Darknet file sharing

application of a private peer-to-peer
distributed file system concept

Johan Andersson & Gabriel Ledung
Abstract

Peer-to-peer network applications has been a tremendous success among end users and has therefore received much attention in academia and industry, as have illegal public file sharing in media. However, private peer-to-peer file sharing between family, friends and co-workers have attracted little interest from the research community. Existing approaches also limit the users by not allowing for native interaction with userspace applications. In this paper we explore how private file sharing can be made safe, fast and scalable without constraining the users in this aspect. We demonstrate the concept of a private file sharing application utilizing a decentralized peer-to-peer network overlay by creating a prototype with extreme programming as methodology. To maximize the freedom of users the network is accessed through a virtual file-system interface. The prototype proves this to be a valid approach and we hope readers can use this paper as a platform for further developments in this area.

Abstract in Swedish

Fildelningsapplikationer som använder peer-to-peer teknik har varit en enorm framgång blandslutanvändare och har därmed erhållit mycket uppmärksamhet från akademi och industri, liksom olaglig fildelning fått inom media. Däremot har inte privat fildelning mellan vänner, arbetskamrater och kolleger tilldelats samma uppmärksamhet från forskningssamfundet. Nuvarande tillämpningar begränsar användaren genom att inte tillåta naturlig interaktion med användarapplikationer. I denna uppsats utforskar vi hur privat fildelning kan göras snabb, skalbar och säker utan att begränsa användaren ur den aspekten. Vi demonstrerar ett koncept för privat fildelning som nyttjar decentraliserad peer-to-peer arkitektur m.h.a en prototyp som tagits fram med extreme programming som metodologi. För att maximera användarnas frihet nyttjas ett virtuellt filsystem som gränssnitt. Prototypen visar att vår tillämpning fungerar i praktiken och vi hoppas att läsaren kan använda vårt arbete som en plattform för fortsatt utveckling inom detta område.

Acknowledgments

We would like to thank our families and friends for their patience and support during these months and especially our supervisor Fredrik Bengtsson for the invaluable input and constructive critique given.

Keywords

file sharing, darknet, peer-to-peer (P2P), distributed file system, virtual file system, anonymity, BitTorrent, distributed hash tables (DHT)
## Table of contents

1. **Introduction** ................................................................. 1
   1.1 Background .................................................................. 1
   1.2 Purpose ....................................................................... 4
   1.3 Method ........................................................................ 4
   1.4 Delimitations .................................................................. 5
   1.5 Thesis outline ............................................................... 5

2. **Definitions and dependencies** ........................................ 6
   2.1 Network protocols .......................................................... 6
   2.2 Darknet and private peer-to-peer ...................................... 7
      2.2.1 Anonymity ............................................................... 9
      2.2.2 Trust .................................................................... 9
   2.3 BitTorrent ..................................................................... 10
      2.3.1 Functionality ......................................................... 10
   2.4 File systems ................................................................. 12
      2.4.1 Files ................................................................. 12
      2.4.2 Directories ............................................................ 14
      2.4.3 File-system mounting .............................................. 14
      2.4.4 Protection ............................................................. 14
      2.4.5 Concrete and virtual file systems ................................ 15
   2.5 Distributed file systems .................................................. 15
      2.5.1 Performance measurement .......................................... 15
   2.6 Survey results .............................................................. 16
      2.6.1 Infinit ................................................................. 17
      2.6.2 Colony FS ............................................................ 17
      2.6.3 Dust FS ............................................................... 17
      2.6.4 Dropbox .............................................................. 17
      2.6.5 OneSwarm ............................................................ 18
      2.6.6 Freenet ................................................................. 18

3. **Concept** ....................................................................... 19
   3.1 BitTorrent and file systems ............................................. 20
   3.2 User stories ................................................................. 21
   3.3 File Register ............................................................... 21
      3.3.1 File operations ....................................................... 23
   3.4 Communication protocol ............................................... 23
      3.4.1 Messages ............................................................. 24
   3.5 Invite users ................................................................. 24
   3.6 Initiate download .......................................................... 24

4. **Development** .............................................................. 25
   4.1 Delimitations for the prototype ........................................ 25
   4.2 Language and platform .................................................. 25
   4.3 Methodology ............................................................... 25
      4.3.1 Extreme programming .............................................. 25
      4.3.2 Prototyping .......................................................... 26
4.4 Development libraries.........................................................................................................................27
4.4.1 MonoTorrent.................................................................................................................................27
4.4.2 Dokan..................................................................................................................................................27
4.5 Class diagram........................................................................................................................................28
4.6 Results of the implementation.............................................................................................................29
4.6.1 Sub-Zero Network Protocol............................................................................................................31
4.6.2 Local cache.........................................................................................................................................31
4.6.3 Tree implementation............................................................................................................................31
4.6.4 User invitation.......................................................................................................................................31
4.6.5 Download initiation...............................................................................................................................32
4.7 Testing....................................................................................................................................................32

5 Discussion and conclusion.........................................................................................................................34
5.1 The field of file sharing.........................................................................................................................34
5.2 Creation of the application......................................................................................................................34
5.3 The application..........................................................................................................................................35
5.4 Further work............................................................................................................................................36
5.5 Conclusion...............................................................................................................................................37

Appendix 1 - File systems from Wikipedia
1 Introduction

Since the dawn of modern computing the act of sharing information digitally might be said to have been a central theme. One example is Usenet, a news and discussion system, that was conceived in 1979 and soon became immensely popular in academic and hacker circles (Lueg & Fisher, 2003). Usenet may also be where the practice of not just sharing texts erupted on a large scale due to the possibility of attaching encoded binary files to posts.

However, due to their inherent nature, public networks is obviously not a good solution to share sensitive data. It could be anything from party pictures to videos of freedom fighters in Iran risking their lives to spread important information of oppression. The point is that due to the combative legal actions taken by the copyright industry this first decade of the millennium, general file sharing has been almost completely reduced to denote something illegal, although it is no more illegal than sending mail by the postal service.

Sounds simple enough but many courts (Kravets, 2008) around the world today are deep into year long processes, with thousands of people prosecuted, trying to make sense of this gray water. One thing is clear to us though, file sharing applications are tools and it is the actions taken by people using the tools that should be under scrutiny, not the tools themselves. Another approach is needed altogether, file sharing does not have to be this way.

We can only agree with Thomas Koll, developer of ShareDirect\(^1\) (Andrews, 2004),

> “If you want to swap files with family, friends and co-workers, you ought to be able to do it in a way that doesn't make you feel like you're trolling the Web's Tenderloin district.”

We were inspired by the idea of sneaker nets, namely a group of people that meet by foot (wearing sneakers) exchanging usb-sticks that contain sensitive information and the implication that no one from the outside would be aware of the exchange.

And so we simply asked ourselves; instead of walking around with a usb-stick visiting our friends or accessing public file sharing networks, what if we could skip the walking part by having all of our friends share a "virtual usb-stick"?

1.1 Background

Copying information is essentially what computers do and since that didn't change when we started connecting computers into widespread networks it's not hard to imagine that today there is hundreds of file sharing applications for different uses. A simple look in the filesharing section of Download.com is sufficient to understand the large extent of the need in file sharing.

Traditionally the hierarchical concept of clients getting data from dedicated servers has been standing strong. This is how the world wide web works and many other services on the Internet. But running a server can be very expensive since the resources needed to serve clients is the sole responsibility of the server owner.

---

1 ShareDirect, http://www.laplink.com/sharedirect/
Thus it was usually corporations and larger organizations that together formed the world wide web and internet. However, with the new millennium also came a dramatic upswing for a concept just as old as Usenet; peer-to-peer.

Whereas the client/server concept is a one-to-many relationship, peer-to-peer is instead many-to-many and all nodes of the network are equal because each network host is a client and a server simultaneously. That was how the Usenet servers exchanged news articles between each other while the regular user connected to a server in a client fashion. Schollmeier (2001) tries to capture this duality of server and client in one node by combining the words into one; servlent. In this paper though we will instead use the more common term peer.

The power of peer-to-peer networks lies in them growing with the growing demand of the users, because every user is contributing resources (Li, 2008), such as bandwidth, storage and CPU cycles. Since there is no extra cost of adding peers it is easily scalable. Applications utilizing this approach has figuratively exploded in recent years, according to a report by Schulze and Mochalski (2008) over half of the worlds internet traffic, on average, is relayed in peer-to-peer networks.

Another strength of the peer-to-peer model is that it allows for decentralized networks, what we call pure, this is more secure since centralized network topologies introduces a certain degree of threat to the network and its users (Biddle et al, 2002). Being a single point of failure, the most obvious threat would be the possibility of a malicious agent trying to infiltrate the service or sabotage it using a denial of service attack (Huang & Gouda, 2007), but also important to consider is the control the owner has over such a service. DropBox is one of many popular services for personal file sharing between the users own devices, stationary as well as mobile. It employs a seemingly rigorous safety arrangement, but in the end you still have to trust them not to poke around in your personal data.

---

1 DropBox, http://www.dropbox.com/
We call a peer-to-peer network depending on some kind of centralized service *hybrid.* For example, DirectConnect\(^1\), as implied by its name, connects nodes directly through a peer-to-peer network, however since the network is defined by a central hub it may be called hybrid peer-to-peer rather than pure peer-to-peer.

Another way of differentiating between different peer-to-peer aspects is whether the network allows for direct connections between any pair of users, in which case we call it *group-based,* see Figure 1, or whether it only allows direct connections between users who know one another, that is referred to as *friend-to-friend,* see Figure 2.

![Friend-2-Friend network. Every peer has access to the files of their connected peers and vice versa. The peers can not access each others files unless they have authorized each other.](image)

However peer-to-peer can be used in a more efficient way. By just connecting two peers directly potential bandwidth is wasted because the download speed of one user is limited the upload capacity of another. This is showed by studies of alternate peer-to-peer technologies such as BitTorrent\(^2\) (Arthur & Panigrahy, 2006). The reason why BitTorrent is more efficient than DirectConnect is that rather than sending every bit of information in a sequence from a single source, small parts are downloaded in parallel from many nodes on the network.

So being a superior technology in some aspects, another approach could be to set up a private BitTorrent tracker and start distributing torrent files for friends and colleagues that way, but this is by no means an easy nor optimal approach since a server to host the tracker and someone with the technical know-how is needed. This would also open up for security related issues since it would mean one point of failure for distribution of new files, and the possibility of peer lists that might end up in the wrong hands. It would probably be easier and faster to just email an attachment.

---

\(^1\) DirectConnect, http://dcplusplus.sourceforge.net/

But with all these applications and the different technologies behind them available, why did we want to create another way to share files? Well, since private file sharing has received little attention from the research community (Rogers & Bhatti, 2007) and there are few applications specialized in this field, we believe we can make a valuable contribution to the community of research and open source by reviewing the state of affairs and implementing our own application. We can explore possible solutions and also incorporate whatever features we find desirable, others may then hopefully use our results and continue our work.

1.2 Purpose

The purpose of this thesis is to research and propose an IT artifact to prove that it is possible to implement our concept of an application that allows private file sharing between peers in a secure, fast and simple way using a distributed file-system through a decentralized peer-to-peer network overlay.

1.3 Method

Since we wanted to deliver an IT artifact juxtapositional to this thesis we chose a Design and Creation approach (Oates, 2006). Vaishnavi and Kuechler (2004) describes this process as five steps;

- Awareness.
- Suggestion.
- Development.
- Evaluation.
- Conclusion.

Awareness is the recognition of a problem and suggestion means the movement from curiosity about the problem to offering a tentative solution. Development is where the idea is implemented. Evaluation means reviewing the developed artifact and tries to determine its worth and possible deviations from the expectations. The conclusion is where the results are consolidated, the knowledge gain is identified and possible unexpected outcomes that cannot be explained are identified for possible further research. These phases are not followed in a step-wise fashion, instead they form a more fluid iterative cycle.

To gain awareness, we first had to assert what had been done before us. We found only one prior study related to our subject, Rogers and Bhatti (2007), so we also conducted our own to ensure we were up to date on recent developments. Articles concerning specific peer-to-peer file-system applications proved hard to find by searching the databases randomly, such as typing "peer-to-peer" in Scholar¹. Instead we first used a list of file system projects on Wikipedia² to get an idea of what to study in greater detail. The results of the studies helped us conceptualize our goals. We present the projects most similar to ours later in the thesis.

¹ Google Scholar - Stand on the shoulders of giants, http://scholar.google.com
² Wikipedia list of file systems, see Appendix 1
For the implementation we chose an agile development method, namely extreme programming (XP). In the beginning of the project, we had high level goals but vague requirements. Due to these circumstances we faced problems when estimating the size and magnitude of the implementation. The probability that changes would emerge in the requirements was very high and we deemed a methodology with a short iteration cycle more fitting instead of a strict one. According to Beck and Andres (2005) some reasons to choose XP, depending on situation, are test-driven development, pair programming, small increments and it being suited for small teams.

We found these reasons and the overall aim for simplicity fitting for the development situation on this project and we therefore chose to use XP when developing the prototype even though XP has been debated and criticized since it was first introduced (Lindstrom & Jeffries, 2004). Some concerns are about pair programming, when for example the two programmers does not want to work together problems may arise, this is not a problem for us though since we have no problems working together.

More criticism according to Stephens and Rosenberg (2003) are e.g. a lack of structure and necessary documentation, insufficient design and increased possibility of scope creep. We feel that these critiques might be true but at the same time we believe they are more likely to occur on bigger projects than smaller ones like ours. Therefore, we believe the pros outweighs the cons and XP in this case is a good choice as a development methodology.

1.4 Delimitations

The time we have at our disposal for this thesis is not enough to complete a fully usable software, therefore the application is only a prototype with limited functionality as a proof of concept. Security is an important aspect of our concept but no cryptography technology is implemented in the prototype for the same reasons. However, judging from past experiences we believe it would be easy for us to implement such features modularly in future revisions. We also decided to exclude any form of version handling of files although this would be interesting to test in a peer-to-peer environment. Lastly, due to the limited functionality, we did not test the system in any large scale or perform any usability studies.

1.5 Thesis outline

This chapter has explained the background of the project. Chapter two gives detailed information about techniques and definitions on which this project is dependent, it also contains the survey results. Chapter three describes the concept in detail and chapter four the development of the prototype. A conclusion and discussion is presented in chapter five.
2 Definitions and dependencies

The goal of this thesis is to explore and demonstrate our concept of a private peer-to-peer file sharing application with a virtual file-system interface, therefore the techniques described in this chapter are all necessary and important for the reader to understand, the concept is explained in detail in later chapters.

2.1 Network protocols

The term *protocol* here refers to data communications procedures and is a kind of agreement about the exchange of information in a distributed system. A protocol is constructed formally much like a language with a *syntax*, *grammar* and *semantics* (Holzmann, 1990).

- It defines a precise format for valid messages, just like the dots and dashes of morse code.
- It defines the procedure rules for the data exchange, e.g. how to start and end conversations.
- It defines a vocabulary of valid messages that can be exchanged and their meaning.

Protocols have been used in long-distance communication systems throughout human history, first with fire and then electricity (e.g. the telegraph) but as is noted by Holzmann (1990), until recently there was always human operators who could be relied upon to make common sense decisions to resolve unexpected problems. For that reason, when machines communicate the grammar of the protocol must be logically consistent and complete, there can be no ambiguous use of terms.

![Figure 3: Depicting the layered communication of the OSI model.](image)

Since the medium of exchange can vary a lot, wired or wireless, there was the problem of bridging networks and the different types of hardware. One solution to this is illustrated by the OSI model, as can be seen in Figure 3, where protocols or protocol suites operate in different logical layers. The lowest layer is concerned with the electric or electromagnetic link.
between devices and the top layer is concerned with connections between applications. Units of data in one layer is always encoded into a datagram of the layer beneath.

That makes the underlying network structure opaque so that for example the application layer protocols are only concerned with other application layer protocols, e.g. a web server does not care whether the browser client is connected via transatlantic fiber optic cables or via satellite links.

The protocol suite used to build and connect networks into our grand Internet is called TCP/IP, Transmission Control Protocol/Internet Protocol. In Figure 4 we show how it relates to the OSI model. Aside from TCP and IP, some other common protocols of the suite include; HTTP, Hypertext Transfer Protocol used for the World Wide Web to define browser access to data on web servers. FTP, File Transfer Protocol is used to transfer files from and to file servers. The three common mail protocols are also part; POP, Post Office Protocol, IMAP, Internet Mail Access Protocol and SMTP, Simple Mail Transfer Protocol.

![Figure 4: The TCP/IP suite mapped to the OSI model.](image)

All the protocols discussed in this thesis are based on TCP/IP, or more specifically on the transport layer protocols TCP and UDP, Universal Datagram Protocol. The difference lies in TCP monitoring connections and therefore guaranteeing that sent messages are received if a connection is established, much like a regular phone call. UDP on the other hand is best described like the postal service where you can only hope your package reaches its destination, although it usually does.

### 2.2 Darknet and private peer-to-peer

We define darknet as a collection of networks and technologies used by private groups of people communicating or otherwise sharing digital content without the possibility for outside parties to supervise the transactions. Such a darknet does not have to be a separate physical network. It is, in this paper, rather used to describe an application and a protocol layer riding on top of existing networks.
Others have also defined darknet albeit with different intentions, and the term seems to have evolved into denoting illegal file sharing in general. Biddle et al (2004) defines darknet in their paper *The Darknet and the Future of Content Distribution* as an idea based on three assumptions (the term *object* is here interchangeable with *file*):

1. Any widely distributed object will be available to a fraction of users in a form that permits copying.
2. Users will copy objects if it is possible and interesting to do so.
3. Users are connected by high-bandwidth channels.

The darknet is then the distribution network that emerges from the injection of objects according to assumption 1 and the distribution of those objects according to assumptions 2 and 3.

Their reasoning is logically stringent but we can not agree on this definition since it is only concerned with mass scale distribution and we see no reason why darknet should not apply to a smaller scale as well, such as those emerging from social networks.

The authors further explain their intention as such:

“The term "widely distributed" is intended to capture the notion of mass market distribution of objects to thousands or millions of practically anonymous users. This is in contrast to the protection of military, industrial, or personal secrets, which are typically not widely distributed and are not the focus of this paper.”

While we find the intention is clearly stated and understandable we still would like to argue that it is unnecessary to limit the scope of the definition in this regard. Further, our focus in this thesis is what they termed *personal secrets*, but we feel no need to exclude other notions of information sharing from the definition, indeed that is why our definition is not concerned with intentions at all.

Biddle et al further state that darknets adhere to five infrastructure requirements:

- **Input**, facilities for injecting new objects into the darknet.
- **Transmission**, a distribution network that carries copies of objects to users.
- **Output**, ubiquitous rendering devices, which allow users to consume objects.
- **Database**, a search mechanism to enable users to find objects.
- **Storage**, e.g. a caching mechanism that allows the darknet to retain objects for extended periods of time.

We agree on these requirements to define a general file sharing network, however we cannot see any motivation for such a network to be called dark. While not explicitly expressed, we suspect the authors use the term *dark* to imply possible illegal activities within such a network. However the important insight of Biddle et al is that it does not matter if darknets are technologically isolated since they are socially connected. Thus one might talk of The Darknet
as if there existed such a network on a global scale, because users often belong to more than
one network.

We propose that privacy should also be a requirement for such a network for it to meet the ex-
pectations to the dark part as we perceive it, as an impenetrable black box, "under the radar" and away from "prying eyes". Privacy could be ensured by either anonymity or trust.

2.2.1 Anonymity

We define anonymity in accordance with ISO/IEC 15408-2 (ISO, 2005):

“Anonymity ensures that a user may use a resource or service without disclosing the user’s identity.”

Identity in regards to computer networks would be IP-addresses, usernames, and such. How-
ever there will always be a risk factor involved since absolute anonymity can never be guaran-
teed by any system. We simply cannot know what technologies there will be tomorrow to e.g.
break encryption.

In the field of peer-to-peer systems anonymity can be divided in categories (Xiao, 2007):

- **Publishing anonymity**, users create something without being discovered. Also called censorship-resistance.
- **Initiator anonymity** (sending) and **responder anonymity** (receiving) usually merged to form **mutual anonymity** in which users are protected from other entities during mes-
sage delivery.
- **Complete anonymity**, would be all of the above together with cryptographic protection of the transmissions themselves.

2.2.2 Trust

Trust can be defined using Merriam-Websters\(^1\) definition 1 a:

“Assured reliance on the character, ability, strength, or truth of someone or
something.”

This means one have to assure the ability of all invited parties not share sensitive information with others. Of course it implies a subjective sliding scale depending on situation, you might for example trust your spouse to take good care of the children but not to choose wallpaper for the bedroom. Regardless there should be an infrastructure there to support this approach to privacy.

Finally, since we intend darknet to be a broad term, we find that private peer-to-peer network more accurately describe our work in this thesis. The term is defined by Rogers and Bhatti (2007) as follows:

\(^1\) Merriam-Webster Dictionary, http://www.merriam-webster.com
“We define a private peer-to-peer network as an Internet overlay in which the resources and infrastructure are provided by the users, and new users may only join the network by personal invitation.”

This definition excludes many online social networks and media sharing websites because of their heavy reliance on public servers, however it does not therefore imply decentralization, as long as access to such servers is also restricted to invited users and the servers themselves are owned by the same users. With this definition private peer-to-peer can be viewed as a subset of darknet since we argue that a darknet does not have to be a peer-to-peer network.

2.3 BitTorrent

BitTorrent is a peer-to-peer file sharing protocol and the most popular one according to a report by Schulse and Mochalski (2008). In the same report one can read that BitTorrent takes up a substantial amount of all the internet traffic in the world, between 27 and 55 percent depending on geographical location. This makes BitTorrent not only the most used protocol for file sharing, but of all application layer protocols.

The first implementation of BitTorrent was released in July 2001 (Cohen, 2001). Bram Cohen is the creator of the protocol and its initial specification. After more development time BitTorrent was officially unveiled at CodeCon 2002, which was an event organized by the same man (Piccard & Sachs, 2006). The development of the protocol has since then come a long way and has in its current state incorporated many features to heighten performance and add functionality for the users (BitTorrent, 2010).

2.3.1 Functionality

When BitTorrent was first introduced it had a slightly different and rather innovative approach to file sharing when compared to other methods (Piccard & Sachs, 2006). Instead of downloading data from a single source BitTorrent could use several in parallel to increase performance and reduce vulnerability of the network.

The now almost classic approach to BitTorrent is to set up a special server called tracker which, as the name suggests, tracks users who are transferring files of different kinds (ibid). A user who wants to share certain files creates a torrent which contains meta-data about trackers, what files and folders are included and size of the specified files (Cohen, 2008). The newly created torrent is significantly smaller than the files it describes, usually only a few kilobytes. This torrent can then be uploaded to an index site for other people to access. Note that an index site is merely a web server listing torrent files and that it is different from a tracker server that instead coordinates communication. When the creator then loads the torrent file into a BitTorrent client, the files described by it is immediately available.

To initiate a download of the files described by a torrent, users first download the torrent from an index site and opens it in their BitTorrent client which then connects to the trackers specified in the torrent and acquires a list of peers that are sharing the same files. The tracker returns a random subset of existing clients, this is number is usually 50 (Arthur & Panigrahy, 2006). After this the client connects to the peers and requests data. The files are distributed in their own network as a number of independent data blocks, known as pieces.

Usually a downloader is referred to as a leecher and an uploader as a seeder. The network of seeders and leechers combined is called a swarm. A client can share individual pieces it has
fully downloaded with the swarm even if it has not finished downloading the entire file. This approach allows for parallelism that is impossible if entire files are treated as atomic blocks. By using this technique the bandwidth required for file transfers is spread among more peers than just the seeders which helps to speed up the transfer and ease the load on the seeders, this is exemplified in Figure 5. When the download is complete seeding is initiated.

While either download or upload occurs, the client can request new peers if that number falls below 20 (Arthur & Panigrahy, 2006). These requests also includes statistical information about the clients connection, e.g. how much data have been uploaded and downloaded. With this information the tracker can e.g. keep a log of the happenings, keep track of ratios for the users or notify the administrators of the tracker about suspicious behavior by the peers (Piccard & Sachs, 2006). The ratio of up- and downloading is an important factor for users to keep seeding and not just do a so called hit n’ run, where the torrent file is deleted from the client when the download is finished. This behavior is also refereed to as leeching or freeriding (Rogers & Bhatti, 2007).

It is possible to share using BitTorrent without using a tracker or even a torrent file. An extension to the BitTorrent protocol makes this possible by implementing a distributed hash table (DHT) based on Kademlia (Piccard & Sachs, 2006). A DHT, provides a lookup service that is similar to how a hash table works, pairs of (key, value) are saved in the DHT. Any given node in the network can request data or find new nodes by retrieving the value from a given key. Kademlia is the DHT used by BitTorrent. It specifies the network structure and how the interchange of information through node lookups works (Maymounkov & Mazières, 2002).

The communication uses UDP and each node in the network uses a unique identifier, such as a hash value generated from the data. This identifier does not only have the purpose of identification since the algorithm also uses it to locate values. When the algorithm is searching for values it needs to know the key and can then search through the network. This is done in several steps where each step will find nodes that are closer to the key until the last contacted node returns the value or no value is found.
Trackerless BitTorrent does not use centralized trackers, the peers does however act as simple trackers on their own and transmit peer lists. These data transfers does not offer any kind of statistical information about upload and download. The principle of trackerless BitTorrent, as shown in Figure 6, starts with the creation of a torrent file that is designed for trackerless connection. Regular torrent files contains an URL to a tracker, trackerless on the other hand use a DHT-URL. The peer uses that URL as an entry into the DHT network and is from that entry able to search through the network for specific files and connect to others peers that share those files.

![Figure 6: One example how BitTorrent can work without a tracker. The user uses Peer A as an entry to the DHT network and can then query other peers.](image)

### 2.4 File systems

For most users the file system is the aspect most visible of an operating system, OS. The file system provides a storage mechanism and access to both the data and programs of the OS, but also all the users of the computer system. A file system can usually be divided into two parts. The first one is a collection of files, which each stores related data. The second part is a directory structure, which is used to organize and provide necessary information about all files located on the system (Silberschatz et al, 2005)

#### 2.4.1 Files

For ease of use for us humans the OS provides a uniform logical view of information storage (Silberschatz et al, 2005). The OS abstracts from the physical properties of its storage to define a logical storage unit, the file. A file is a collection of information that is related and recorded. Usually, files represents programs and data, and a file is a sequence of bits, bytes, lines or records. The meaning of the data is defined by the file's creator and user. A file is therefore extremely general. Files always have a certain defined structure, e.g. a text file is only a sequence of characters,
on the other hand, an executable file is a series of structured code sections that can be brought into memory and the instructions can be executed in a useful manner (ibid).

Attributes
As described by Silberschatz et al. (2005) a file always has certain attributes, e.g. it has a name by which it is referred to, usually a name is a string of characters. Files can have many different attributes, but generally the attributes consists of these:

- **Name**, the only information kept in readable form for humans.
- **Identifier**, this is a unique tag which identifies the file within the file system, the identifier is non-human readable.
- **Type**, information needed for systems that support different types of files.
- **Location**, where a file can be located.
- **Size**, current size.
- **Protection**, access-control information that determines who can do what with the file.
- **Time, date and user identification**, information about e.g. creation, last time used and last time modified.

File operations
For us to properly be able to define a file, the operations that can be performed on files needs to be considered (ibid). The OS can, among other operations, make system calls to create, write, read, reposition, delete and truncate files:

- **Create**, in this operation space is allocated for the file and an entry is created in the directory for the newly allocated file.
- **Write**, to write a file the system sends a call specifying both the name of the file and what information to write.
- **Read**, to read from a file, the system makes a call that specifies the name of the file and where, in memory, the next block of that particular file should be put.
- **Reposition**, the directory is searched for the appropriate entry and the current file-position is repositioned. This operation is also known as file seek.
- **Delete**, to delete a file the directory is searched for the file specified. When found, all related space is released, so it can be reused by other files. The entry for the directory is also deleted.
- **Truncate**, this operation is similar to delete, but only the content of the file is deleted, the attributes are saved. The space of the deleted content is released for other files.

These are the minimal set of required file operations and can be combined to create new operations, e.g. a copy of a file can be created by creating a new file and then reading from the old
and writing to the new one. Files can also be locked to prevent other users or programs to ma-
nipulate the same files.

2.4.2 Directories

To organize the sometimes huge amount of files in a system, one way can be to invoke the es-
tablished concept of directories (Silberschatz et al., 2005). Each volume that contains a file
system must also contain information about the files in the system. This information is kept in
entries in a device directory, more commonly known as a directory. The directory can be
viewed as a symbol table that can translate file names into their directory entries. We want to
be able to insert, delete, search and list entries in directories. There are several schemes for
this purpose. The simplest one is the single-level structure where all files can be found in the
same directory. This is easy to understand but comes with heavy limitations once the number
of files and users grow. A file must have a unique name and conflict arises if two users tries to
save different files that have the same name.

A two-level structure tries to solve the problem with name collision by giving each user of the
system a personal user file directory. This also comes with disadvantages; the problem of
finding a certain file in a large amount of files still exists and the two-level structure isolates
the users from each other. While this may have advantages in some situations this approach
does not allow users to cooperate on a task or access each others files.

The most common directory structure is the tree. The tree has a root directory and all files
have a unique path names, extending outwards like branches where the files themselves are
the metaphorical leafs. A tree directory contains a set of files or sub directories. File names
only has to be unique in regards to the directory in which they reside, not to the tree in gener-
al. Directories are also files but are treated in a special way since they are containers for other
files or folders.

2.4.3 File-system mounting

Like a file needs to be opened before it can be used, a file system needs to be mounted (ibid).
The mounting procedure starts by the OS being given the name of the device, such as a hard
drive, and the mount point, which is the location within the file structure where the file system
will be attached. After this the OS needs to verify that the device contains a valid file system.
The OS asks the device driver to read the device directory and verify that the directory has the
expected format. Finally, the OS makes a note in the directory structure that a file system is
mounted at that specific mount point.

2.4.4 Protection

Information stored in computer systems needs to be kept safe from physical damage and pro-
tected from improper access (Silberschatz et al., 2005). The need for protection is a direct res-
ult from the possibility for users to access the files, controlled access is wanted. The most
common approach to provide this protection is to make access dependent on the identity of
the user. The most general scheme to this is to have an access-control list associated with files
and directories. The list specifies user names and what access each user has.

When a user makes a request for a file the OS checks the list to see if that particular user has
access or not. If not, a protection violation will occur and the user is denied access. Access-control lists main problem is their length and to create a list like this may be an unrewarding task. This can be avoided, to some degree, since many systems recognizes three standard classifications of users:

- Owner. The creator of the file is the owner.
- Group. A set of users that are sharing the file and need similar access is called a work group or simply group.
- Universe, all other users in the system.

Another approach is to give files a password, this can however be a tedious activity if there are many passwords for different files, and if there exists only one password the security may be compromised if this password is discovered by unauthorized entities (ibid).

2.4.5 Concrete and virtual file systems

Most of the file systems available today gives access to physical sectors on an underlying data storage mechanism (Silberschatz et al., 2005), these file systems are categorized as concrete since they are close to the hardware. Common sizes of the sectors includes 512 bytes, 1 and 2 KiB and the size is fixed. The file system has the responsibility to organize the sectors into directories and files, keep track of unused sectors and which sectors belongs to which file. A number of sectors combined form a so called block, which is how the file systems usually access data.

On the other hand, a virtual file system works as an abstraction layer above a concrete file system. The reason why a virtual file system is needed is to allow applications to access different types of concrete file systems in a uniform way. By using this method an application can access local as well as network storage without concern of the difference between the two sources.

2.5 Distributed file systems

A distributed file system (DFS) allows access to files from different hosts via a computer network (Silberschatz et al., 2005). By using a DFS it is possible for multiple users on multiple machines to share different files and resources for storage. Nodes that are connected to the network does not have direct access to the underlying data block storage but interact using a protocol. This makes it possible to restrict access depending on e.g. access lists on both server and client depending on how the protocol is designed.

A DFS should appear to its clients as a conventional, centralized file system. The locations of the servers and storage devices should be made invisible for the user. This means that the client interface of a DFS should not distinguish between local and remote files. The DFS has the responsibility to locate the files and transport the data.

2.5.1 Performance measurement

The most important performance measurement of a DFS is the time that is needed for service requests to be satisfactory (ibid). In a conventional system, this time only consists of disk-access and a small amount of processing by the CPU. In a DFS on the other hand, additional
overhead is attributed to the DFS structure. The overhead includes the time to deliver the request to the server, as well as the time for the response from the server. In addition, for each direction, to the transfer of the requested information, the client must also process the communication protocol software.

2.6 Survey results

In this chapter the results of the survey *How to Disappear Completely: A Survey of Private Peer-to-Peer Networks* made by Rogers and Bhatti (2007) are briefly presented alongside with the results of our own survey. *How to Disappear Completely* focus on private peer-to-peer applications while our own survey mostly focuses on peer-to-peer file system applications and also is up to date with recent developments. They therefore complement each other and provide a solid foundation as we are looking to combine these two aspects. A summary of their findings can be found in Table 1 below where they studied several different applications and classified them by three factors: scale, visibility and centralization.

<table>
<thead>
<tr>
<th>Name</th>
<th>Scale</th>
<th>Visibility</th>
<th>Centralization</th>
</tr>
</thead>
<tbody>
<tr>
<td>DirectConnect</td>
<td>Local</td>
<td>Group</td>
<td>Central server</td>
</tr>
<tr>
<td>Groove</td>
<td>Local</td>
<td>Group</td>
<td>Dedicated relays</td>
</tr>
<tr>
<td>PowerFolder</td>
<td>Local</td>
<td>Group</td>
<td>Members may act as relays</td>
</tr>
<tr>
<td>Shinkuro</td>
<td>Local</td>
<td>Group</td>
<td>Members may act as relays</td>
</tr>
<tr>
<td>WASTE</td>
<td>Local</td>
<td>Group</td>
<td>Members may act as relays</td>
</tr>
<tr>
<td>Easter</td>
<td>Local</td>
<td>Friends</td>
<td>Email servers</td>
</tr>
<tr>
<td>Alliance</td>
<td>Flexible</td>
<td>Friends of friends</td>
<td>Decentralized</td>
</tr>
<tr>
<td>Cryptic6</td>
<td>Flexible</td>
<td>Friends of friends</td>
<td>Decentralized</td>
</tr>
<tr>
<td>Galet</td>
<td>Flexible</td>
<td>Friends of friends</td>
<td>Decentralized</td>
</tr>
<tr>
<td>GNUnet</td>
<td>Flexible</td>
<td>Friends</td>
<td>Decentralized</td>
</tr>
<tr>
<td>Turtle</td>
<td>Flexible</td>
<td>Friends</td>
<td>Decentralized</td>
</tr>
<tr>
<td>anoNet</td>
<td>Global</td>
<td>Friends</td>
<td>Decentralized</td>
</tr>
<tr>
<td>Freenet0.7</td>
<td>Global</td>
<td>Friends</td>
<td>Decentralized</td>
</tr>
<tr>
<td>CSpace</td>
<td>Global</td>
<td>Friends, DHT</td>
<td>Decentralized(DHT)</td>
</tr>
<tr>
<td>Retroshare</td>
<td>Global</td>
<td>Friends of friends, DHT</td>
<td>Decentralized(DHT)</td>
</tr>
<tr>
<td>Octopod</td>
<td>Global</td>
<td>Group, DHT</td>
<td>Decentralized(DHT)</td>
</tr>
</tbody>
</table>

*Table 1: A classification of file sharing applications in regard to three axis. Table by Rogers & Bhatti (2007).*

Each of the factors considered involves a trade-off between ease of use, scalability and privacy. We used this survey as a starting point for the requirements for the concept, as well as for our own survey in which we tried to find applications that were closely related to the ideas we had for the concept. The information in our survey consists of the differences, similarities and why the existing solutions are not optimal for our problem. A quick summary of the applications we chose to look closer at can be found in table 2. They are not all file system applications, instead they have interesting ideas and solutions to security issues concerning file sharing.
### Table 2: A summary of the applications in our survey.

<table>
<thead>
<tr>
<th>Name</th>
<th>Scale</th>
<th>Visibility</th>
<th>Centralization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infinit</td>
<td>Local</td>
<td>Group</td>
<td>Central server</td>
</tr>
<tr>
<td>ColonyFS</td>
<td>Flexible</td>
<td>Friends of friends</td>
<td>Decentralized</td>
</tr>
<tr>
<td>Dust FS</td>
<td>Flexible</td>
<td>Group</td>
<td>Decentralized, server as relay</td>
</tr>
<tr>
<td>Dropbox</td>
<td>Flexible</td>
<td>Friends</td>
<td>Central server</td>
</tr>
<tr>
<td>OneSwarm</td>
<td>Global</td>
<td>Friends of friends</td>
<td>Decentralized</td>
</tr>
<tr>
<td>Freenet</td>
<td>Friends</td>
<td>Friends</td>
<td>Decentralized</td>
</tr>
</tbody>
</table>

2.6.1 Infinit

Infinit\(^1\) is an interesting approach to file sharing and backup. It claims to be a file system application that can enable users to store, share and access files in a safe way from any device connected to the internet. It is not yet available to the public and little information is published, it is therefore hard to tell more about the functionality since it is not exactly clear what the features are. It seems certain though, that it is a centralized system. Apparently there will be no limitations of the storage on the server, if this is true or not only time will tell. BitTorrent is not utilized. We still wanted to mention it if it gets further developed in the future.

2.6.2 Colony FS

Colony FS\(^2\) is a distributed file system that claims to put emphasis on anonymity and security inspired by the movement of ants. Not much is known about the underlying techniques due to the fact the project was cancelled before the release of a public working system. It is clear though that it did not implement BitTorrent for file transfers.

2.6.3 Dust FS

There is not much information available on their website, but it is clear that Dust FS\(^3\) is a file system application for Linux which uses FUSE, File System In Userspace. FUSE allows developers to implement a fully functional filesystem in a userspace program. Dust FS incorporates BitTorrent for file transfers. The choice to only have the application on Linux limits the number of possible users. Dust FS, much like Infinit, uses a central server to relay messages to the nodes. This gives the system vulnerability issues such as a single point of failure. The data transfer is not routed through the server since that would defeat the purpose of BitTorrent. It seems that also this project has been abandoned by its developers since there has not been any updates to the source code since April 2009.

2.6.4 Dropbox

With Dropbox\(^4\) you get a special folder on your desktop that has the ability to automatically upload files to the Amazon S3\(^5\) cloud service when files are dropped on it, hence the name.

---

1 Infinit, [http://i.nfin.it/](http://i.nfin.it/)
3 DustFS, [http://dustfs.zejur.net/](http://dustfs.zejur.net/)
4 Dropbox, [https://www.dropbox.com/](https://www.dropbox.com/)
This means that Dropbox is a centralized service and it has a storage limit of two gigabyte unless you are ready to pay for more.

2.6.5 OneSwarm

OneSwarm\(^1\) is a BitTorrent client that aims to give the user a higher sense of security and anonymity by, for example, implementing source-address rewriting to make it very hard to identify both sender and receiver. It is web based and thus not a file system application, but their ideas about anonymity addresses a lot of the issues about file sharing today and inspired us when considering safety measures for our own application.

2.6.6 Freenet

Freenet\(^2\) lets the users share files, browse and publish so called "free sites", which are web sites that only can be accessed through Freenet. The key in Freenet is anonymity. Freenet is decentralized service and has a "darknet" mode which makes it very hard to detect. Communication is encrypted and routed through other nodes. Users contribute to the network by giving bandwidth and part of their hard drive for storing files. Files are kept or deleted depending on how popular they are. Freenet is more than only file sharing, by incorporating "free sites" it can e.g. have chat forums. One interesting aspect is how the data is stored on your hard drive, it is encrypted and you really have no idea about its content and thus it is hard to say if you can be held accountable for that data or not.

---

1 OneSwarm, http://oneswarm.cs.washington.edu/
3 Concept

The vision of the final implementation is a file system application that allows users to move any file or folder onto a virtual drive, the information on the drive is then shared with all peer-to-peer connected nodes through an underlying network architecture. The purpose is to have a shared virtual space and also means the possibility to share portable applications, not only files. Both surveys in chapter 2.6 were used to concretize high level goals and from there we could derive that we would need something that could manage files in an easy manner, invite users and handle users in a growing network. The survey by Rogers and Bhatti (2007) uses three axis that are worth to consider, these are scale, visibility and centralization. The focus of the concept is private file sharing, thus the scale is smaller and visible to only friends. We also wanted it to be completely decentralized and user driven to promote rigid security.

The transfer of files is made possible by implementing the BitTorrent protocol. The application could work both as a group-based network or more securely by only allowing trusted nodes, it then acts as a friend-to-friend network. By only allowing trusted peers the network can be considered as a darknet. This means that peers in the network will only have access to your shared files if you have granted them access. To allow for both models gives users freedom of choice, but since we want to explore the possibilities of trust the group-based model will not be implemented.

When a user wants to share something they simply drags the file onto the virtual volume. The application has a local cache on which all files that are shared are stored. The application holds a list containing information about what files are shared, the list distinguishes between local files and files on the network. When a new file is added the file list is updated and a message is broadcasted to the connected nodes. They in turn read the content of the file list and add aliases of the files contained in the file list. This may not be feasible in a large public network, but since our aim is small friend-to-friend networks with a limited amount of users and files we do not believe this will be a problem.

Another implication of our approach is how versioning of files is handled. If file X is available to some peers and one of them changes the contents of this file the system will understand it to be a totally new file since the identifier is a hash value generated from the contents, regardless of it having the same name in human readable form. There are two solutions to this inconvenience, one is to implement a system to handle versioning and the other to simply regard files as read only objects. While the potential for a versioning system is very promising and indeed powerful we still believe that it falls out of scope for this thesis. Thus we will not allow for files to be tampered with, only written and deleted. In practice this means our system should not be used for collaborative work at this time, but publishing and archiving.

How nodes communicate and transfer meta-data to each other is made possible by implementing a new application layer protocol. The actual data transfer of shared files is not handled by this protocol, that is handled by BitTorrent.

To download a file the user can open it like any regular file, but if it is not in the local cache this action will initiate a download from all sources that holds this file. If the user does not want to open the file directly, she can choose to simply download it for later use. To speed up the transfer, nodes can be set to work as cache nodes, the optimal setting for a cache node is for it to have high upload capabilities. These nodes are only regular nodes which has been set to download and share all files on the network. This helps with initial seeding and to remedy
the problem of broadband connections that are asynchronous and often have a higher download capacity than upload.

Files and folders not stored locally all have a marking on the icon to distinguish them from files that actually are on the local computer. For the users to be able to search the network this is provided through Windows own search mechanism. The application has a system tray icon from which settings can be accessed to control connection specifics, friends and local cache size.

Security is provided by offering encryption of data transfers and routing of IP-addresses to make it hard, if not impossible, to track the source of a file. The user can also search for files and the application allows files to retain in the network as long as there exists at least one user with said file.

3.1 BitTorrent and file systems

The concept combines the BitTorrent protocol with file systems. The reasons why we chose to use BitTorrent instead of other technologies for file transfers are the following:

- Scalability, the technology itself is very much scalable. For example, adding new peers is an easy task.

- Widespread technology, as stated earlier BitTorrent takes up a lot of the data that is transferred on the internet. It is widely accepted and under constant development which means new features and even better performance in the future (BitTorrent, 2010).

- BitTorrent has proven that it can greatly reduce bandwidth issues since the needed bandwidth is distributed among the connected peers and not limited to a single source. This parallelism is the biggest reason to use BitTorrent.

- Different open source libraries gives the possibility to explore the concept on different platforms.

- Decentralization, as mentioned earlier it can be used without any kind of centralized server which suits the decentralized nature of the project.

Why we chose to create a file system application instead of e.g. a web based one consists of three reasons:

- Freedom, this way a user could even install a portable application on the virtual drive which would then be executable by any other user.

- Simplicity, what could be easier, from a user perspective, than implementing an application that works like plugging in a USB-stick? If you ever used a computer with Windows you will most likely be able to use this application without problems.

- There are very few file system applications focused on file sharing and BitTorrent. We wanted to really explore the possibilities of combining these techniques into one creative application.
3.2 User stories

The user stories we created and refined throughout the development process.

*Virtual volume*
User must choose a name for the volume.
User must choose drive letter for volume.
User can create an arbitrary amount of volumes.

*Participate*
User can share files with other users.
User can invite other users to a virtual volume.
User can receive invite, decline or accept.

*File operations*
Drag n drop procedure for operations.
User can delete files.
User can copy files to volume.
User can share files between volumes.
User can rename files.

*Manage settings*
User can change connection specific settings, e.g. upload, download.
User can manage local cache size.
User can manage friends.

3.3 File Register

The register is the internal data representation containing information about the file and folder structure. It can be manipulated and displayed as a virtual filesystem and is arranged logically in a non-binary tree structure. It is also used to generate a list of local files that can be shared to other peers. The tree structure can be represented in many different ways, the underlying relations are still the same. Figure 7 exemplifies this concept in a Windows file system.

Figure 8 shows how the file 'testfile.txt' is represented in the tree and the relationship between external and internal file meta data representation. Figure 9 shows how file data is stored in a dedicated cache directory under a hash identifier.
Figure 7: A file system can be structured as a tree. This example shows a simplified Windows file system.

Figure 8: The relationship between external and internal file meta data representation.

Figure 9: Data of a particular file is stored in a dedicated cache directory under a hash identifier.
3.3.1 File operations

When acted upon by the user, any activities in our mounted virtual filesystem will be relayed from Windows by the file system driver to the application which then does its own internal processing before sending any relevant data back.

Find files
The find file request gives a directory path and expects an attached list to be populated. Since the application internally represent the virtual file system of a volume by a file tree, one elegant way of finding a certain node is by traversing the tree and then, if found, access the children of that particular node. The application converts the file data of the children to a list and returns the result to Windows, which in turn, through the virtual volume, displays the directories and files. This is again exemplified in Figure 8.

Open files
To open a file means that Windows only needs to know that the file exists and is ready to be read by any associated application. Therefore the application merely asserts that the file exists in the internal file tree and also indicates to the file system driver if it is a directory or not. However it does not evaluate its whereabouts in the cache or network.

Read files
First the application tries to determine whether the file exists in the local cache or if it should be streamed over the network. If the file is tagged as local it means 100% of the file is available and so the stream is read from the cache identified by it's unique hash name. If not in the local cache a network search for the same unique identifier is initiated and if found a network stream is provided which tries to create a partial cache file for reading. As long as the offset and buffer size is within the current limits of the file, any application accessing the file will perceive it as no different from other files in the system.

Write files
Incoming write stream to open file will in practice result in a new file being created in the cache and the file register. Peers will also be notified of the new file. If a user makes changes to a file it means a different hash value is generated for it.

Delete files
The file will be deleted from the cache and from the register. The now updated file list is replicated to the other peers. This does not mean the file will also be deleted from the peers, only that it will no longer be available from this peer.

3.4 Communication protocol

The new communication protocol operates in the TCP/IP application layer (See Figure 4). The application listens on a TCP port and when it receives an incoming message it forwards the message and makes appropriate method calls. After the connection between two applications has been established, the peers can communicate by sending messages and file lists as stated below.
3.4.1 Messages

A message is a serialized object containing the message itself as a simple string and the IP address of the sender. A message can also be a response to an initial request, it can then also contain a file list or file hashes.

Depending on what the message contains different action is taken. A common but important situation is the first request for a file list which occurs whenever a new peer is invited. The new peer first makes a request to the user that invited her, the response contains the file list. When the new peer receives the response it adds the file list to its own network list and the system shows the files that are available. Important to note here is that the list that is sent only contains the files that are actually shared by the user providing the invitation. If this user has connections with other peers, their file lists are not sent. For the new user to receive the lists from the other peers, they need an invitation from each of the peers. This is reasonable if we want to follow the Friend-to-friend model. This measure does also provide a sense of trust between the peers.

3.5 Invite users

To invite a friend to join your network all you need to do is to create a link and send it to the person you want to share files with. A link in this case is an encoded string that contains the IP address. The invited user simply adds the link into the application and it decodes the string and connects to the IP specified. The IP is saved in a list.

3.6 Initiate download

When files are added by a peer a message is broadcasted to all peers connected. The message contains the hash value for the newly added files. The receivers use the hash value to initiate the download and since they already have the IP addresses stored the transfer itself should pose no problem since the combination of the two is all that is needed to locate the files. The downloads can be either automatic when new files are added or they can be manual as in trying to open a file.
4 Development

In this phase we built the prototype from the solutions and ideas crafted in the earlier phases. By using Design and Creation terminology (Oates, 2006), the prototype is an instantiation that demonstrates our concept. An instantiation is a working system that demonstrates how ideas can be implemented in a computer based system. We named the prototype Sub-Zero, the name is inspired by its possibility to "fly under the radar".

4.1 Delimitations for the prototype.

Due to the limited time we had at hand we decided early on to only create a prototype as a proof of concept. The following was decided as out of scope for the prototype:

- Routing of IP-addresses.
- Full encryption, this includes communication and local cache storage.
- The invitation and download procedures are simplified.

4.2 Language and platform

The development platform for this project is Microsoft .NET and the programming language is C#. We felt that a .NET application best suited our needs because of the versatility of the platform, but also because of our previous experience. To be able to work separately, when needed, we used Google Code to share and distribute updates for the source code.

4.3 Methodology

As stated earlier we chose to use extreme programming as system development method for this project.

4.3.1 Extreme programming

The main goal of Extreme programming (XP) is to have a methodology that organizes people to produce higher quality software more productively by countering the effects of changing requirements. The main idea is to have multiple short development cycles.

XP has a number of basic values, activities, principles and practices (Beck & Andres, 2005). The following paragraphs gives a brief introduction to XP.

Activities

The only truly important product of system development is code. Without it, no working product can be delivered. Testing is also an important activity, it is not possible to know if a function works without testing it. In XP projects, often two types of tests are used, unit tests and acceptance tests. Unit tests determines if a feature works as intended. Acceptance tests tries to verify the requirements as the programmers have understood them corresponds to the
customers actual requirements. We only used unit testing since the concept does not have any outside customers.

Values
XP favors collaboration between users and programmers, both verbal communication and feedback. It also encourages the developers to start with the simplest solutions first, more functionality can be added later. A main difference between XP and more conventional methodologies is the focus on designing and coding for the needs of today instead of tomorrow. Feedback is important in XP, it relates to different dimension of system development, e.g from the customer, the system and the team. Courage and mutual respect between team member are also important values.

Principles
The principles of XP are based on the values and are more concrete and are easier to translate to guidance in practical situations. XP sees feedback as very useful if done rapidly since feedback is critical for learning and making changes. Simplicity is a big part of XP, it rejects the idea to plan for the future, and instead focus on today. Finally, embrace change means exactly that, instead of working against it, embrace it.

Practices
What we found most interesting were the practices and they are mostly the reason why we chose XP. The practices includes pair programming, test-driven development, refactoring, small releases and coding standards.

4.3.2 Prototyping
According to Benyon et al. (2005) a prototype is a concrete representation of a system which is not fully developed. In the domain of system design the main distinguishing characteristic of a prototype is that it is interactive. Even if the prototype is just on paper something needs to happen when a user 'presses' a button, e.g in the form a post-it note.

There are two main kinds of prototypes: lo-fi and hi-fi. A lo-fi prototype is usually made of paper and can be produced quickly. It has a bigger focus on underlying design ideas, such as navigation and structure. On the other hand, a hi-fi prototype is produced in software which takes longer time. It is also more useful for detailed evaluation or usability studies. On this project we have created a hi-fi prototype. One reason to not use a lo-fi prototype is that the most interesting functionality in our application is not in the user interface, more so in the underlying technical aspects.

A prototype can also have different levels of functionality. Horizontal functionality shows the whole system but only includes general functions. Opposite to horizontal is vertical functionality which shows the full range of features for a small number of functions. Naturally there can be combinations of these and our prototype is.

To produce a prototype there are different methods, two well known are throw-away and evolutionary prototyping. With evolutionary you start with raw requirements with leads to a first version. As feedback and changed requirements emerge, this leads to new versions and more feedback. After a number of iterations the prototype will be a complete system.
In throw-away prototyping, prototypes are created but only as specification for the final product. In this way the quality of the final product is often higher because of the learning experience through the different prototypes. We chose to use an evolutionary approach because we do not believe that the development of this application will stop in the foreseeable future and therefore it did not seem like an optimal approach to use throw-away prototyping.

The main reason for us to choose to create a prototype is quite simple; we will not be able to deliver a completely working system with the time that has been given for the thesis.

4.4 Development libraries

Due to the technical nature of the project we quickly realized that there was no reason to invent the wheel again by implementing BitTorrent and writing a file system driver on our own. Therefore we went searching to find development libraries for this purpose. Basically, the only requirement we had for the libraries was the possibility to use them in Visual Studio. In the case of BitTorrent we found MonoTorrent, which to our knowledge, is the only one that is developed enough to be used in serious applications. Regarding the file system aspect we found only one library for Windows that could be used in Visual Studio, Dokan. The following paragraphs presents both of the libraries and how they helped us with the development.

4.4.1 MonoTorrent

MonoTorrent is an open source cross-platform library implementing the BitTorrent protocol (McGovern & Dufour, 2010). It is based on the Mono\(^1\) implementation of the .NET Framework, which means it can be compiled and executed on all major operating systems of today.

The aim of MonoTorrent is not to provide a graphical interface, instead the goal is to provide a rich API for developers to create a graphical user interface without them having to implement the BitTorrent specification. MonoTorrent has a small community of dedicated developers who keeps the library up to date.

MonoTorrent supports a lot of features e.g. encryption, DHT, peer exchange, metadata download, IP address white and black listing, fast resume and prioritized downloading. Our concept does not make use of any tracker and therefore peer exchange and metadata download is of great importance to us. Another interesting feature is the white and black listing of IP addresses, which is helpful when we want to restrict access to only specific nodes in the network.

4.4.2 Dokan

To create a new file system in Windows a file system driver needs to be developed. To develop a device driver that works in kernel mode is a very resource consuming task. By using the Dokan\(^2\) library instead we can create our own file systems without writing this device driver. Like MonoTorrent, Dokan is an open source library.

---

1 Mono, http://www.mono-project.com/
2 Dokan, http://dokan-dev.net/en/
Dokan consists of a kernel mode file system driver, dokan.sys, and a user mode DLL, dokan.dll. Requests on file operations e.g. read, write and create is sent to the Windows In/Out subsystem which forwards the request to the Dokan file system driver. By using the functions provided by the Dokan library, file system applications are able to make callbacks to the file system driver. The result of these callbacks is sent back to the user program. For example, an OpenDirectory request is sent to the Dokan file system driver, the driver will invoke the OpenDirectory callback provided by the file system application. The result is sent back to Windows Explorer as a response to the request. By doing this the Dokan file system driver acts a proxy between user programs and file system applications. One advantage of using this approach is that it allows developers to create file systems in user mode which is safe and somewhat easy to debug.

Dokan is not C# native and for us to use it for this thesis we needed a wrapper. Luckily for us, the developers of Dokan have created wrappers for several languages, including C#. The wrapper needed a little bit of tweaking due to its low priority for the developers, and therefore it was a bit outdated compared to the latest version of Dokan.

### 4.5 Class diagram

The class diagram, Figure 10, shows the classes and the methods in the prototype. It describes the structure and layout of the application and defines the relationships between the classes and libraries. We have tried to keep it simple and slimmed for better comprehension.
4.6 Results of the implementation

Each implementation iteration has focused on different aspects of the user stories that were found and refined during the analysis and design. The stories were ranked in order of importance and logical aspect, meaning what we would need to have in place before continuing the next iteration of development. During the implementation we went back to the analysis and design phase to work out a better solution for us to implement or simply to verify the requirements. In the beginning of the implementation the highest ranked user stories were the one concerning the virtual volume. With a demo application for Dokan it was logical to start with a hi-fi prototype, especially due to our lack of experience from the libraries.

Figure 11 shows how a Sub-Zero drive can be mounted or unmounted. It also shows a simple TextBox control which can display what files are shared and what files are being downloaded. The TextBox control also shows information about uploading and downloading. Figure 12 shows that the OS recognizes Sub-Zero as a regular drive. Invitation of friends or the use of received invitations is handled by clicking the Participation-button.
Figure 11: Simple layout for mounting a virtual drive.

Figure 12: Windows explorer showing the now mounted SubZero drive.
4.6.1 Sub-Zero Network Protocol

As stated earlier, the communication protocol, dubbed Sub-Zero Network Protocol, is a simple TCP based protocol. The application acts as both a server and a client and it can send and receive Sub-Zero Messages which are serialized objects of the `NetworkMessage` class. These messages contain either simple strings, file lists, IP addresses, hashes or a combination. The send message method in the `SZClient` class is the most important method. It takes the message in question and a list of IP addresses to send the message to.

After some tests to try to determine how to implement the protocol we chose to use the `TcpClient` and `TcpListener` classes instead of the more advanced `Socket` class due to the protocol being rather simple, in this regard the `Tcp` classes were more suiting. In the future however, we will most likely change this to the `Socket` class which also offers more functionality in regards to asynchronous data transfer.

4.6.2 Local cache

The local cache is in the prototype specified to a folder on the Windows C-drive called Sub-Zero. It acts as a single-level directory storing all files that are shared. Due to the tree-structure of the virtual drive it does not matter that all files are in the same folder, the internal tree keeps the folder structure intact. The files does not have their original names, instead they are given a hash name as a simple measure to reduce the impact of redundancy. A file can therefore exist in several directories on a virtual volume without consuming more physical space.

4.6.3 Tree implementation

As mentioned earlier in the thesis, the data structure for the file register is a non-binary tree. We were inspired by the work of Dan Vanderboom (2008) and his implementation.

The tree uses generics for its type to be easily distinguished. The code needed to make the Tree functional resides mainly in two classes: `TreeNode` and `TreeNodeList`. Each `TreeNode` object has links to both parent and children. If we for example add a child to node X, that child's parent needs to be set to X. This tampering with the `Add` method was easiest in a custom collection class, the `TreeNodeList`.

Next, we have the `TreeNode`. If the parent property is updated, that is if a node is moved to another part of the tree, we need to make sure that it gets removed from the parent's list of children, we also need to add it as a child to the new parent. The tree structure is not very useful unless it carries some kind of payload and, as mentioned, it is very handy to use generics to tell the compiler that we for example want to build a tree of ints, simply put; `Tree<int>`.

4.6.4 User invitation

The invitation of users is simplified in the prototype compared to the concept. To invite someone you need the IP of a user. The address is then added to the application by entering it into the 'Use Invitation' field. You will receive that users file list and can start downloading the files that are shared. If you instead want to invite someone the 'Create Link' button gets your IP address and pastes it into the field for you to copy and send to friends you want to in-
vite. The **InviteUser** form can be seen in Figure 13. All connected peers show up in the Text-Box control and the IP addresses of the peers are stored in a list for simple access.

![InviteUser Form](image)

*Figure 13: The user form with contact list and link input/output.*

### 4.6.5 Download initiation

In the prototype, data transfer is initiated automatically once new files are added. When a new file is added the application creates an internal Torrent and a message is broadcast to connected peers containing the hash value for current file taken from the newly created Torrent. The hash is collected by the application and seeding of the new file starts.

The receivers uses the hash value to initiate the download and since they already have the IP addresses stored the transfer itself is rather easy thanks to the power of Mono Torrent. The combination of address and hash value allows the peers to find each others files and download them. The method that downloads the files uses the hash value to collect the rest of the meta data, containing more detailed information about the file, from the sender, and can thereafter start the transfer from all users that shares the specific files.

### 4.7 Testing

In accordance to XP philosophy we wanted all code to pass unit testing (Beck & Andres, 2005). We used the built in unit testing in Visual Studio 2008 to assert that our methods actually worked and did what they were supposed to do. Any bugs found were considered and fixed either directly or in later iterations. A test project was added into the solution and for each class in our implementation a corresponding test class was created. In the test classes, by default, Visual Studio adds "Test" at the tail of **MethodName** and **ClassName**.
A generated test method for the method `CreateManager` looks like this:

```csharp
public void CreateManagerTest()
{
    Test target = new Test();
    string path = string.Empty; /\ TorrentManager expected = null;
    TorrentManager actual;
    actual = target.addToList(path);
    Assert.AreEqual(expected, actual);
    Assert.Inconclusive("Verify the correctness of this test method.");
}
```

The parameters needs to be instantiated with correct values for the test to work. When instantiated the test method has the following appearance:

```csharp
public void CreateManagerTest()
{
    Test target = new Test();
    string path = @"C:\SubZero\bin\Debug\t.torrent";
    TorrentManager expected = new TorrentManager(
        Torrent.Load(path),
        "C:\\Temp\\SubZero\\",
        new TorrentSettings()
    );
    TorrentManager actual;
    actual = target.addToList(path);
    Assert.AreEqual(expected, actual);
}
```

To run the tests, one way is to set the test project as startup project and then build, to only test a single method all you need to do is right click on the method itself and choose 'Run Test'. The tests will be executed and hopefully they are passed.
5 Discussion and conclusion

This thesis describes the research we have made in the file sharing field, our concept and the application we dubbed Sub-Zero that demonstrates this concept. This final chapter describes the problems, results, what we have learned as well as further work.

5.1 The field of file sharing

When we had stated our goals for this thesis we started researching the field of file sharing. We knew there were quite a few technologies for this purpose, but as the research went on we made daily discoveries about new techniques and we were astonished by the sheer number of ways you could share files. Soon we found our own path in this jungle and we hope we have adequately described the results of this endeavor.

5.2 Creation of the application

Through the creation of this application we have learned a lot about different techniques and had to deal with undocumented development libraries. This was not a problem with MonoTorrent thanks to help from the original developers and the community that has emerged around it. It was different with Dokan though, which does not have the same lively community as MonoTorrent. What this meant for us, was many hours of trial and error in the beginning to develop just simple functions and to understand the strange and non-helping error messages Windows gave us. This certainly enforced the idea about the importance of documenting both your code and the error messages it produces.

The Design and Creation methodology (Oates, 2006) proved to be a pleasant approach to work with this paper. It was not very clear to us from the beginning how to combine the technical aspect of developing an artifact with the academic requirements of a thesis paper. We found moving iteratively through the phases of awareness, suggestion, development, evaluation and conclusion do be as natural as can be expected and it has been a valuable reference framework. The idea of fluid iterative cycles correspond well to our way of thinking and we surely recommend this very much helpful methodology.

We are aware that XP says that if you need to document your code you have made it too complicated, and this is true, to some extent at least, but in this case the source code is very complex for big parts, and it would without any doubt have helped us very much in the development with proper API documentation of Dokan. On the other hand, all this trial and error have taught us a lot about this library and most likely helped to develop a deeper understanding which we might not have gained otherwise. We could tackle the initial issue from different angles and hopefully this resulted in a better and more sturdy application. These problems are best described in code and we therefore hope others will find our code useful in other endeavors.

Another area of system development which had a big impact on this project is requirements, or more correctly, changing and unknown requirements. From the beginning we had our goals clearly stated, though the more detailed requirements were vague at best. Throughout the development requirements kept changing when we made discoveries and faced new technical problems. Our choice of using XP payed off thanks to it being a methodology that is agile and more easily can coupe with change since it is not strictly planned like other methodologies.
such as waterfall and RUP. To our experience development with such methodologies, where the different phases are more static and the possibility of scope creep is more likely, requirement changes might have severe impact on budget and time frame, especially with waterfall.

We definitely find that XP can be a good methodology when the system in question is a bit smaller and does not contain an overload of user stories. With the increase of a systems size we can see a rising problem with difficulty of getting a good overview of the system. In that case a more thorough design phase and more documentation might be needed to confront the problem, but this is only our speculation.

During the development we also had chance to explore several other techniques within the .NET platform. The network protocol we developed makes use of some for us novel techniques and it was interesting to study how object states could be saved and transferred through a network and then be read on the other end. Since we did not have much previous experience of sockets and TCP classes, we only scratched the surface regarding this, we can see some great improvements for the application in the future just by optimizing our protocol.

From previous experiences we had worked more than once with binary trees, but not so much with non-binary ones, so it was worthwhile to read up on and implement one of our own supporting the file system.

Testing with the built in framework in Visual Studio is also a really good way to find out if your code does what you have intended. It is very easy to use, all it requires is that you add a test project into the projects solution and instantiate a few variables.

With everything considered, all these new experiences regarding the technical side of this thesis are somewhat complex and we feel that we have accomplished much during the limited time we had at hand when we started writing this paper.

5.3 The application

We have put a lot of effort into this thesis and the development of Sub-Zero. In the first chapter we wrote about the, according to us, three most important aspects about file sharing; speed, scalability and security. Although security never was a priority for the prototype it is still considered in the concept. This is only logical since we wanted a proof of concept. Scalability and speed is provided by the implementation of BitTorrent. Freedom is also an important factor which is provided by the file system aspect of the application.

Network vulnerability is minimized since there are no central servers which, if they go down, will break the network. Another advantage is that no information is stored on any servers that the users have no access to. But just as there are advantages with the decentralized approach there are also inherent shortcomings. The biggest problem with the decentralization is dynamic IP addresses. When someone is invited the IP address is used to connect and share file lists and when that computer is shut down it might not have the same IP the next day. With different addresses the users need to invite each other again.

A hybrid solution to this is to use a server that can keep track of users, this is close to a tracker though, and is not what we want since the aim is a pure decentralized system. This can also be seen as a tradeoff between usability and privacy. One idea we have to counter this is if some members of the network has a reliable IP address, the application can then remember the latest
connections and save those addresses for later sessions. In a group based network, only one member needs a stable address since the rest of the addresses can be learned from that member. On the other hand, in a friend-to-friend network this would mean that each member needs at least one friend with a reliable address.

In both cases, if a user fails to connect to earlier known nodes the need to trade updated addresses out-of-band arises, using for example email. If one wants to pursue the group-based model a small scale Distributed Hash Table for the group would probably be a feasible approach, but this claim needs to be tested first.

For other developers we can surely recommend to develop your own file system application using Dokan if you develop on Windows, although you will need to study this or other projects that builds on Dokan to be able to to something worthwhile with it. And if you for example want to create your own BitTorrent client MonoTorrent is a very versatile tool.

5.4 Further work

The most crucial and important part that we left out from the prototype is the one concerning security. Since the aim was to create a proof of concept security never was a priority even though it is an important aspect in the concept. We want to implement routing of IP addresses and encryption for both the Sub-Zero Network Protocol and the BitTorrent file transfers.

In regards to our protocol it would probably need to be streamlined and the TcpListeners and TcpClients be replaced with the more advanced Socket class, this would also allow for more asynchronous connections and better performance, this is probably very important when the network grows to achieve higher speed and less delays. This leads to another area we want to explore in the future; to test the efficiency and performance in a larger scale network. This would be very interesting to see what results we would achieve and what we would need to do to increase performance.

Most likely we would also need to make the application controls more user friendly. To achieve this the application would probably benefit from a smaller usability study by for example having a usability expert perform a heuristic evaluation. One area concerning this that could improve usability is to implement the system tray icon for quick and easy access to the settings, instead of having it like today where all controls are in a Windows form and are somewhat confusing.

These are all things we need to take care of before we can launch the application for public use. Since file sharing is always evolving and involves millions of people every day, it is exciting to have contributed to this area and just the idea that it may be used by people in the future feels both peculiar and very rewarding and certainly helps to keep us motivated to update Sub-Zero with more functions and better performance in the future.
5.5 Conclusion

The thesis describes the work we have done in the field of private file sharing by the Design and Creation method which we found to a natural and very helpful approach. For the development of the prototype we used XP as methodology. As stated earlier in this chapter we believe this was the right choice for this project since it was small in scale but with vague requirements. The application works well when considering the short development time and we would like to claim that we have reached the goal of the paper. The concept works in practice and we are pleased with the result. It is however hard to predict the actual performance in the real world when many nodes are connected and not simply the two computers we have been using in the development.

The most notable aspects of the concept are the virtual file system and usage of BitTorrent. This is what makes the concept go beyond simple file sharing. The Dokan implementation shows some very promising possibilities for the future, not only in this field, it is applicable to others as well since it can be a very powerful and versatile tool, even though somewhat problematic today with the current libraries available.

The research we made has hopefully contributed to form a clearer view of the field and we hope that the reader finds our work interesting and can use this paper as a platform for further developments in this area.
References


Electronic sources


Appendix 1 - File systems from Wikipedia


Distributed file systems

- 9P, the Plan 9 from Bell Labs and Inferno distributed file system protocol. One implementation is v9fs. No ACLs.

- Amazon S3

- Andrew File System (AFS) is scalable and location independent, has a heavy client cache and uses Kerberos for authentication. Implementations include the original from IBM (earlier Transarc), Arla and OpenAFS.

- Apple Filing Protocol (AFP) from Apple Inc.. AFP may use Kerberos authentication.

- DCE Distributed File System (DCE/DFS) from IBM (earlier Transarc) is similar to AFS and focus on full POSIX file system semantics and high availability. Available for AIX and Solaris under a proprietary software license.

- File Access Listener (FAL) is an implementation of the Data Access Protocol (DAP) which is part of the DECnet suite of network protocols created by Digital Equipment Corporation.

- Microsoft Office Groove shared workspace, used for DoHyki

- NetWare Core Protocol (NCP) from Novell is used in networks based on NetWare.

- Network File System (NFS) originally from Sun Microsystems is the standard in UNIX-based networks. NFS may use Kerberos authentication and a client cache.

- OS4000 Linked-OS provides distributed filesystem across OS4000 systems.

- Self-certifying File System (SFS), a global network file system designed to securely allow access to file systems across separate administrative domains.

- Server Message Block (SMB) originally from IBM (but the most common version is modified heavily by Microsoft) is the standard in Windows-based networks. SMB is also known as Common Internet File System (CIFS). SMB may use Kerberos authentication.

Distributed fault-tolerant file systems

- Coda from Carnegie Mellon University focuses on bandwidth-adaptive operation (including disconnected operation) using a client-side cache for mobile computing. It is a descendant of AFS-2. It is available for Linux under the GPL.
• Distributed File System (Microsoft) (Dfs) from Microsoft focuses on location transparency and high availability. Available for Windows under a proprietary software license.

• InterMezzo from Cluster File Systems uses synchronization over HTTP. Available for Linux under GPL but no longer in development since the developers are working on Lustre.

• Moose File System (MooseFS) from Gemius SA is a networking, distributed file system. It spreads data over several physical locations (servers), which are visible to a user as one resource. Works on Linux, FreeBSD, NetBSD, OpenSolaris and MAC OS X. Master server and chunkservers can also run on Solaris and Windows with Cygwin.

• Tahoe-LAFS is an open source secure, decentralized, fault-tolerant filesystem utilizing encryption as the basis for a least-authority replicated design.

Distributed parallel file systems

• Fraunhofer Parallel File System (FhGFS) from the Fraunhofer Society Competence Center for High Performance Computing. Available free of charge for Linux under a proprietary license. (High availability features are on the roadmap, currently only local file locking is supported)

• Parallel Virtual File System (PVFS, PVFS2). Developed to store virtual system images, with a focus on non shared writing optimizations. Available for Linux under GPL.

• Starfish is a POSIX-compatible, N-way redundant file system created by Digital Bazaar Inc. and published under a pseudo-open source license. Available for Linux and Mac OS. Windows support is available via Samba.

Distributed parallel fault-tolerant file systems

• CloudStore from Kosmix is a Google File System workalike, available under Apache License 2.0.

• Cosmos from Microsoft focuses on fault tolerance, high throughput and scalability. Only available for use in Microsoft Bing Datacenters. Used internally at Microsoft for storing and processing large, Terrabyte and Petabyte sized data sets. Processing over data in Cosmos is done using Dryad.

• dCache by Fermilab and DESY. A write once filesystem, accessible via various protocols (not mountable). Is available free of charge.

• ExaFS distributed file system from Exanet. Runs as part of ExaStore, a Linux based NAS solution that runs on commodity Intel based hardware, serving NFS v2/v3,
SMB/CIFS and AFP to Windows, Mac OS, Linux and other UNIX clients. Available under a proprietary software license.

- FS-Manager from CDNetworks focused on Content Delivery Network. Available for Linux under proprietary software license.

- Gfarm file system uses OpenLDAP or PostgreSQL for metadata and FUSE or LUFS for mounting. Available for Linux, FreeBSD, NetBSD and Solaris under X11 License.

- General Parallel File System (GPFS) from IBM. Support replication between attached block storage. Available for AIX, Linux and Windows. Symmetric or asymmetric.

- GlusterFS is a general purpose distributed file system for scalable storage. It aggregates various storage bricks over Infiniband RDMA or TCP/IP interconnect into one large parallel network file system. Released under GNU General Public License v3.

- Google File System (GFS) from Google focus on fault tolerance, high throughput and scalability. Only available for use through Google App Engine.

- IBRIX Fusion from IBRIX. Available for Linux under a proprietary software license.

- Lustre is a POSIX-compliant, high-performance filesystem originally developed by Cluster File Systems, and later acquired by Sun Microsystems in 2007. Lustre has high availability via storage failover, but multi-server RAID1 or RAID5 is still in the roadmap for future versions. Freely available for Linux under GPL. Versions for Solaris and Windows are also under development.

- MogileFS from Danga Interactive is not POSIX compliant, uses a flat namespace, application level, uses MySQL for metadata and HTTP for transport. Available for Linux (but may be ported) under GPL.

- OneFS distributed file system from Isilon. BSD based OS on dedicated Intel based hardware, serving NFS v3 and SMB/CIFS to Windows, Mac OS, Linux and other UNIX clients under a proprietary software license.

- Panasas ActiveScale File System (PanFS) from Panasas uses object storage devices. Available for Linux under a proprietary software license.

- PeerFS from Radiant Data Corporation focus on high availability and high performance and uses peer-to-peer replication with multiple sources and targets. Available for Linux under a proprietary software license.

- TerraGrid Cluster File System from Terrascale Technologies Inc implements on demand cache coherency and uses industrial standard iSCSI and a modified version of the XFS file system. Available for Linux under a proprietary software license.

- XtreemFS is a free and open-source (GPL) cross-platform file system for wide area networks. It replicates the data for fault tolerance and caches metadata and data to improve performance over high-latency links. SSL and X.509 certificates support makes XtreemFS usable over public networks. It supports also Striping for usage in a cluster.
In development:

- **WebDFS** An Open Source scalable, decentralized file store similar to MogileFS in function and purpose. Uses HTTP as the transport. Data is automatically and optimally re-arranged to accommodate the addition of new resources. The lack of central meta data management greatly simplifies deployment and use.

- **Ceph** from University of California, Santa Cruz which utilized entire block devices. Available for Linux under the LGPL. Merged for Linux kernel 2.6.34.

- **zFS** from IBM (not to be confused with ZFS from Sun Microsystems or the zFS file system provided with IBM's z/OS operating system) focus on cooperative cache and distributed transactions and uses object storage devices. Under development and not freely available.

- **GLORY-FS** also from ETRI. Very similar to the Google File System or Hadoop, but it is fully POSIX compliant. It is specially optimized for large scale web 2.0 content services. Version 2.5 is available for Linux via special technology transfer program provided by ETRI. Windows version is under development.

- **PNFS** (Parallel NFS) - Clients available for Linux and OpenSolaris and back-ends from Panasas, EMC Highroad and IBM GPFS

- **Parallel Optimized Host Message Exchange Layered File System (POHMELFS) and Distributed STorage (DST).** POSIX compliant, added to Linux kernel 2.6.30

- **Sector** from National Center for Data Mining. Sector is a high performance, scalable, and secure distributed file system. The Sector/Sphere project is developed at the National Center for Data Mining (NCDM) and is supported in part by the National Science Foundation. Available for Linux GPLv2.

**Peer-to-peer file systems**

- **CFS** is a read-only file system based on the Chord DHT

- **Cleversafe** uses Cauchy Reed-Solomon Information Dispersal Algorithms (IDAs) to separate data into unrecognizable slices and distribute them, via secure Internet connections, to multiple storage locations.

- **Infinit** is a large-scale peer-to-peer file system developed in C++ which enables users to both reliably and securely store their files in a location-independent and replicated way; and to share files with a controlled set of users, friends etc.

- **Ivy** is a multi-user read/write peer-to-peer file system. Ivy has no centralized or dedicated components, and it provides useful integrity properties without requiring users to fully trust either the underlying peer-to-peer storage system or the other users of the file system.

- **ColonyFS** emphasises anonymity, security and dependability, is written in Java and C#.