory chosen by the user.

Class diagrams are included in Appendix 6.13, which highlight the simplicity of the code needed to create client applications thanks to the common codebase in the shared library.

The graphical application makes use of the history class from the shared library to allow the user to move backward through the list of places they have visited, as well as moving up through levels in the remote file system. This interface closely follows that found on the file browsers found in most operating systems. In order to display the items from the remote file system, a custom model is used to store the information about each file. An Icon Provider displays either a Folder or File icon depending on the type of the file, so that the user knows whether double clicking on an item (a common means of interacting with a file system browser) will open a subdirectory or attempt to download the file.

4.5.4 Updated Time Management

Taking into account the changes to the design, the implementation time plan has been updated to reflect the time lost and estimated time requirements to complete the replacement features. This is included in Appendix 6.3 (Figure 5)

4.6 Test Plan

As part of this project, testing will be undertaken in several different stages.

4.6.1 Unit Testing

Each class that comprises the software will carry out a single, focussed task and unit testing will be carried out to ensure that these pieces of code perform correctly. Tests functions will be written to run each of the class methods to ensure the correct result.

A mixture of black-box and white-box methodologies will be used in part of these unit tests. Black-box tests will be used to ensure a selection of inputs produces the correct output data. White-box tests will be used to trace the logic paths through the class methods and strategically test for specific boundary cases.

The unit tests will be kept permanently in a separate development project away from the main source tree. By doing this, the source tree can be kept tidy, but the unit tests can be used for regression testing.

The unit tests will all be automated using the QtTestLib framework, which will make creating, running and aggregating the results of the unit testing straightforward. The unit testing framework provided by Qt will also make it possible to test the Signals and Slots system used in Qt based code for asynchronous behaviour, which will be used extensively with the networking and graphical user interface code and might otherwise be very difficult to properly test.

4.6.2 Regression Testing

Whenever bugs are discovered and fixed, the entire collection of unit tests in the project will be rerun to ensure no erroneous behaviour is reintroduced into the software. Regression testing will help ensure that the source trunk is always kept free of known bugs.

4.6.3 User Testing

Towards the end of the project the software will be packaged up for different platforms and distributed to users for beta testing. Fellow students will be recruited to install and use the software for a period of time, recording any buggy behaviour they discover.

In addition to finding bugs, the user testing will also help to uncover weaknesses in the user interface and inline documentation.

In order to keep records of user feedback, an off-the-shelf bug tracking web application will be used. All the beta testers will be given access to post new bugs to the system. Because the primary target audience of this software are technically proficient users, it is expected that most of the testers will be familiar with such systems, and that it will be the cleanest and simplest way to gather information.

Beta testing is important because the developer of a piece of software is likely to only test certain use cases whereas other users are more likely to have different usage patterns and therefore trigger bugs not found by the developer.

4.7 Test Results

4.7.1 Unit and Regression Testing

During the development of the software components, various base classes have been unit tested to ensure correct operation. One test class has been written for each component that needs to be tested in the software products. These have been grouped together into a single application which runs all of the tests and outputs the results.

The test output produced is long and verbose so will not be included in its entirety, however a small subset of the test output has been included in Appendix 6.14 to highlight the kind of information output by the test suite.

While not all the unit tests have been written yet and work will continue on this toward the end of the project, at the present time all unit tests pass successfully.

4.7.2 User Testing

At this time, only a small amount of user testing has been undertaken, with plans to continue testing over the next two weeks. The bug tracker has been used to record both bugs recently discovered by the developer and those reported to the developer by several users. A screenshot of the tracker has been included in Appendix 6.15 and shows the bugs submitted to the developer at the time of writing.

User testing is being carried out by fellow students who are eager to try the software for themselves on both Windows and Linux platforms. So far problems have been identified where two users attempt to use the same name and the application does not alert the users to the collision, and also several usability issues with the graphical user interface. These issues will be fixed before the end of the project.

It is expected that the user testing will uncover more issues in the near future and hopefully these will also be fixed before the project ends.

4.8 Conclusions

As the project is rapidly nearing its completion, this section of the report looks at what has been achieved and what still remains to be done.

4.8.1 Outcomes

There are three software products that have been produced for this project: a single graphical server application that will be run on every machine that is a part of the network, and two client applications which can be used to pull files off of the network. All of these applications have been written to run on any operating system supported by the Qt framework which includes Windows, Linux and Mac OS X based computers. Given that these operating systems run on the vast majority of computers found in everyday use, this software should be usable by almost anybody. Appendix 6.16 contains a selection of screenshots of the software running on several different operating systems and environments.

The software exhibits the following features:

- Guiding the user through the setup of the software the first time it is run, for maximum ease of use to new users. (Figure 15 and Figure 16)
- Automatically selecting the network addresses on which to announce its presence.
- Automatically discovering other peers on the network.
- Automatically determining the level of trust in the identity of other peers where possible based on the trust in mutually trusted neighbours.
- Manual configuration of the trust in other peers. (Figure 18)
- Configuration of shared resources, and access permissions to each resource. (Figure 17 and Figure 19)
- Browsing shared resources on the network. (Figure 23)
- Downloads of single files from the network to a specified directory on the local machine. (Figure 22)
- Recursive downloads of shared directories from the network to a specific directory on the local machine.

4.8.2 Completeness

Where possible, the specifications have been followed to produce this software but in some cases complete adherence to the specification was not achieved.

4.8.2.1 Redistribution

Item 7 of the specification called for a means of distributing the software to new users directly through the system so that new users without an Internet connection would be able to download a copy of the software through another peer.

This can only be partially realised for a number of reasons. Primarily, the software is cross platform, and while a single source package could be compiled and run on any system, new users would want a precompiled package, and there would need to be one of these for each operating system. Unfortunately, given the size of all the dependencies and the number of platforms that need to be supported, this would mean the redistributables would be in the order of magnitude of 100mb..

There is also a circular reference problem inherent in trying to redistribute the software through itself, and that is how to construct three or four software packages that all include each other. While this is likely a problem that could be solved, it has not been for this project.

4.8.2.2 Connection Security

Item 17 of the specification calls for the option to send files through either unencrypted or encrypted channels. The software actually encrypts all communications whether they are metadata about transfers, such as directory listings, or actual file transfers.

The justification for this is that the primary use of this software will be to transfer documents which tend to be small in size. The impact on the processor to encrypt small files should not be too great and in return the user gets a much greater level of privacy and security.

4.8.2.3 Optional Features

The optional features listed in the specification have not been implemented due to time constraints.

4.8.3 Problems Faced

During the implementation stages of the project, multiple problems were encountered which caused a significant loss in development time.

4.8.3.1 Complexity of the Virtual File systems

Due to inadequate levels of prototyping of the virtual file systems, the true complexity of these parts of the project was completely underestimated. In retrospect the sheer volume of work needed to implement either the Windows Namespace Extension or Fuse module could well fill the time allocated to this entire project alone.

One week of development time was lost to futile experimentation with the VFS implementations before it was decided to drop support for these features in the software and design an alternative. More research and prototyping into these areas during the design stages might have prevented this situation.

4.8.3.2 Bugs in the Qt Framework

A short time into the implementation stages of the project, Trolltech released developer pre-release versions of a new series of their Qt product. The new 4.4 series offered many new features which would have been useful for this project, such as Thread Pooling and a new Http Client implementation. There was a trade-off made, between using the new framework with the risk that things may not work properly, or using a stable version of the framework and having to re-implement the same features Trolltech were already working on and would release before the project completed.

It was decided to accept the risk of using the unstable framework, as it was perceived that more progress could be made this way. There were bugs encountered in the networking stack which lead to a lack of progress for a short while during development. Nevertheless, it is perceived that the benefits of having used the new features in the pre-release code far outweigh the lost time due to the bugs.

4.9 Further Work

While the project is nearing completion, the software product is by no means considered finished. There are several aspects which could be worked on to greatly improve the product. Some work will be carried out after this report is finalised, but there will be more work that can be done that will be outside the scope of this project.

4.9.1 Before the Project's Completion

At this time, not all file operations are checked against the access controls put in place. Before the project is complete, work will be done to ensure that all operations implemented so far validate the connected peer's privileges.

During the development, the environment was stable and packet corruption was not an issue. In a real world environment this will not be the case and file data may be corrupted as it is transferred over the network. To combat this, checksums will be added to ensure the file data is received successfully, and if corruption is detected, the chunk will be requested again.

4.9.2 Beyond the Project

The software is currently lacking any end user documentation other than the descriptive text present in the graphical user interface and the readme file intended to help the user install the software. A full user manual should be a high priority task for the future.

Upload support has not been implemented, as it was felt that the primary usage scenario for this software would be for one peer to download what they wanted from a list of files shared by another. It might improve the usability of the software if users could also upload files to a peer's shared directory as well. This feature will be left as something which could be added in the future.

As described previously, virtual file system front ends to the network were part of the original design and were considered to be a very important feature for the usability and transparency of the software. It is unfortunate that these could not be implemented this time, but hopefully could be included in a future version of the software.

Testing has shown that the file transfer performance is somewhat less than desirable, achieving speeds of between approximately 600-1,000 kilobytes per second. While the documents being transferred are small this is not disastrous, but some profiling could be carried out to determine where the performance bottlenecks are and increase the throughput.

As a final thought, the introductions system in the software allows a user to automatically trust a new peer if their identity has been signed by at least one other peer directly designated as a trusted introducer by the user. An extension to this would be to allow chains of trusted introducers where a user can allow a trusted introducer to vouch for other trusted introducers who can themselves vouch for the identity of peers. This might be implementable as defining the amount of trust in each introducer as the depth to which they can introduce new introducers; for example an introducer with a depth of one could introduce introducers but those could not vouch for anyone else. Each time a new introducer is automatically introduced, the depth is decremented by one. This would allow larger communities to grow with fewer manual introductions of peers. This design has not yet been thought out, but could be an improvement for a future version.

5 References

Apple Inc. (n.d.). *Bonjour*. Retrieved November 2007, from Developer Connection: http://developer.apple.com/networking/bonjour/index.html

Chen, R. (2006, April 7). *Computing over a high-latency network means you have to bulk up*. Retrieved December 2007, from The Old New Thing: http://blogs.msdn.com/oldnewthing/archive/2006/04/07/570801.aspx

Cheshire, S., & Krochmal, M. (2006, August 10). *draft-cheshire-dnsext-multicastdns-06.txt.* Retrieved October 2007, from Multicast DNS: http://files.multicastdns.org/draft-cheshire-dnsext-multicastdns.txt

Cheshire, S., Aboba, B., & Guttman, E. (2005, May). *Dynamic Configuration of IPv4 Link-Local Addresses.* Retrieved March 2008, from Internet Engineering Task Force: http://www.ietf.org/rfc/rfc3927.txt

CollabNet, Inc. (2006). *Subversion*. Retrieved October 2007, from Tigris.org - Open Source Software Engineering Tools: http://subversion.tigris.org/

Extreme Tech. (2004, May 7). *Longhorn In-Depth*. Retrieved November 2007, from Extreme Tech: http://www.extremetech.com/article2/0,1697,1588161,00.asp

FUSE. (n.d.). *Filesystem in Userspace*. Retrieved December 2007, from http://fuse.sourceforge.net

Gantt, H. (1910). Work, Wages and Profit. *The Engineering Magazine* .

Gurevich, M. (1961). *The Social Structure of Acquaintanceship Networks.* Cambridge, MA: MIT Press.

Harris, E. (2003). *The Next Step in the Spam Control War: Greylisting.*

Hipp, D. R. (2008). *SQLite Home Page*. Retrieved November 2007, from SQLite: http://www.sqlite.org

Institute for Advanced Professional Studies. (2007). *NFSv4: Overview of New Features*. Retrieved November 2007, from Institute for Advanced Professional Studies: http://www.iaps.com/NFSv4-new-features.html

Kamvar, S. D., Schlosser, M. T., & Garcia-Molina, H. (2003). The EigenTrust Algorithm for Reputation Management in P2P Networks. *Proceedings of The Twelfth International World Wide Web Conference.* Budapest, Hungary.

Karneges, J. (2007, March 18). *JDNS*. Retrieved November 2007, from Delta XMPP Project: http://delta.affinix.com/jdns/

Karneges, J., Bar-Lev, A., & Hards, B. (2007). *Qt Cryptographic Architecture (QCA)*. Retrieved December 2007, from Delta XMPP Project: http://delta.affinix.com/qca/

Mattis, P. (n.d.). *GTK+ Overview*. Retrieved November 2007, from GTK+ Homepage: http://www.gtk.org

Net Applications. (2008, March). *Operating System Market Share (March 2008).* Retrieved April 16, 2008, from Net Applications: http://www.netapplications.com/

Palmer, S. R., & Felsing, M. (2001). *A Practical Guide to Feature-Driven Development, 1st Edition.* Pearson Education.

Qt Centre. (2007). *The Qt Centre Programming Contest 2007 Winners*. Retrieved March 2008, from Qt Centre: http://www.qtcentre.org/contest-first-edition/winners

Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G. J., & Lear, E. (1996, February). *Address Allocation for Private Internets.* Retrieved March 2008, from Internet FAQ Archives: http://www.faqs.org/rfcs/rfc1918.html

Rensin, D. (2004, January). Create Namespace Extensions for Windows Explorer. *MSDN Magazine* .

Repantis, T., & Kalogeraki, V. (2006). Decentralized trust management for ad-hoc peer-to-peer networks. *ACM International Conference Proceeding Series , 182*.

Trolltech. (n.d.). *Qt*. Retrieved November 2007, from Trolltech: http://trolltech.com/products/qt/homepage

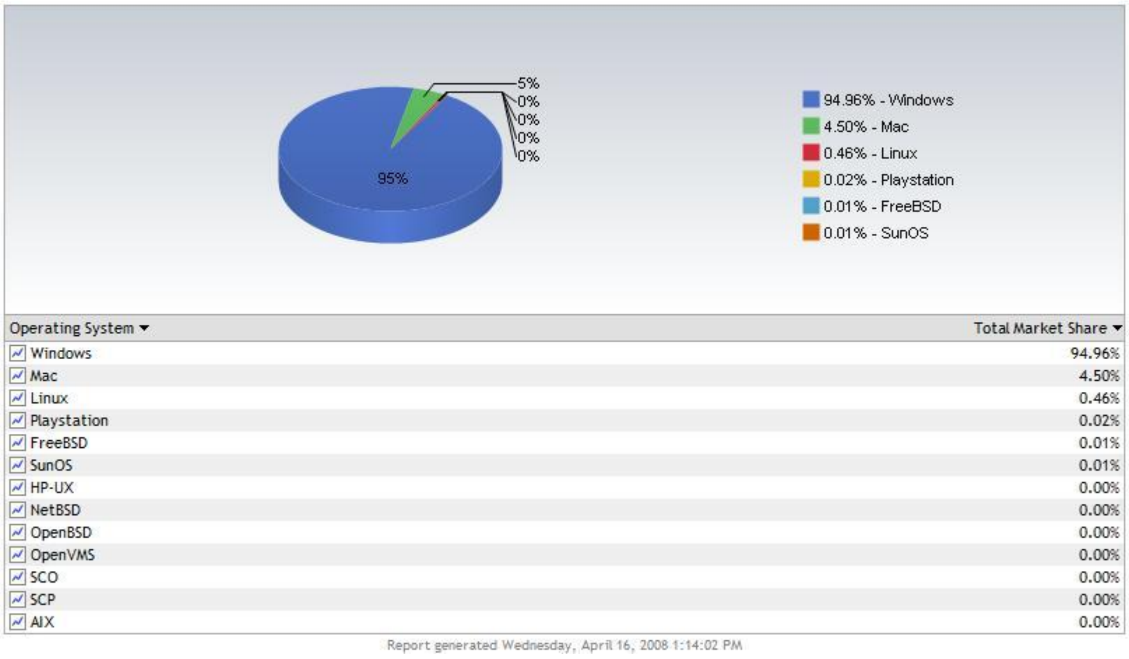
Westphal, K. (2001, January 22). NFS and NIS Security. *Security Focus* .

Zimmermann, P. (2002). *PGP Corporation*. Retrieved November 2008, from PGP Corporation: http://www.pgp.com

Zimmermann, P. R. (1994). How does PGP keep track of which keys are valid? In *The Official PGP User's Guide.* Cambridge, MA: MIT Press.

6 Appendices

6.1 Operating System Usage Statistics

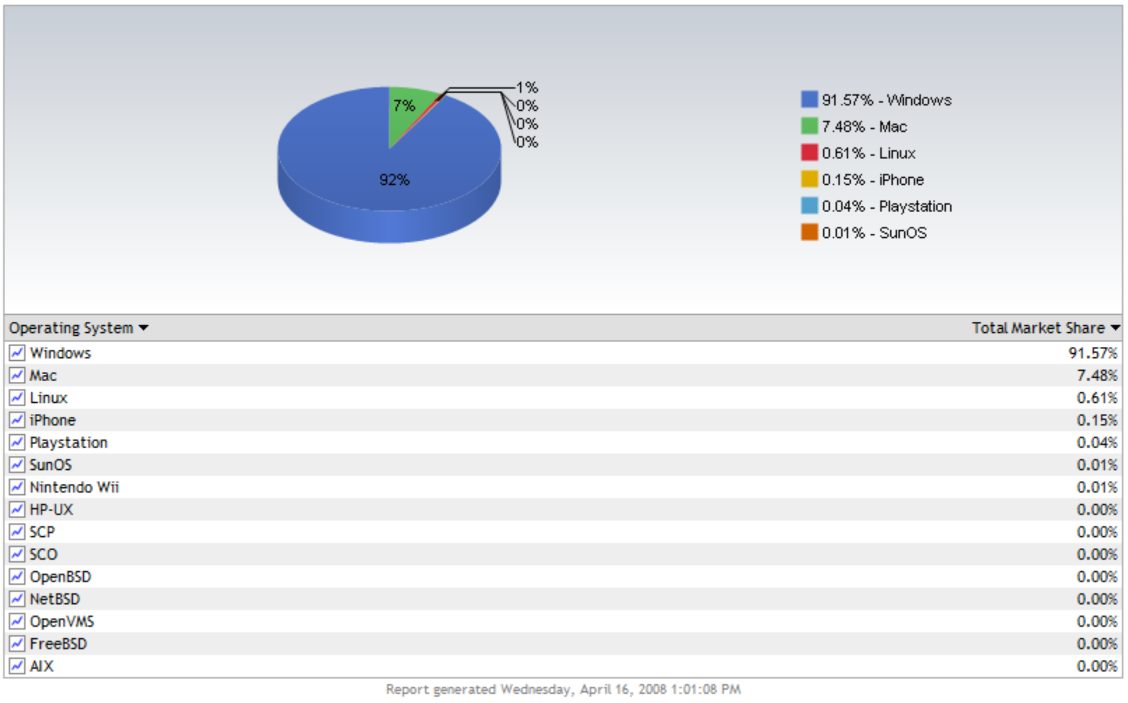


- 🟦 94.96% - Windows
- 🟩 4.50% - Mac
- 🟥 0.46% - Linux
- 🟨 0.02% - Playstation
- 🟦 0.01% - FreeBSD
- 🟧 0.01% - SunOS

Operating System ▾	Total Market Share ▾
Windows	94.96%
Mac	4.50%
Linux	0.46%
Playstation	0.02%
FreeBSD	0.01%
SunOS	0.01%
HP-UX	0.00%
NetBSD	0.00%
OpenBSD	0.00%
OpenVMS	0.00%
SCO	0.00%
SCP	0.00%
AIX	0.00%

Report generated Wednesday, April 16, 2008 1:14:02 PM

Figure 1 - Operating System Market Share as recorded by Net Applications (March 2006)



- 🟦 91.57% - Windows
- 🟩 7.48% - Mac
- 🟥 0.61% - Linux
- 🟨 0.15% - iPhone
- 🟦 0.04% - Playstation
- 🟧 0.01% - SunOS

Operating System ▾	Total Market Share ▾
Windows	91.57%
Mac	7.48%
Linux	0.61%
iPhone	0.15%
Playstation	0.04%
SunOS	0.01%
Nintendo Wii	0.01%
HP-UX	0.00%
SCP	0.00%
SCO	0.00%
OpenBSD	0.00%
NetBSD	0.00%
OpenVMS	0.00%
FreeBSD	0.00%
AIX	0.00%

Report generated Wednesday, April 16, 2008 1:01:08 PM

Figure 2 - Operating System Market Share as recorded by Net Applications[23] (March 2008)

[23] (Net Applications, 2008)

6.2 PGP Web of Trust

Each user in the web has an RSA key-pair. The public part may be signed by other users, which binds the key to the user and acts as a confirmation of their identity. The signatures are stored with the user whose key has been signed so that they can provide a list of all their signatures when trying to communicate with another user.

Each user may nominate select individuals to be trusted introducers, people the user trusts to vouch for the identity of others. When a user initiates communication with a previously unknown individual, he or she can request a list of the signatures from the new user. If any of the signatures were made by a trusted introducer, the user can accept the new peer is who they claim to be.

There are currently no known reliable attacks against RSA so as long as the private part of each user's key remains secret, the web cannot be compromised. That is to say that no user will be able to tamper with the signatures created for them by other users, or falsify new signatures to appear to have been created by another user.

As each user collects more and more signatures created for them by other users, the likelihood that they will be recognisable by another user is increased, as described by Zimmerman in the PGP Manual[24]:

"As time goes on, you will accumulate keys from other people that you may want to designate as trusted introducers. Everyone else will each choose their own trusted introducers. And everyone will gradually accumulate and distribute with their key a collection of certifying signatures from other people, with the expectation that anyone receiving it will trust at least one or two of the signatures. This will cause the emergence of a decentralized fault-tolerant web of confidence for all public keys."

In accordance with the Small World Experiments[25], if the web becomes large enough there should be a point where peers should have enough signatures that their identity can be trusted by anyone else.

Finally, it is worth noting that the network of peers who know and trust in each other's identities is a social one, rather than a physical computer network. The signatures accumulated by a user can be carried from one computer network to another. A user will have a single identity regardless of where they may be.

[24] (Zimmermann P. R., 1994)
[25] (Gurevich, 1961)

6.3 Time management

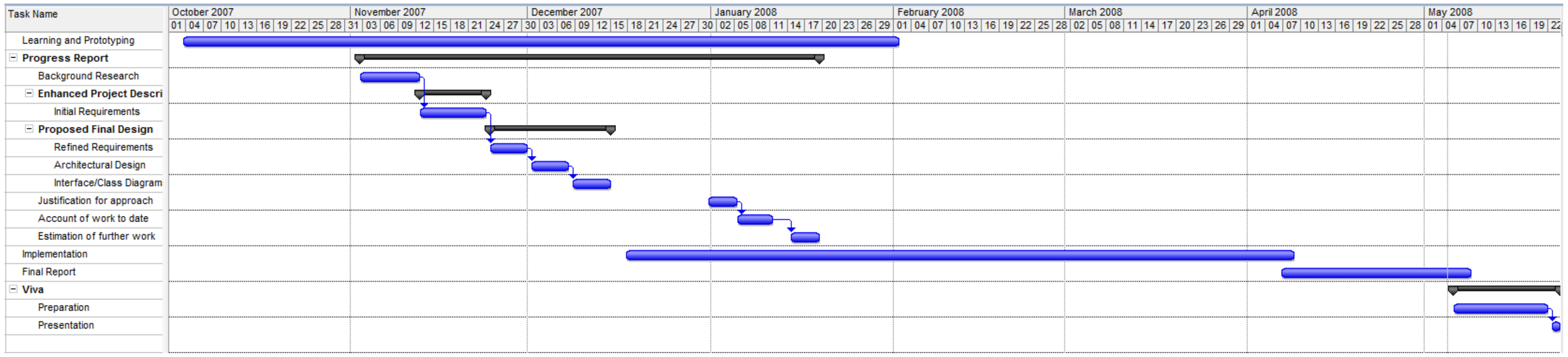


Figure 3 - Gantt chart plotting estimated time usage through the design stages of the project

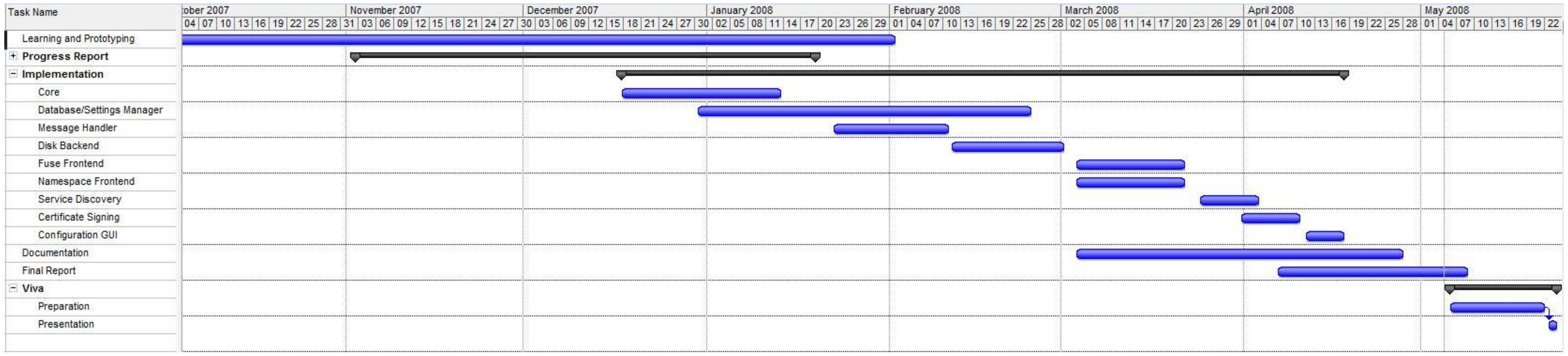


Figure 4 - Updated Gantt chart plotting estimated time usage through the implementation stages of the project.

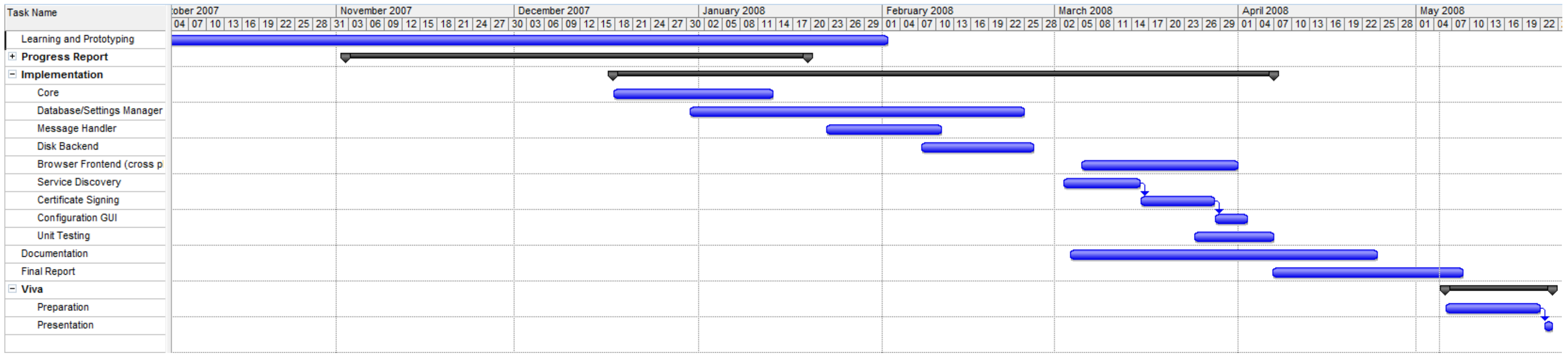


Figure 5 - Updated Gantt chart plotting estimated time usage through the revised implementation stages of the project.

6.4 Requirements and Specification

6.4.1 Definitions

Term	Meaning
User	A person wanting to share resources with other people on the network.
Client	The software used by users in order to share resources with each other.
Resource	A file or directory made available for sharing over the network.
The system	The collective of all software, and protocols being designed as part of this project.
The network	The collective of all users who use the system in order to share resources with each other over the local network.

6.4.2 Core functionality

1. The system should be cross platform.
2. The system should let users share individual files, or entire directories with other users of the network.
3. The user should have a single identity throughout the network.
4. The system should impose no artificial limits on the size or number of files being shared by the user.
5. The system should operate on the local ipv4 network subnet.
6. The system should use IP multicast, rather than broadcast in order to keep network bandwidth usage as low as possible (i.e. not sending packets to network segments with no clients connected).

6.4.3 Usability

7. The system should provide new users a means of getting client software through existing users, so they can join the network without having to look for software online.
8. The process for creating a new user account should be as automated as possible; the new user shouldn't have to answer lots of questions in order to use the software.
9. Clients should be able to detect the presence of each other automatically, without any explicit information provided by the user (such as network addresses, usernames).
10. The client should expose a simple interface for the user to see other users on the network, and list their shared resources.
11. Clients should be able to trust the identity of other users of the system, without the user having to manually recognise each new user on the network.
12. The client should have a central place to configure all aspects of its operation.
13. Users should be able to give their client a recognisable name, so that other users can easily identify which user they are sharing with.

6.4.4 Security

14. A User should be able to vouch for the identity of new users to the system, and their recommendations should be available to all other users on the network.
15. Users should not be able to impersonate other users of the system.

16. Clients should have a guarantee of assurance that communications are coming from who they claim to be from – safe from spoofing or traffic replay attacks.
17. Users should be able to restrict which other users may access individual files or directories they are sharing with the system.
18. Users should have the option to send resources over a secure connection; certain files may contain sensitive information, but encrypting all file transfers uses excessive amounts of resources.
19. Meta information should be sent over a secure connection where possible (directory browsing, requests, etc). Otherwise clients denied by access restrictions may be able to passively glean information about the resources a user is sharing.

6.4.5 Non-functional Requirements

20. The client should be easy to install, and shouldn't ask the user complicated questions.
21. The client should be as intuitive to use as possible. This means following OS specific conventions, e.g. start menu and desktop icons on windows, finder interaction on Mac OS.

6.4.6 Optional Features

22. The client could have interoperability with existing file-sharing protocols in order to share files with users who don't have the client software installed (such as a means of sharing the installer for the client software itself).
23. The client could operate on the site-local ipv6 network subnet, to transcend physical network boundaries, such as wired and wireless network segments.

6.5 Architectural Overview

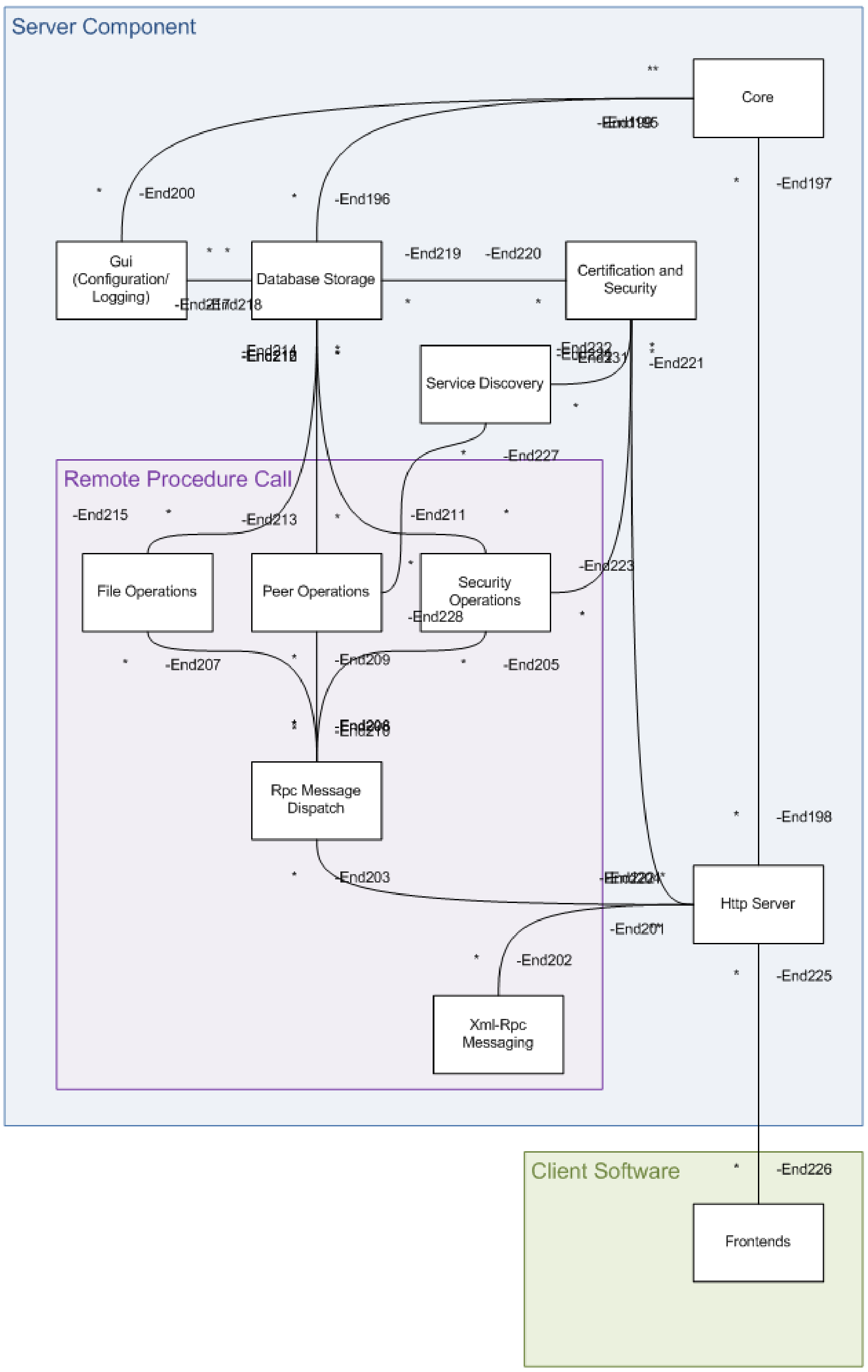


Figure 6 - Architectural Overview of the main components in the software design

6.6 Database Entities and Relationships

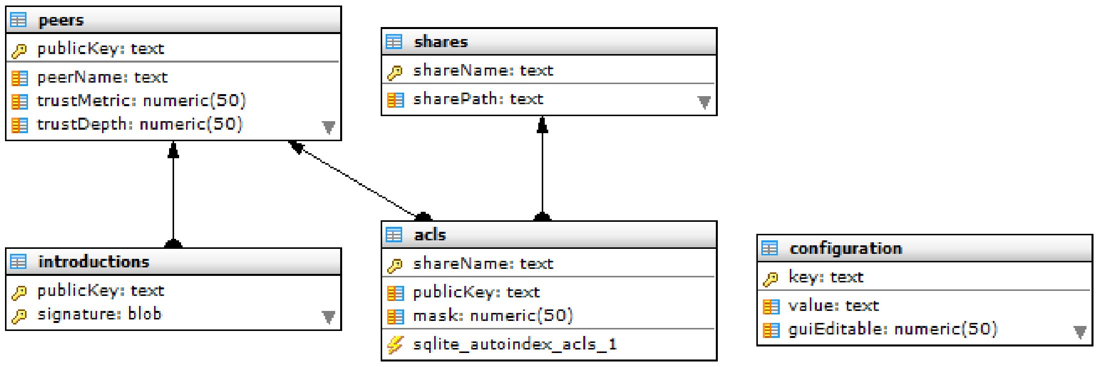


Figure 7 - Entity Relationship diagrams for internal SQLite database in the RSDSS Server

6.7 Network Address Selection

One of the challenges in this project has been to select the right IP addresses to advertise on. Multicast DNS support two different types of records, shared and unique[26]. Shared records may return multiple responses from different peers on the network, and will be used to list peers connected to the network. Unique records are owned by a single peer and will be used to advertise a peer's network address and port number. The unique records will prevent name collisions on the network, but also mean each peer will only be able to advertise a single network address for each protocol (IPv4 and IPv6).

The challenge then is selecting just one of all of the computer's network addresses to advertise. Choosing the wrong address could mean some peers would be able to see the existence of the server, but not be able to connect to it. This software takes the approach that a "public" (globally unambiguous) IP address is likely to be the most reachable, with RFC1918[27] private addresses less so, and RFC3927 [28]link-local the least accessible. For the private address ranges the assumption is made that the more hosts that could be on the network the more useful the address could be; therefore the larger the subnet the more useful the address might be.

Therefore the software will chose the highest weighted IPv4 address available on the machine, taking the weightings from the following list (in increasing order of precedence):

- Loopback addresses: 127.0.0.0/8
- RFC3927 Link-local addresses: 169.254.0.0/16
- RFC1918 Private Addresses:
    - 192.168.0.0/24
    - 172.16.0.0/12
    - 10.0.0.0/8
- Any other address, which should be globally unique.

For IPv6 there are three main scopes built into the addressing scheme: link local, site local or global. A similar strategy is used to determine the best address to use using these scopes as weightings.

[26] (Cheshire & Krochmal, 2006, p. 3)
[27] (Rekhter, Moskowitz, Karrenberg, de Groot, & Lear, 1996, p. 4)
[28] (Cheshire, Aboba, & Guttman, 2005)

6.8 SSL Host-Side Peer Verification

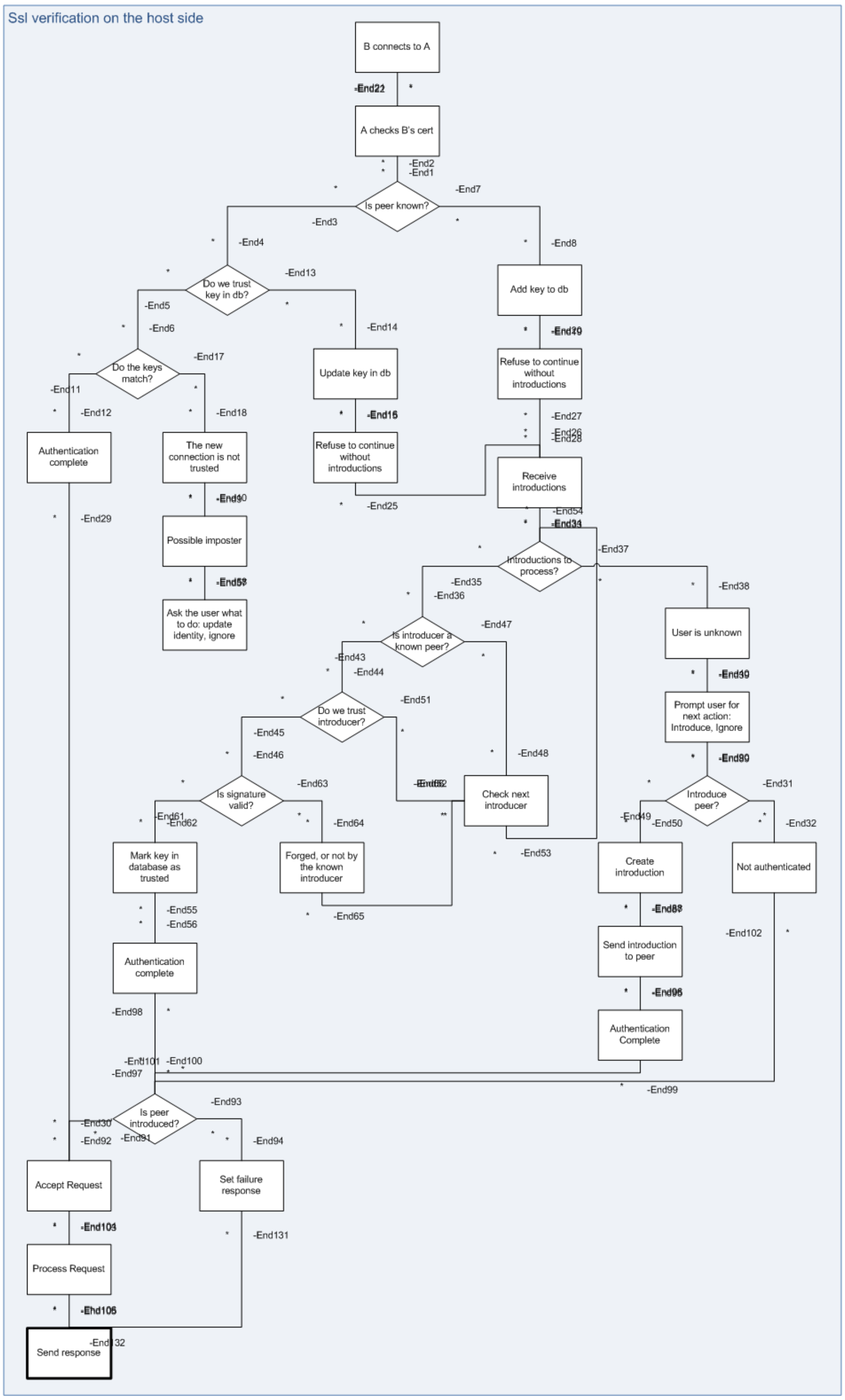


Figure 8 - Flow Diagram showing how a peer's identity will be verified on the host side of the connection

6.9 SSL Client-Side Peer Verification

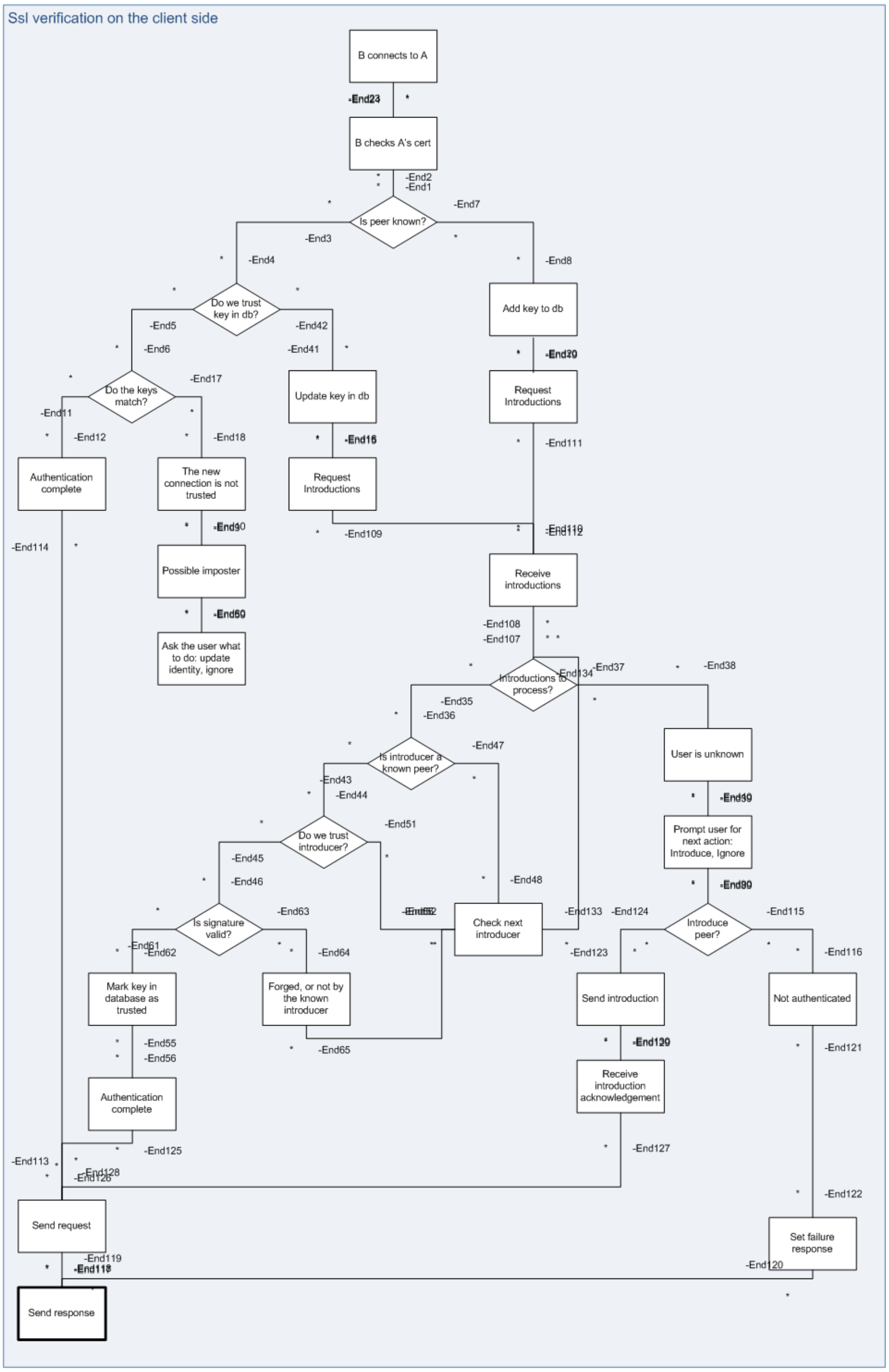


Figure 9 - Flow Diagram showing how a peer's identity will be verified on the client side of the connection

6.10 Download Manager Flow Diagram

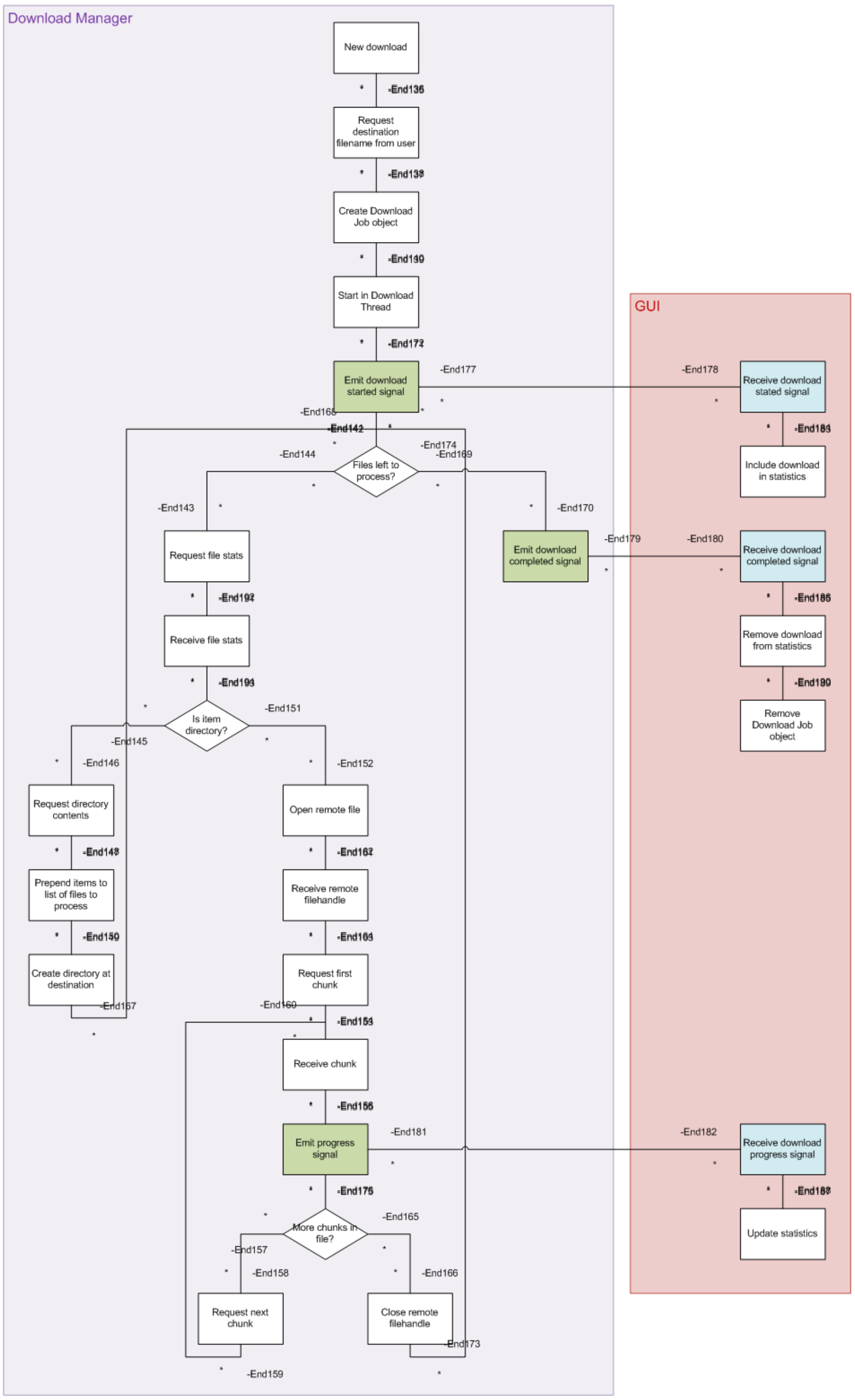

Download Manager
GUI
New download
Request destination filename from user
Create Download Job object
Start in Download Thread
Emit download started signal
Files left to process?
Request file stats
Receive file stats
Is item directory?
Request directory contents
Prepend items to list of files to process
Create directory at destination
Open remote file
Receive remote filehandle
Request first chunk
Receive chunk
Emit progress signal
More chunks in file?
Request next chunk
Close remote filehandle
Emit download completed signal
Receive download stated signal
Include download in statistics
Receive download completed signal
Remove download from statistics
Remove Download Job object
Receive download progress signal
Update statistics

Figure 10 - Flow Diagram showing how recursive file downloads will be processed, including GUI update signals.

6.11 Server Application Class Diagram

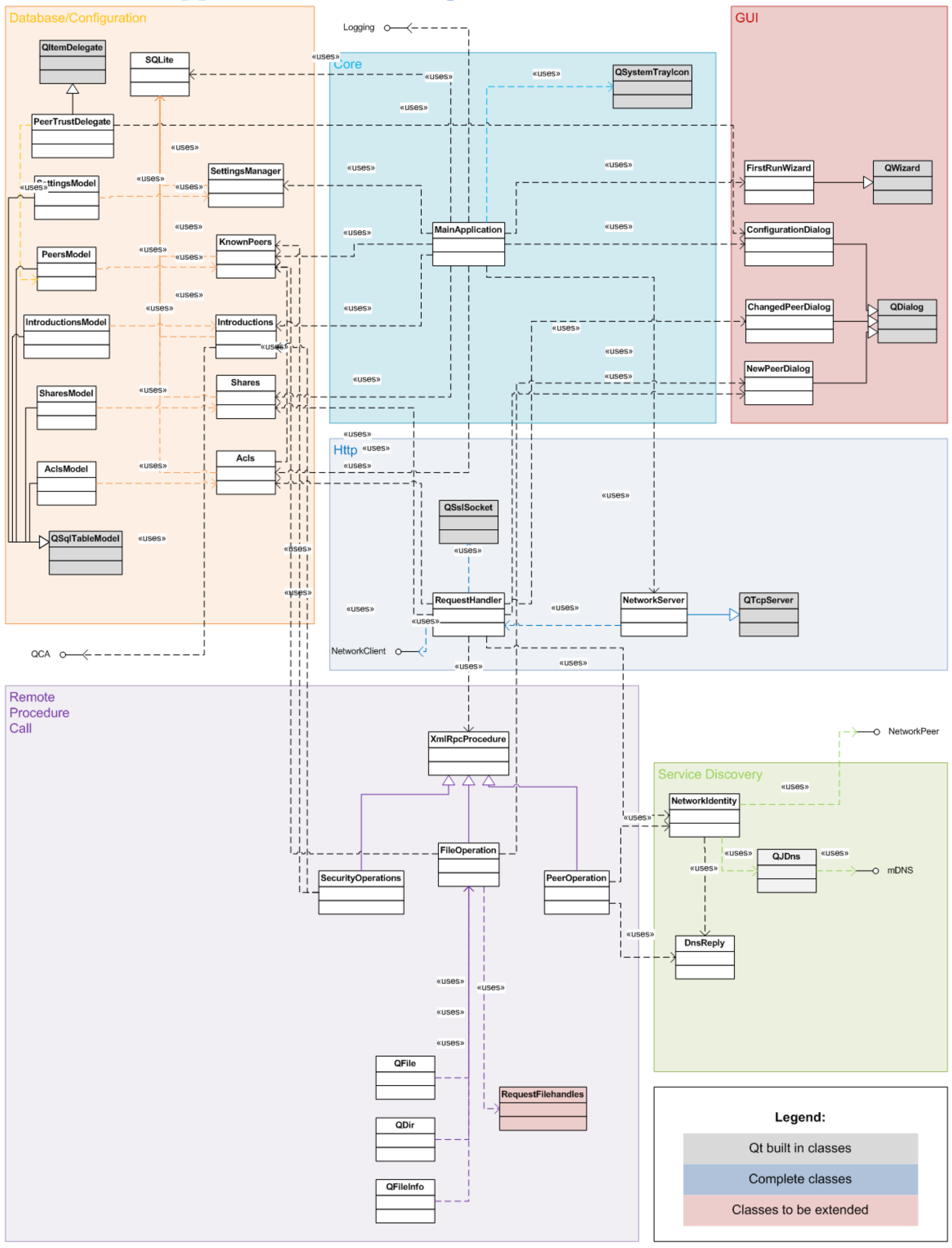


Figure 11 - Class Diagram for server application components

6.12 Shared Library Class Diagram

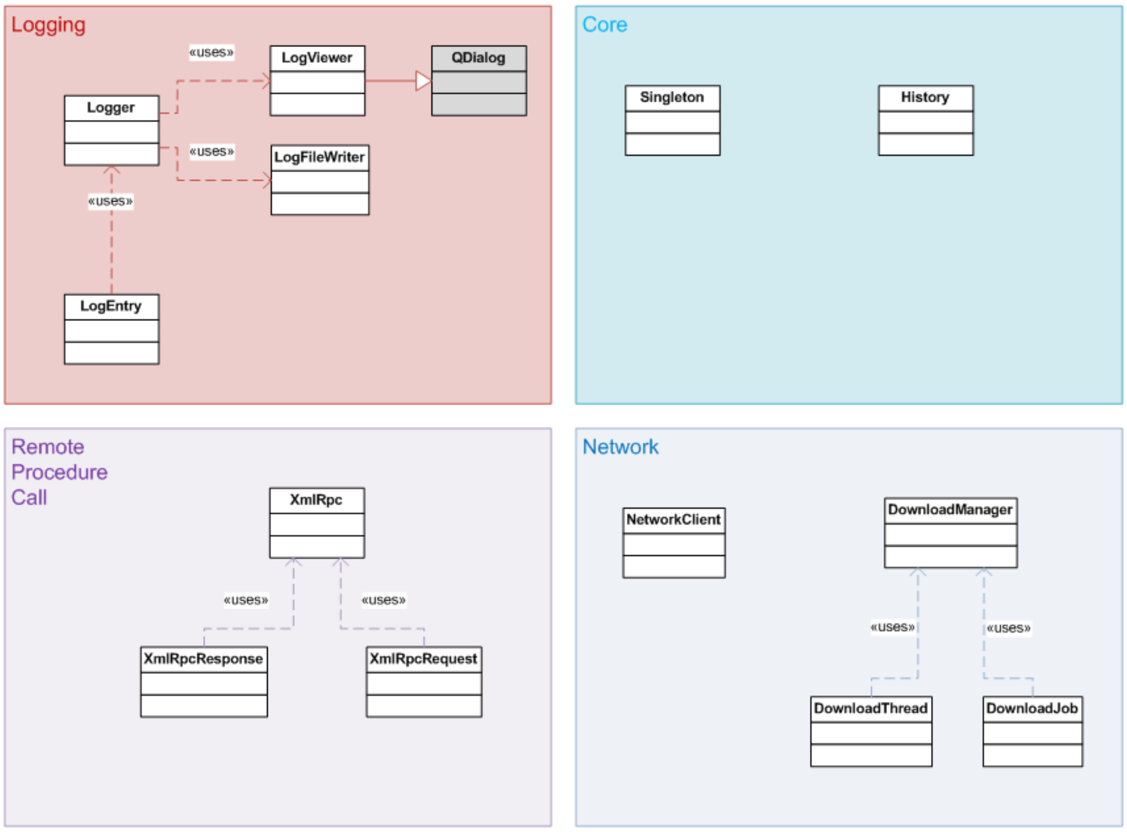
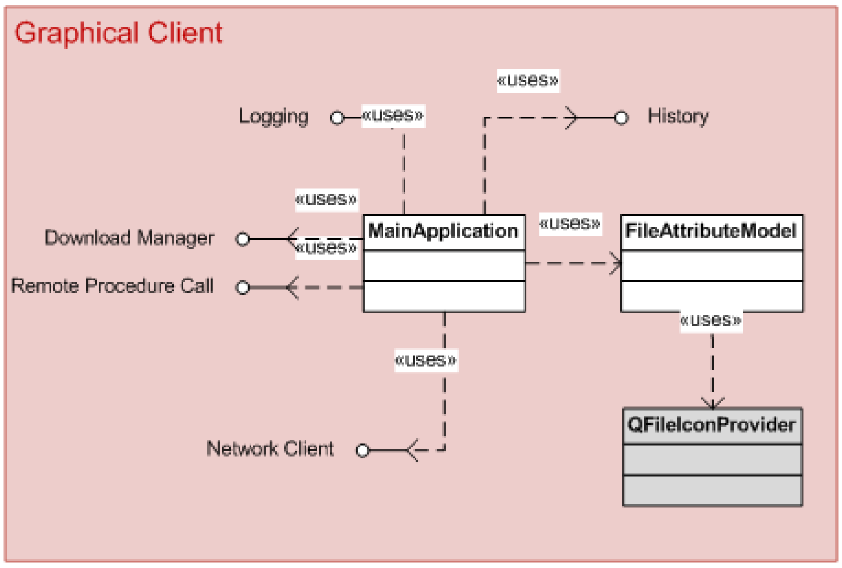


Figure 12 - Class Diagram for shared library components

6.13 Client Applications Class Diagrams



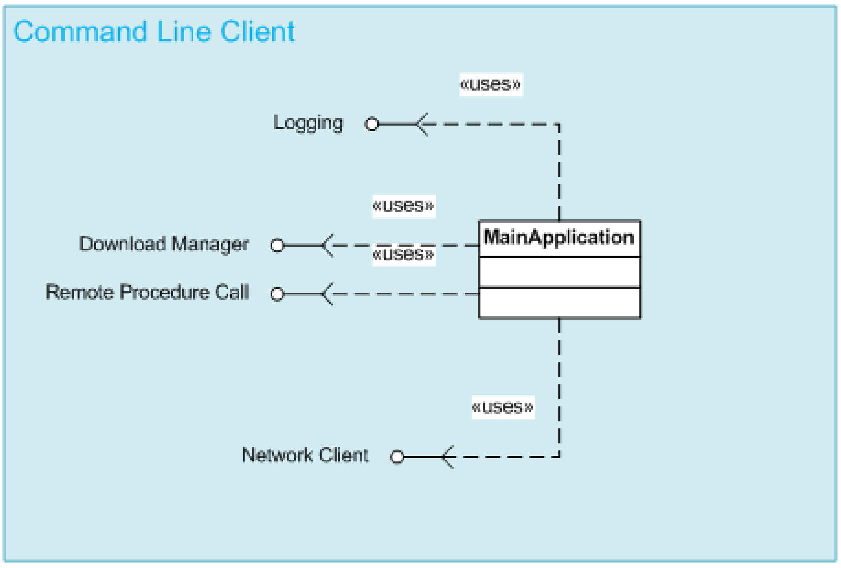


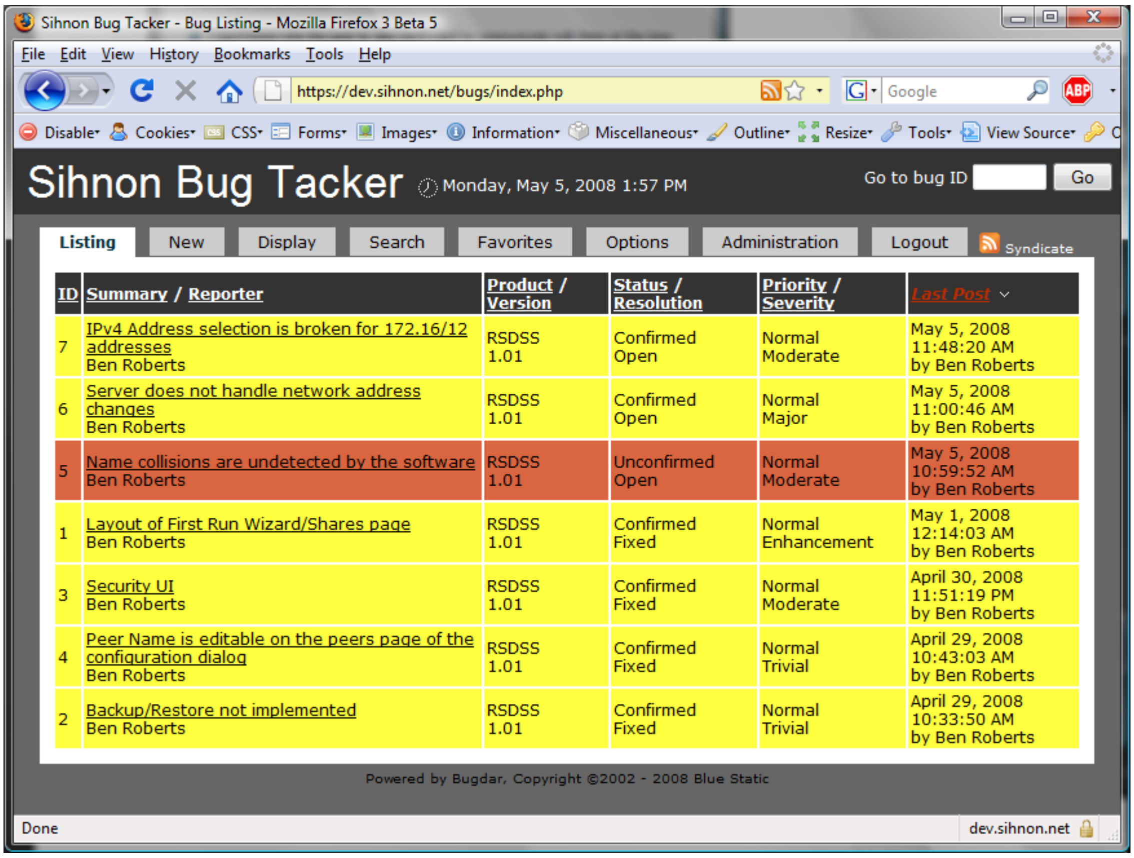
Figure 13 - Class Diagrams for graphical and command-line client applications

6.14 Unit Test Results

The following is an example of the output from the unit test application. An error has deliberately been left in the application in order to highlight a failure case.

```
bash.exe-2.03$ ./test_rsdssserver.exe
********* Start testing of TestAcls *********
Config: Using QTest library 4.4.1-snapshot-20080412, Qt 4.4.1-
snapshot-20080412
PASS   : TestAcls::initTestCase()
PASS   : TestAcls::testGetPermissionNames()
PASS   : TestAcls::testGetPermissionName()
PASS   : TestAcls::testGetPermissionValue()
PASS   : TestAcls::testHasPermissionsDefined()
PASS   : TestAcls::testHasRequiredPermissions()
PASS   : TestAcls::testAddPermission()
PASS   : TestAcls::testRemovePermission()
PASS   : TestAcls::testRemoveAllPermissions()
PASS   : TestAcls::testUpdatePermissions()
PASS   : TestAcls::cleanupTestCase()
Totals: 11 passed, 0 failed, 0 skipped
********* Finished testing of TestAcls *********
********* Start testing of TestShares *********
Config: Using QTest library 4.4.1-snapshot-20080412, Qt 4.4.1-
snapshot-20080412
PASS   : TestShares::initTestCase()
PASS   : TestShares::testCount()
PASS   : TestShares::testGetList()
PASS   : TestShares::testExists()
PASS   : TestShares::testLookupPath()
PASS   : TestShares::testLookupName()
PASS   : TestShares::testAddShare()
PASS   : TestShares::testRemoveShare()
PASS   : TestShares::testUpdateShare()
PASS   : TestShares::cleanupTestCase()
Totals: 10 passed, 0 failed, 0 skipped
********* Finished testing of TestShares *********
********* Start testing of TestPeers *********
Config: Using QTest library 4.4.1-snapshot-20080412, Qt 4.4.1-
snapshot-20080412
PASS   : TestPeers::initTestCase()
PASS   : TestPeers::testCount()
PASS   : TestPeers::testGetNames()
PASS   : TestPeers::testIsKnown()
PASS   : TestPeers::testGetPeerName()
PASS   : TestPeers::testGetTrust()
PASS   : TestPeers::testAddPeer()
PASS   : TestPeers::testRemovePeer()
PASS   : TestPeers::testUpdateName()
PASS   : TestPeers::testUpdateTrust()
PASS   : TestPeers::cleanupTestCase()
Totals: 11 passed, 0 failed, 0 skipped
********* Finished testing of TestPeers *********
********* Start testing of TestSettings *********
Config: Using QTest library 4.4.1-snapshot-20080412, Qt 4.4.1-
snapshot-20080412
PASS   : TestSettings::initTestCase()
PASS   : TestSettings::testGetConfig()
PASS   : TestSettings::testSetConfig()
PASS   : TestSettings::testCreateConfig()
PASS   : TestSettings::testDeleteConfig()
PASS   : TestSettings::cleanupTestCase()
Totals: 6 passed, 0 failed, 0 skipped
********* Finished testing of TestSettings *********
********* Start testing of TestIntroductions *********
Config: Using QTest library 4.4.1-snapshot-20080412, Qt 4.4.1-
snapshot-20080412
PASS   : TestIntroductions::initTestCase()
FAIL!  : TestIntroductions::testCount() Compared values are not
the same
   Actual (Introductions::count()): 2
   Expected (3): 3
.\test_introductions.cpp(97) : failure location
PASS   : TestIntroductions::testIsIntroducer()
PASS   : TestIntroductions::testGetIntroductions()
PASS   : TestIntroductions::testGetSignature()
PASS   : TestIntroductions::testAddIntroduction()
PASS   : TestIntroductions::testRemoveIntroduction()
PASS   : TestIntroductions::testCreateIntroduction()
PASS   : TestIntroductions::testIsIntroductionValid()
PASS   : TestIntroductions::cleanupTestCase()
Totals: 9 passed, 1 failed, 0 skipped
********* Finished testing of TestIntroductions *********
```

6.15 Bug Tracker


Sihnon Bug Tacker - Bug Listing - Mozilla Firefox 3 Beta 5
File Edit View History Bookmarks Tools Help
https://dev.sihnon.net/bugs/index.php
Google
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize Tools View Source
Sihnon Bug Tacker
Monday, May 5, 2008 1:57 PM
Go to bug ID
Go
Listing
New
Display
Search
Favorites
Options
Administration
Logout
Syndicate

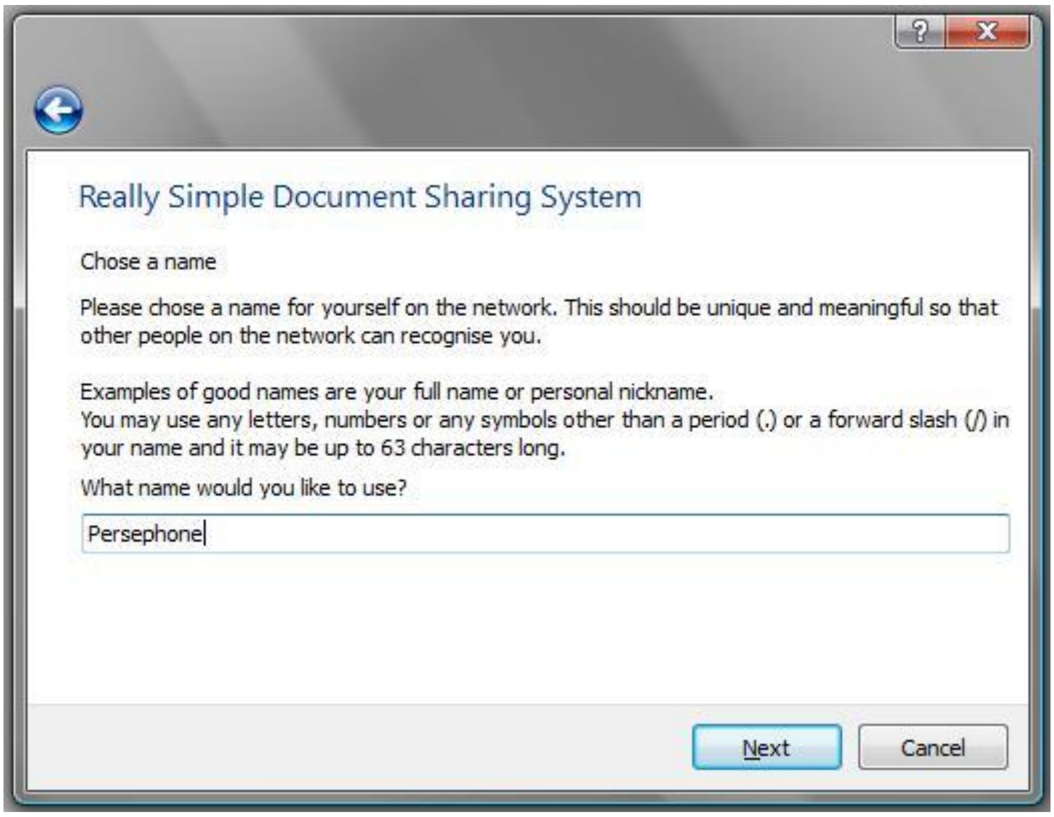
ID	Summary / Reporter	Product / Version	Status / Resolution	Priority / Severity	Last Post
7	IPv4 Address selection is broken for 172.16/12 addresses Ben Roberts	RSDSS 1.01	Confirmed Open	Normal Moderate	May 5, 2008 11:48:20 AM by Ben Roberts
6	Server does not handle network address changes Ben Roberts	RSDSS 1.01	Confirmed Open	Normal Major	May 5, 2008 11:00:46 AM by Ben Roberts
5	Name collisions are undetected by the software Ben Roberts	RSDSS 1.01	Unconfirmed Open	Normal Moderate	May 5, 2008 10:59:52 AM by Ben Roberts
1	Layout of First Run Wizard/Shares page Ben Roberts	RSDSS 1.01	Confirmed Fixed	Normal Enhancement	May 1, 2008 12:14:03 AM by Ben Roberts
3	Security UI Ben Roberts	RSDSS 1.01	Confirmed Fixed	Normal Moderate	April 30, 2008 11:51:19 PM by Ben Roberts
4	Peer Name is editable on the peers page of the configuration dialog Ben Roberts	RSDSS 1.01	Confirmed Fixed	Normal Trivial	April 29, 2008 10:43:03 AM by Ben Roberts
2	Backup/Restore not implemented Ben Roberts	RSDSS 1.01	Confirmed Fixed	Normal Trivial	April 29, 2008 10:33:50 AM by Ben Roberts

Powered by Bugdar, Copyright ©2002 - 2008 Blue Static

Figure 14 - Screenshot of the bug tracker used to record new bugs found in the software and the progress made in fixing bugs.

6.16 Screenshots

This appendix contains screenshots of various parts of the application running under different operating systems.



Really Simple Document Sharing System

Chose a name

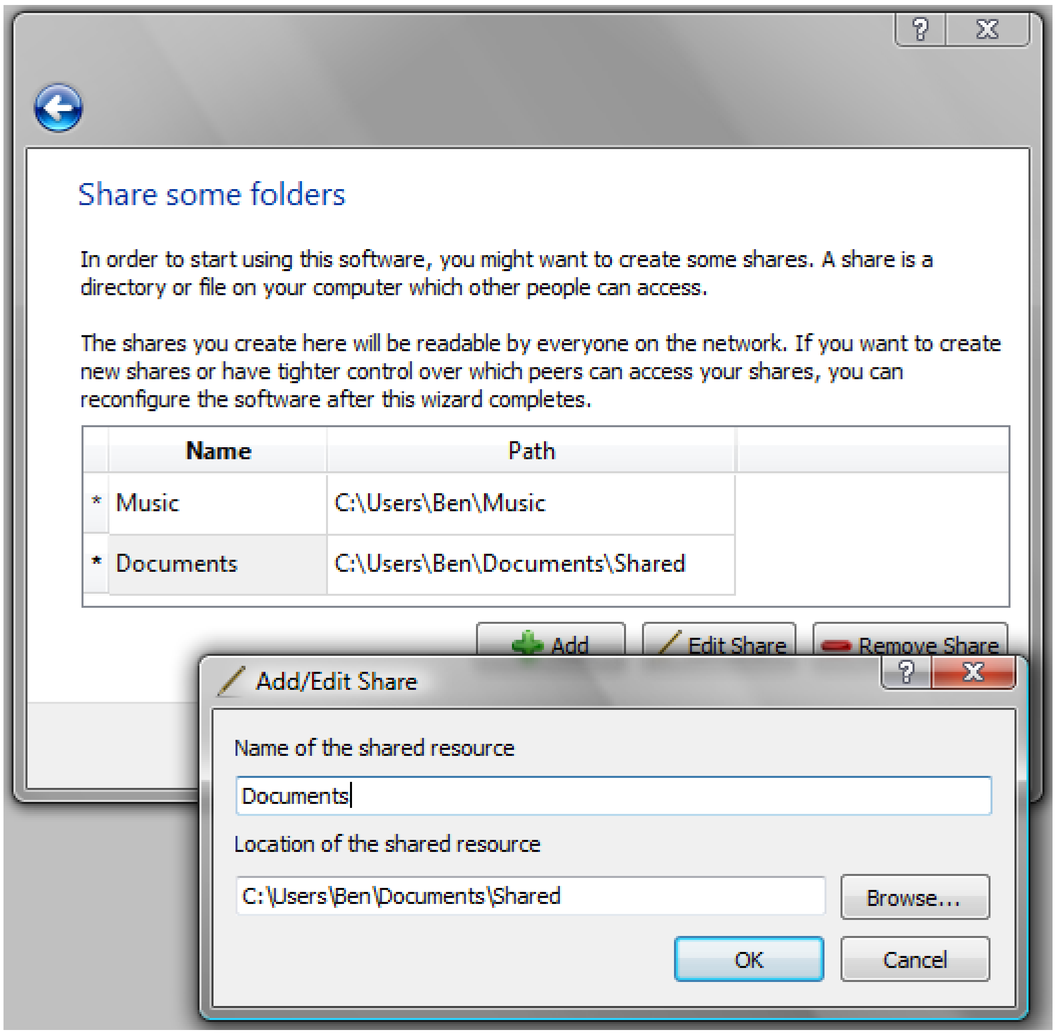
Please chose a name for yourself on the network. This should be unique and meaningful so that other people on the network can recognise you.

Examples of good names are your full name or personal nickname.
You may use any letters, numbers or any symbols other than a period (.) or a forward slash (/) in your name and it may be up to 63 characters long.

What name would you like to use?

Persephone

Figure 15 - First Run Wizard requesting user for a name, running on Windows Vista



Share some folders

In order to start using this software, you might want to create some shares. A share is a directory or file on your computer which other people can access.

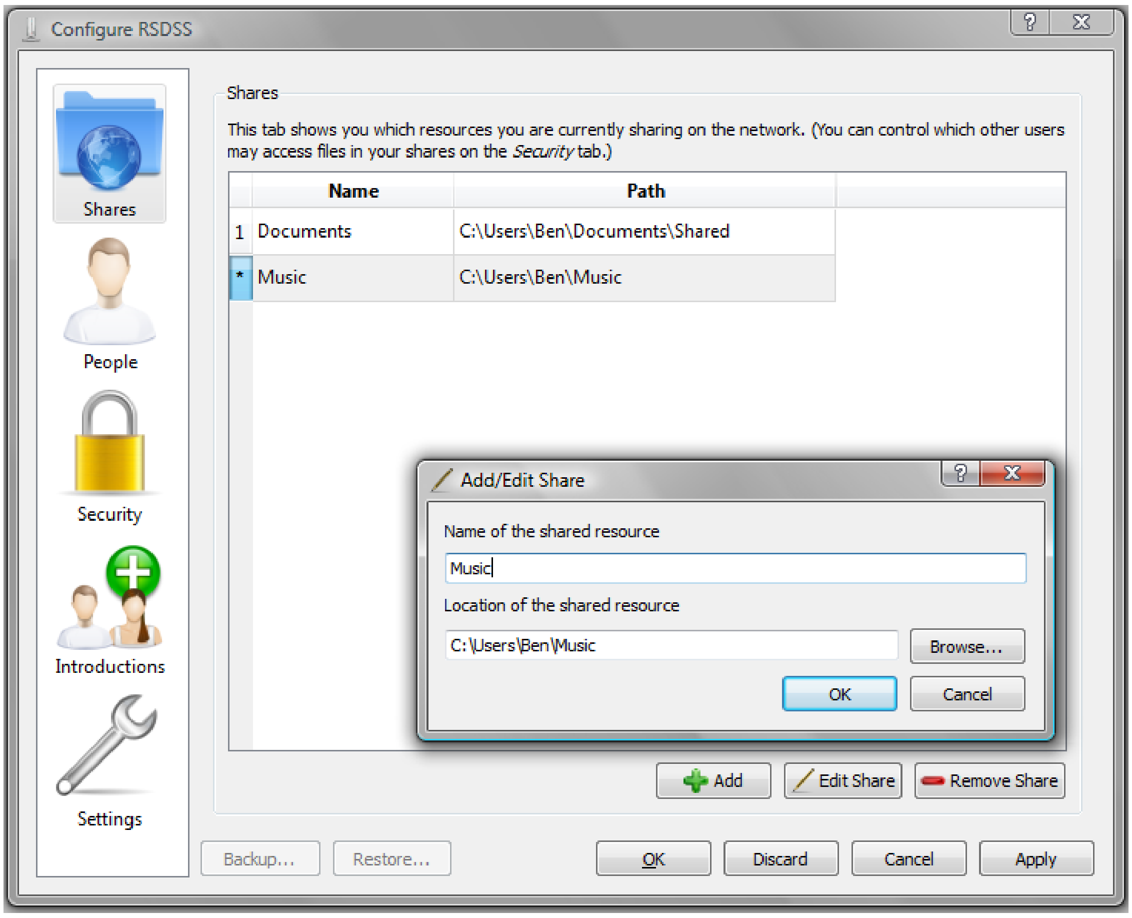
The shares you create here will be readable by everyone on the network. If you want to create new shares or have tighter control over which peers can access your shares, you can reconfigure the software after this wizard completes.

	Name	Path
*	Music	C:\Users\Ben\Music
*	Documents	C:\Users\Ben\Documents\Shared

Name of the shared resource: Documents

Location of the shared resource: C:\Users\Ben\Documents\Shared

Figure 16 - First Run Wizard showing the setup of shared resources, running on Windows Vista



Shares

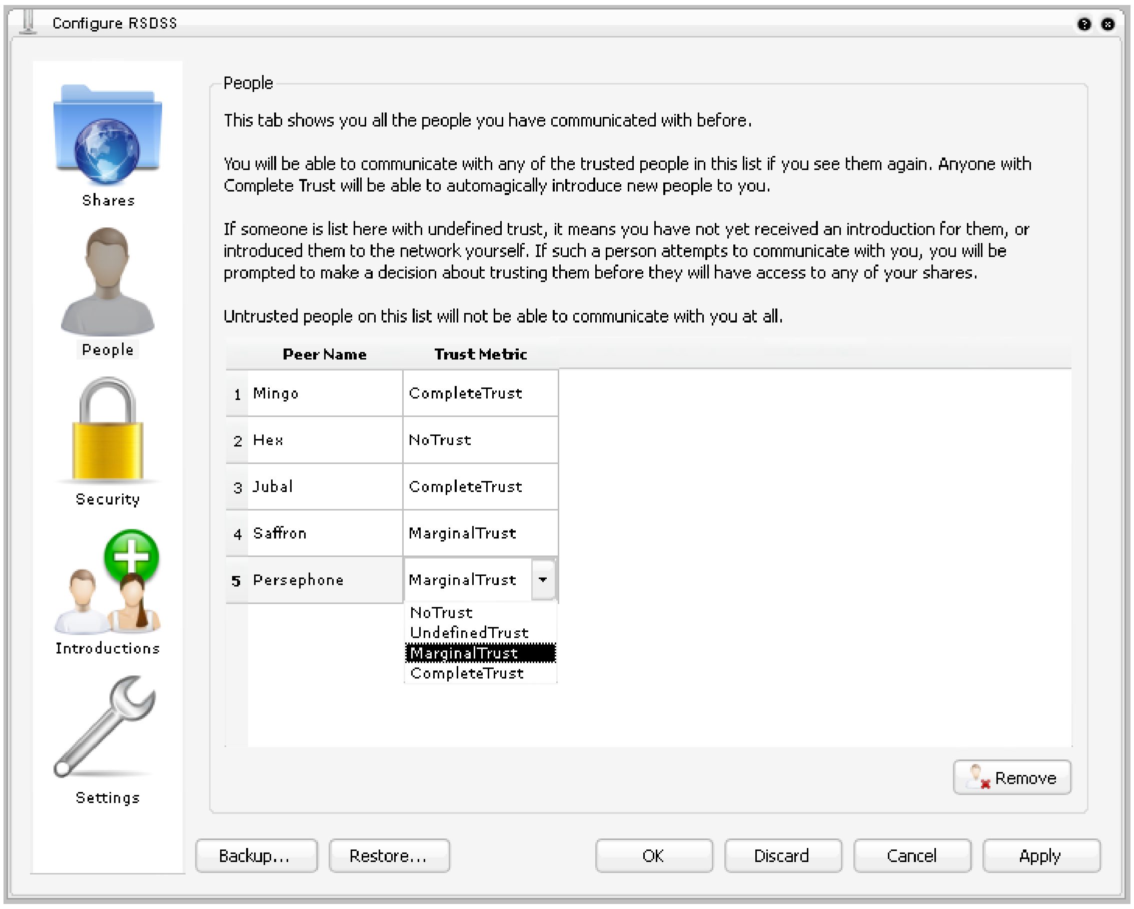
This tab shows you which resources you are currently sharing on the network. (You can control which other users may access files in your shares on the *Security* tab.)

	Name	Path
1	Documents	C:\Users\Ben\Documents\Shared
*	Music	C:\Users\Ben\Music

Name of the shared resource: Music

Location of the shared resource: C:\Users\Ben\Music

Figure 17 - Shared resources management in the server application. Running son Windows Vista



People

This tab shows you all the people you have communicated with before.

You will be able to communicate with any of the trusted people in this list if you see them again. Anyone with Complete Trust will be able to automagically introduce new people to you.

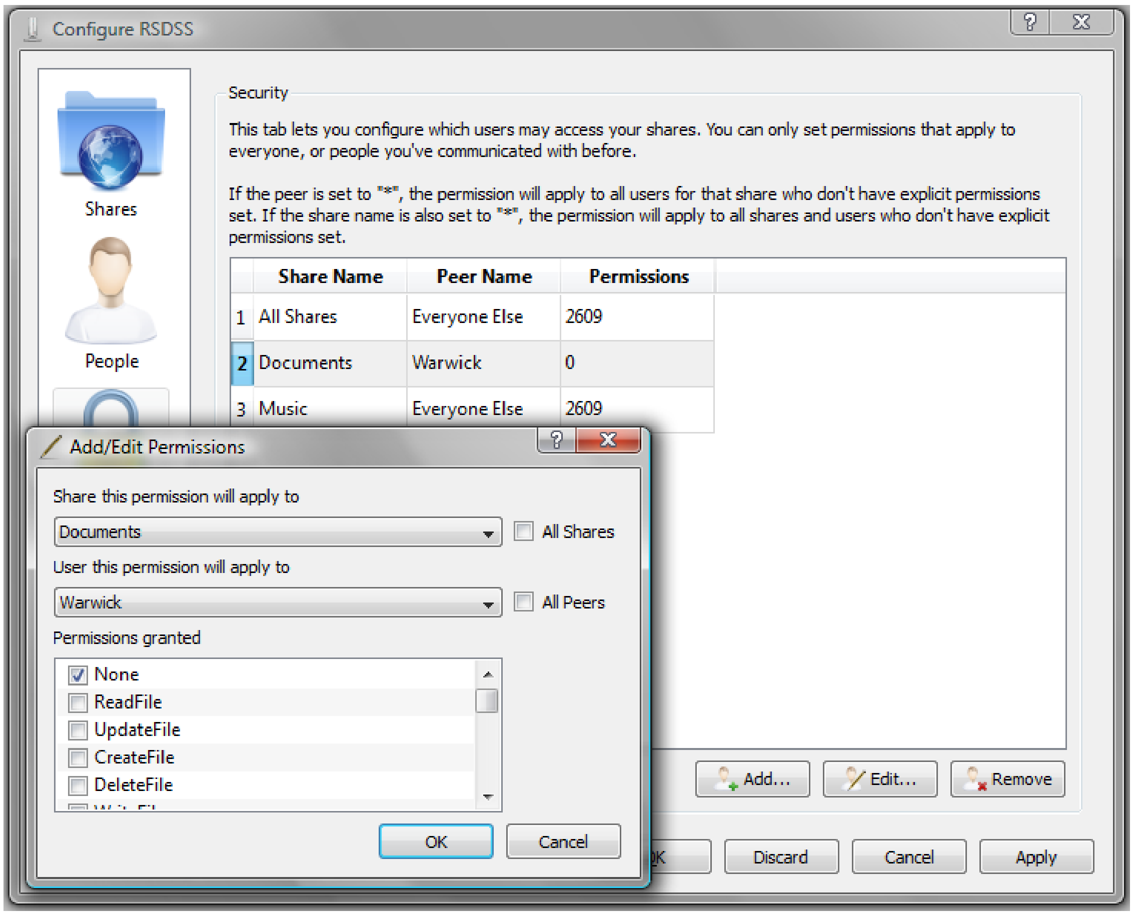
If someone is list here with undefined trust, it means you have not yet received an introduction for them, or introduced them to the network yourself. If such a person attempts to communicate with you, you will be prompted to make a decision about trusting them before they will have access to any of your shares.

Untrusted people on this list will not be able to communicate with you at all.

	Peer Name	Trust Metric
1	Mingo	CompleteTrust
2	Hex	NoTrust
3	Jubal	CompleteTrust
4	Saffron	MarginalTrust
5	Persephone	MarginalTrust

NoTrust
UndefinedTrust
MarginalTrust
CompleteTrust

Figure 18 - Peer trust management in the Server Application. Running on Windows XP



This tab lets you configure which users may access your shares. You can only set permissions that apply to everyone, or people you've communicated with before.

If the peer is set to "*", the permission will apply to all users for that share who don't have explicit permissions set. If the share name is also set to "*", the permission will apply to all shares and users who don't have explicit permissions set.

	Share Name	Peer Name	Permissions
1	All Shares	Everyone Else	2609
2	Documents	Warwick	0
3	Music	Everyone Else	2609

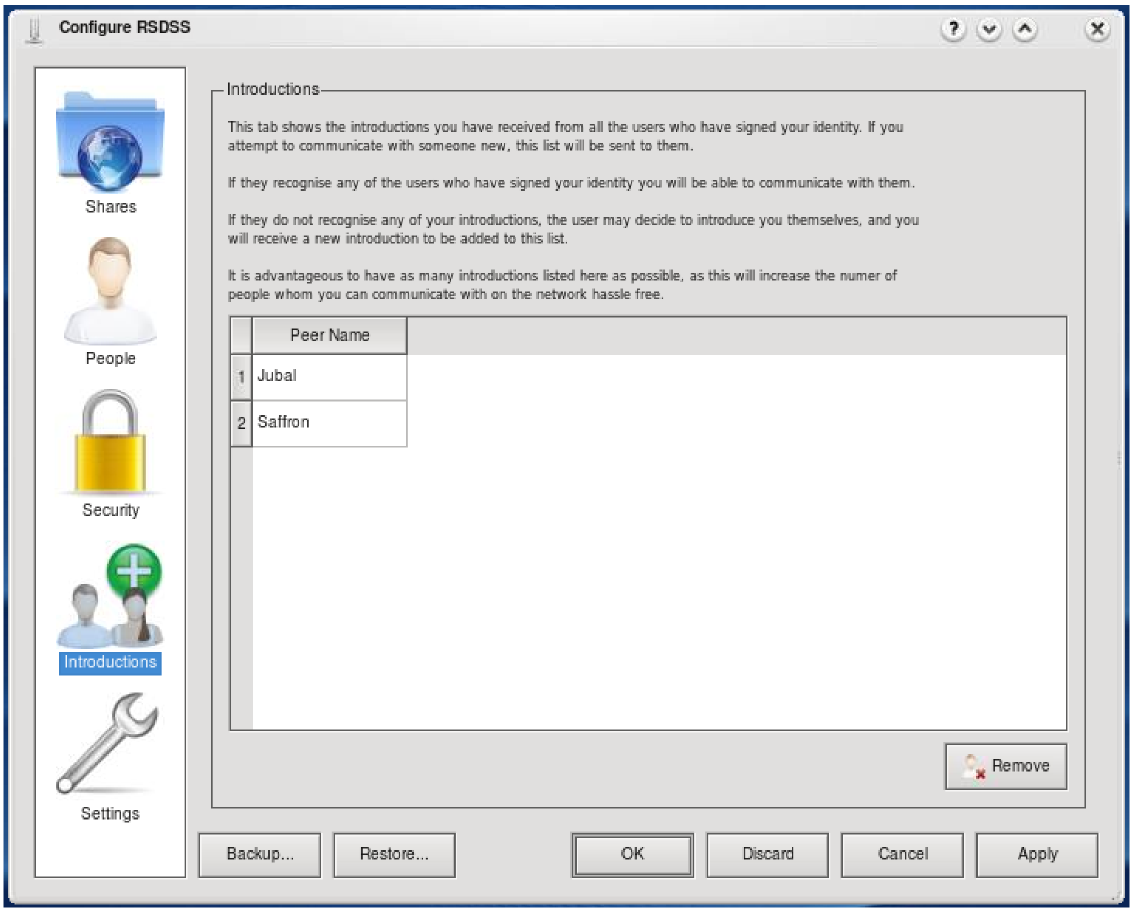
Share this permission will apply to: Documents — All Shares [ ]

User this permission will apply to: Warwick — All Peers [ ]

Permissions granted

- [✓] None
- [ ] ReadFile
- [ ] UpdateFile
- [ ] CreateFile
- [ ] DeleteFile

Figure 19 - Access control management in the server application. Running on Windows Vista.



Introductions

This tab shows the introductions you have received from all the users who have signed your identity. If you attempt to communicate with someone new, this list will be sent to them.

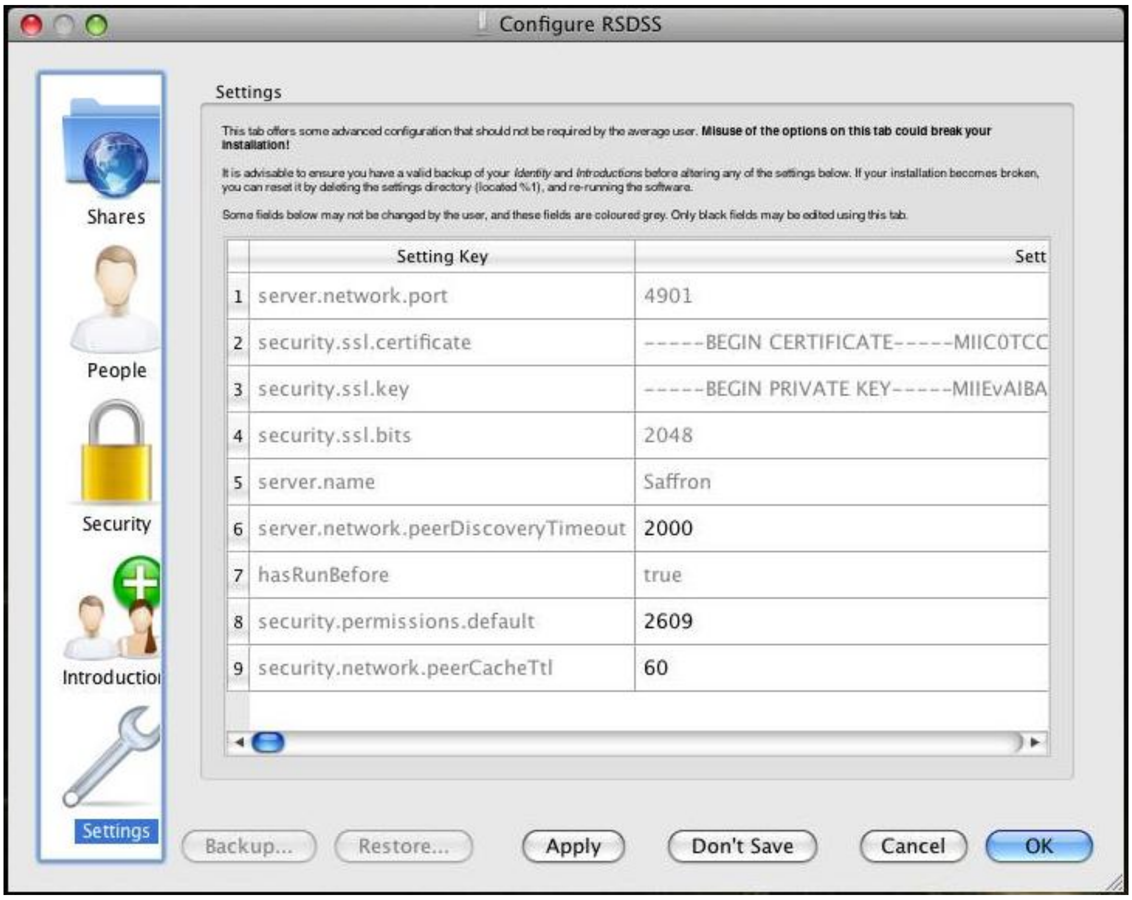
If they recognise any of the users who have signed your identity you will be able to communicate with them.

If they do not recognise any of your introductions, the user may decide to introduce you themselves, and you will receive a new introduction to be added to this list.

It is advantageous to have as many introductions listed here as possible, as this will increase the numer of people whom you can communicate with on the network hassle free.

	Peer Name
1	Jubal
2	Saffron

Figure 20 - Introductions from other peers in the server application. Running on Linux/KDE.



Settings

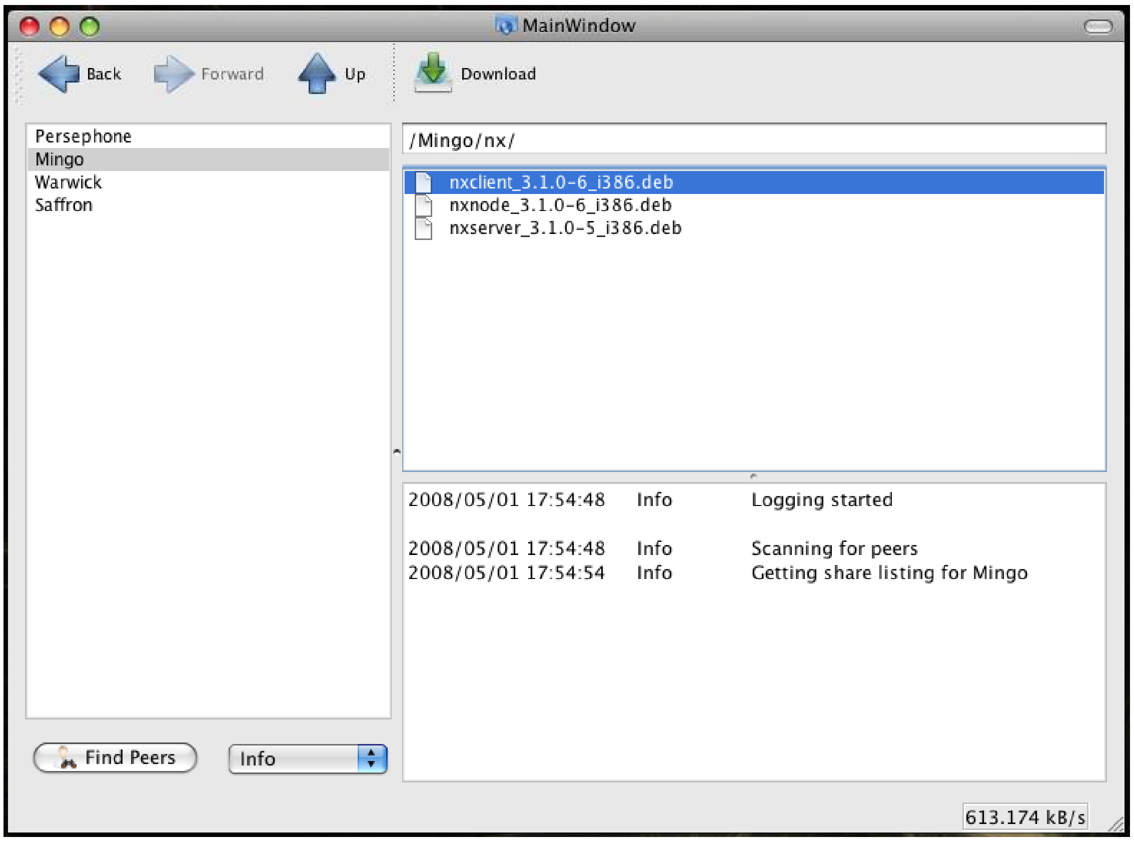
This tab offers some advanced configuration that should not be required by the average user. **Misuse of the options on this tab could break your installation!**

It is advisable to ensure you have a valid backup of your *Identity* and *Introductions* before altering any of the settings below. If your installation becomes broken, you can reset it by deleting the settings directory (located %1), and re-running the software.

Some fields below may not be changed by the user, and these fields are coloured grey. Only black fields may be edited using this tab.

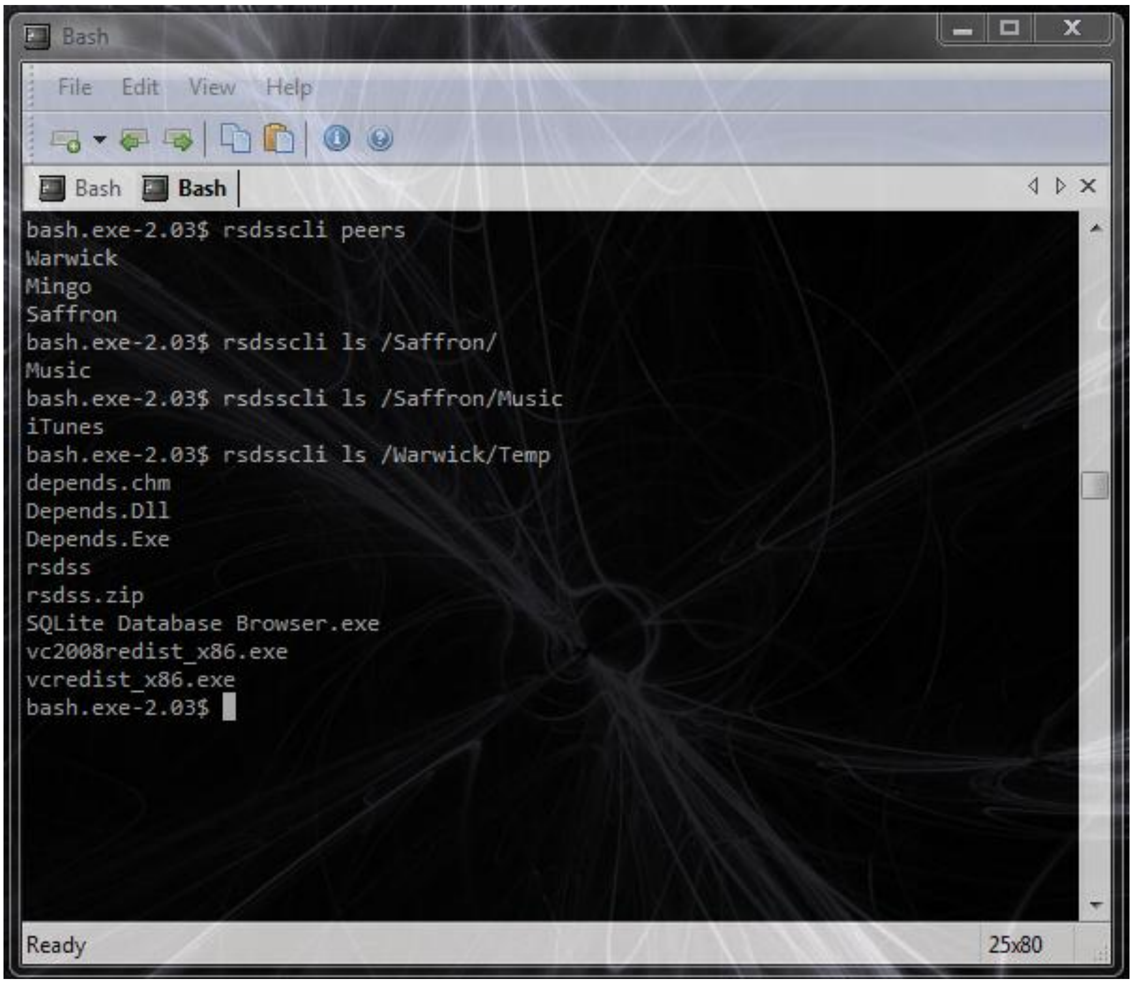
	Setting Key	Sett
1	server.network.port	4901
2	security.ssl.certificate	-----BEGIN CERTIFICATE-----MIIC0TCC
3	security.ssl.key	-----BEGIN PRIVATE KEY-----MIIEvAIBA
4	security.ssl.bits	2048
5	server.name	Saffron
6	server.network.peerDiscoveryTimeout	2000
7	hasRunBefore	true
8	security.permissions.default	2609
9	security.network.peerCacheTtl	60

Figure 21 - Settings management in the server application. running on Mac OS X.



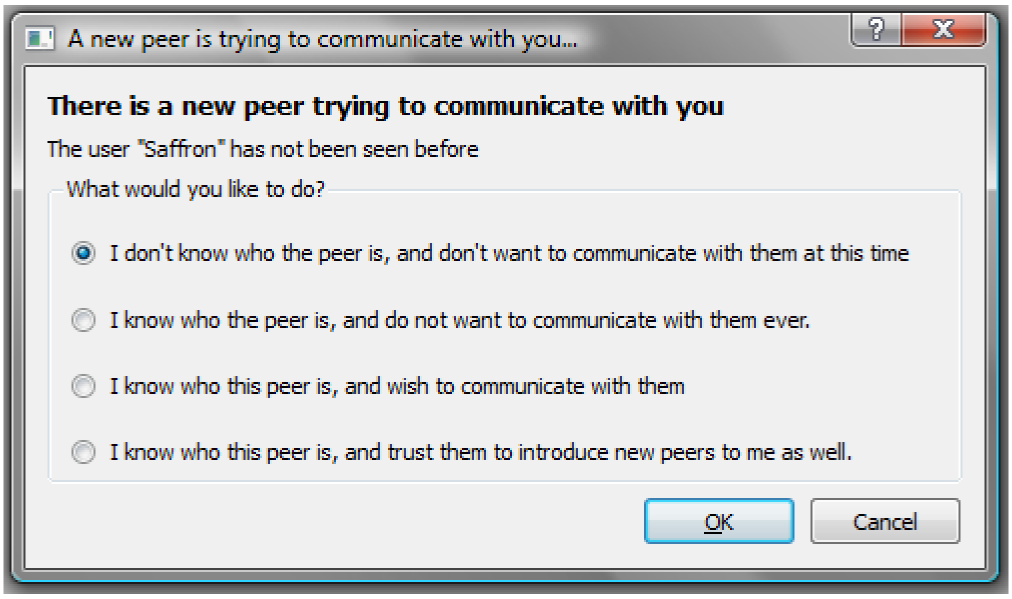
```
2008/05/01 17:54:48    Info    Logging started

2008/05/01 17:54:48    Info    Scanning for peers
2008/05/01 17:54:54    Info    Getting share listing for Mingo
```

Figure 22 - Graphical Client running on Mac OS X, showing peers, directory browsing, and file download.



```
bash.exe-2.03$ rsdsscli peers
Warwick
Mingo
Saffron
bash.exe-2.03$ rsdsscli ls /Saffron/
Music
bash.exe-2.03$ rsdsscli ls /Saffron/Music
iTunes
bash.exe-2.03$ rsdsscli ls /Warwick/Temp
depends.chm
Depends.Dll
Depends.Exe
rsdss
rsdss.zip
SQLite Database Browser.exe
vc2008redist_x86.exe
vcredist_x86.exe
bash.exe-2.03$
```

Figure 23 - Command line client, running on Windows Vista, showing network browsing.



A new peer is trying to communicate with you...

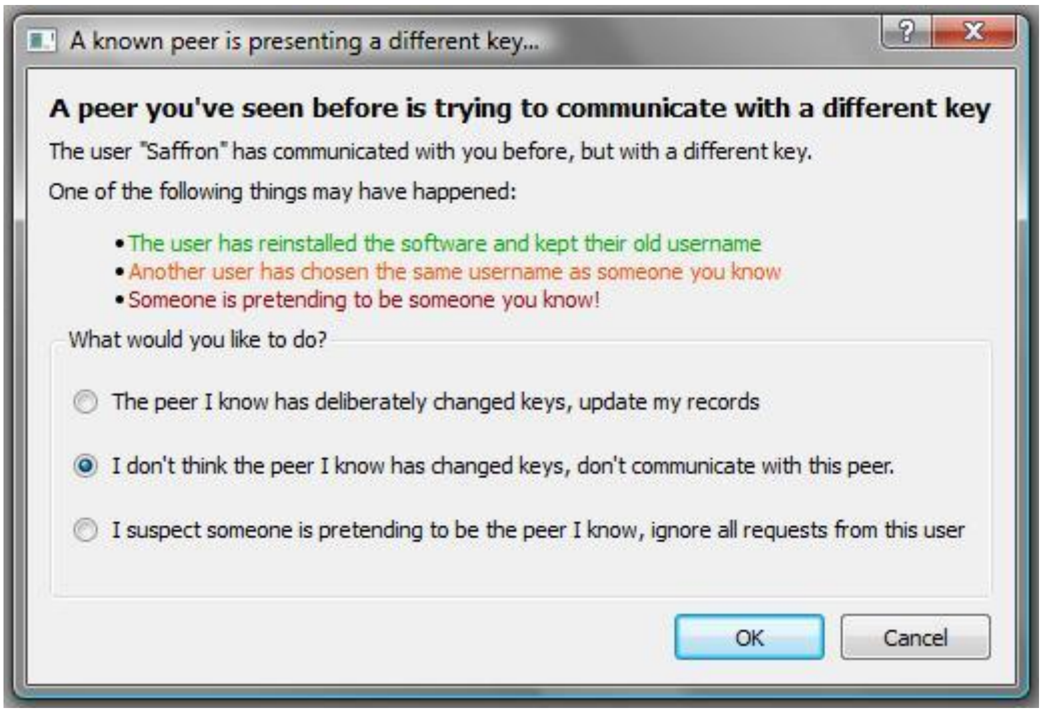
There is a new peer trying to communicate with you

The user "Saffron" has not been seen before

What would you like to do?

- [✓ I don't know who the peer is, and don't want to communicate with them at this time]
- I know who the peer is, and do not want to communicate with them ever.
- I know who this peer is, and wish to communicate with them
- I know who this peer is, and trust them to introduce new peers to me as well.

Figure 24 - Popup dialog shown when a new peer joins the network and attempts to communicate with another peer.



A known peer is presenting a different key...

A peer you've seen before is trying to communicate with a different key

The user "Saffron" has communicated with you before, but with a different key.

One of the following things may have happened:

- The user has reinstalled the software and kept their old username
- Another user has chosen the same username as someone you know
- Someone is pretending to be someone you know!

What would you like to do?

- The peer I know has deliberately changed keys, update my records
- [✓ I don't think the peer I know has changed keys, don't communicate with this peer.]
- I suspect someone is pretending to be the peer I know, ignore all requests from this user

Figure 25 - Popup dialog shown when a peer appears to have changed their key pair.

6.17 Source Code and Precompiled Packages

Attached to this project report should be a CD containing the full source code for all the software developed as a part of this project along with precompiled packages for the major operating systems supported.

A table of contents is included on the CD in a file called index.html.